arXiv:2508.01554v1 [cs.CL] 3 Aug 2025

Are All Prompt Components Value-Neutral? Understanding the Heterogeneous Adversarial Robustness of Dissected Prompt in Large Language Models

Yujia Zheng[1*], **Tianhao Li**[1*],
Haotian Huang[2], **Tianyu Zeng**[3], **Jingyu Lu**[4], **Chuangxin Chu**[5],
Yuekai Huang[6,7], **Ziyou Jiang**[6,7], **Qian Xiong**[8], **Yuyao Ge**[7,9], **Mingyang Li**[6,7†]
[1]Duke University, [2]North China University of Technology [3]Hong Kong Polytechnic University [4]Australian National University [5]Nanyang Technological University [6]Institute of Software, Chinese Academy of Sciences [7]University of Chinese Academy of Sciences [8]Beijing Forestry University [9]Institute of Computing Technology, Chinese Academy of Sciences
`{yujia.zheng, tianhao.li}@duke.edu, mingyang2017@iscas.ac.cn`

Abstract

Prompt-based adversarial attacks have become an effective means to assess the robustness of large language models (LLMs). However, existing approaches often treat prompts as monolithic text, overlooking their structural heterogeneity—different prompt components contribute unequally to adversarial robustness. Prior works like PromptRobust assume prompts are value-neutral, but our analysis reveals that complex, domain-specific prompts with rich structures have components with differing vulnerabilities. To address this gap, we introduce PROMPTANATOMY, an automated framework that dissects prompts into functional components and generates diverse, interpretable adversarial examples by selectively perturbing each component using our proposed method, COMPERTURB. To ensure linguistic plausibility and mitigate distribution shifts, we further incorporate a perplexity (PPL)-based filtering mechanism. As a complementary resource, we annotate four public instruction-tuning datasets using the PROMPTANATOMY framework, verified through human review. Extensive experiments across these datasets and five advanced LLMs demonstrate that COMPERTURB achieves state-of-the-art attack success rates. Ablation studies validate the complementary benefits of prompt dissection and PPL filtering. Our results underscore the importance of prompt structure awareness and controlled perturbation for reliable adversarial robustness evaluation in LLMs. Code and data are available at https://github.com/Yujiaaaaa/PACP.

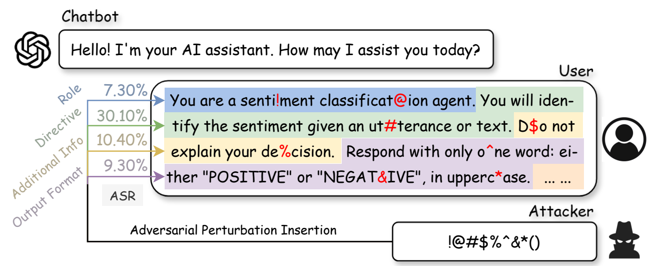

Figure 1: Inserting adversarial perturbations into different prompt components leads to distinct attack success rates (ASR).

1 Introduction

Large language models (LLMs) (Naveed et al. 2023; Zhao et al. 2023) such as DeepSeek, ChatGPT, and LLaMA-3 have demonstrated remarkable capabilities in a wide range of tasks (Thirunavukarasu et al. 2023; Zhang et al. 2025; Demszky et al. 2023; Hou et al. 2024), largely driven by in-context learning via carefully crafted prompts (Dong et al. 2022; Li 2023; Long et al. 2024; Mei et al. 2025). As these models become increasingly integrated into real-world applications, the robustness of their behavior under perturbed prompts is becoming a concern for both reliability and safety (Zhu et al. 2023b,a; Hu et al. 2024; Ghosh et al. 2025).

Recent studies have demonstrated that even subtle variations in prompt phrasing—such as typos, synonym substitutions, or formatting changes—can lead to significant drops in LLM performance, a phenomenon known as prompt sensitivity (Zhu et al. 2023b,a; Wang et al. 2023b). To evaluate this vulnerability, Zhu et al. (2023a) introduced *PromptRobust*, a benchmark targeting adversarial prompt perturbations, while Wang et al. (2023b) proposed *MTTM*, a metamorphic testing framework that applies structured perturbations to toxic inputs for testing content moderation systems. Although these efforts highlight the fragility of LLMs under adversarial prompting, they treat prompts as flat, undifferentiated text—**overlooking the functional structure within prompts that may contribute unequally to model behavior**.

We argue that **prompts are fundamentally compositional, consisting of multiple functional components**—such as task directives, role assignments, input-output delimiters, exemplars, and answer formatting instructions—each guiding LLMs to interpret and execute tasks more effectively. In practical scenarios, users often construct complex prompts by layering basic instructions with supplementary cues, such as specifying the task domain, assigning a role to the model (e.g., "You are a medical expert"), providing concrete examples, or defining output styles. Recognizing the diversity and importance of these components, Schulhoff et al. (2024) provide a comprehensive taxonomy that categorizes prompt elements into six types: *Directive*, *Examples*, *Output Formatting*, *Style Instructions*, *Role*, and *Additional Information*. To further streamline the taxonomy for our analysis, we merge *Style Instructions* into *Output Formatting*, as both primarily define the expected form or tone of the model's output. These components exhibit het-

*Equal contribution, co-first author.
†Corresponding author.

erogeneous adversarial robustness, meaning that some are more vulnerable to perturbations than others. To investigate this, we pose the following research questions: *RQ1*: How does prompt structure and component richness affect the effectiveness of adversarial perturbations? *RQ2*: How does the sensitivity of adversarial robustness vary across different prompt component types? *RQ3*: How do semantic and syntactic perturbations differ in effectiveness across prompt components?

To address these questions, we introduce PROMPTANATOMY, **the first framework for structurally decomposing prompts into canonical components**. Our framework achieves superior accuracy compared to GPT-4o (Hurst et al. 2024) in dissecting long, complex prompt structures. This dissection enables fine-grained analysis of how each component contributes to model robustness. We first complete missing components and then apply PROMPTANATOMY to four complex, domain-specific prompt datasets, resulting in structurally annotated variants (*PubMedQA-PA, EMEA-PA, Leetcode-PA, CodeGeneration-PA*). Each dataset undergoes manual review by human experts, achieving 100% inter-annotator agreement to ensure annotation quality.

Building on this foundation, we propose COMPERTURB, **a component-wise adversarial perturbation method** that targets each prompt component with customized perturbation strategies. This enables controlled robustness evaluation by isolating the effects of specific prompt modifications. We conduct extensive experiments across multiple LLMs and tasks, revealing that certain components are significantly more vulnerable to perturbation than others in specific circumstances. Our findings provide actionable insights for prompt engineers and model developers to craft safer, more reliable prompts in high-stakes applications.

Contributions. This paper makes the following key contributions: (i) We introduce PROMPTANATOMY, the first framework for structurally decomposing prompts for LLMs into multiple canonical components, enabling fine-grained analysis and systematic prompt design. (ii) We apply PROMPTANATOMY to four complex domain-specific prompt datasets, producing variants annotated with structural components (denoted with the suffix '-PA'). Each dataset is manually reviewed to ensure annotation quality, achieving 100% agreement among human annotators. (iii) We introduce COMPERTURB, a component-wise perturbation method that applies targeted adversarial strategies to individual prompt components, enabling fine-grained analysis of their impact on model behavior and adversarial robustness. (iv) We conduct comprehensive experiments across diverse LLMs and datasets based on COMPERTURB, demonstrating which components most significantly influence robustness. Our findings offer actionable insights for prompt engineers and model developers to design safer and more reliable prompting strategies.

Findings. We summarize three key findings from our comprehensive evaluation: (i) **Component and Structure-Guided Perturbations Amplify Adversarial Effectiveness in Complex Prompts.** COMPERTURB performs best on structurally rich prompts, showing that component-aware attacks are especially powerful in complex settings. (*RQ1*) (ii) **Prompt Components Exhibit Heterogeneous Robustness to Adversarial Perturbations.** Components like *DIR* and *ADI* are more susceptible to perturbations, while *ROL* and *OFT* remain relatively robust. (*RQ2*) (iii) **Semantic Perturbations Are More Effective Than Syntactic Ones Across Prompt Components.** Meaning-level changes outperform syntactic ones across components, highlighting the role of perturbation type. (*RQ3*) Base on these findings, We provide guidelines and takeaways for four potential target audience: offensive/defensive security researchers, prompt engineers, developers, and general users.

2 Related Works

Prompt Dissection Prompt dissection breaks prompts into functional components—such as directives, roles, and formatting—that shape LLM behavior (Schulhoff et al. 2024). This modular approach improves control, interpretability, and task-solving by leveraging reusable substructures (Khot et al. 2022), and reveals latent multilingual capabilities in LLMs (Nie et al. 2024). In adversarial contexts, it enhances attack effectiveness through targeted reconstruction (Li et al. 2024c), underscoring the value of component-aware prompt analysis.

Adversarial Robustness Prompt robustness addresses the resilience of LLMs to adversarial manipulations and variations in prompt wording and structure (Wang et al. 2023a). Due to the sensitivity of LLM outputs to minor perturbations in prompts, ensuring robustness has become a critical area of research, particularly in safety-sensitive domains such as healthcare, finance, and science. Additionally, recent advancements extend robustness considerations to LLM-based agents, reflecting the increasing complexity and interactive contexts of deployed systems (Xiong et al. 2025; Hu et al. 2025). everal works propose black-box adversarial attacks that exploit prompt-level vulnerabilities (Zhu et al. 2023b; Zou et al. 2023; Xue et al. 2023; Das, Raff, and Gaur 2024), while others investigate certified robustness or ensemble-based defenses (Aguilera-Martínez and Berzal 2025). These studies underscore the importance of evaluating LLMs under structurally-aware perturbation paradigms beyond token-level noise.

LLM Safety Benchmark A range of safety benchmarks has been developed to evaluate LLM alignment across adversarial, ethical, and domain-specific risks (Lu et al. 2025). General benchmarks like AdvBench (Wang et al. 2021), SafetyBench (Zhang et al. 2023), and SALAD-Bench (Li et al. 2024a) assess refusal behaviors and robustness to jailbreaks. Domain-specific benchmarks—ChemSafetyBench (Zhao et al. 2024), MedSafetyBench (Han et al. 2024), and SciSafeEval (Li et al. 2024b)—test models on scientific misuse, ethical compliance, and jailbreak resistance. Agent-based evaluations like Agent-SafetyBench (Zhang et al. 2024) and SafeAgentBench (Yin et al. 2024) measure risks in dynamic tool-use environments. While these benchmarks standardize vulnerability assessment, they generally overlook the compositional structure of prompts.

3 Methodology

3.1 PROMPTANATOMY: Identifying and Dissecting LLM Prompt Components

Formulation Let $\mathcal{P}$ denote the space of free-form natural language prompts. Each prompt $p \in \mathcal{P}$ is typically an unstructured token sequence designed to elicit LLM behavior. Existing methods treat p as a monolithic input, lacking a formal structure for semantic analysis. We define *prompt dissection* as mapping p into a structured sequence of functional components:

$$\mathcal{A} : \mathcal{P} \rightarrow \mathcal{C}^n,$$

where $\mathcal{C}$ is the set of canonical components and n is the number of identified components. The output $\mathcal{A}(p) = (c_1, c_2, \ldots, c_n)$ represents the decomposed prompt under our framework, PROMPTANATOMY, which enables component-aware analysis and robustness evaluation.

Challenges *Prompt dissection* introduces several practical and technical challenges. First, existing human-in-the-loop methods (Shivagunde et al. 2024) rely heavily on manual annotation, which is time-consuming and infeasible at scale. Second, LLMs often exhibit limited robustness when parsing long or complex prompts. For instance, they may omit content, oversimplify components, or focus disproportionately on the beginning and end of the prompt, neglecting critical middle sections (see examples in the Appendix 8.1). Third, current LLMs struggle with disentangling overlapping or implicitly expressed components, leading to incorrect or incomplete dissection. This issue is amplified when the interpretation of a sentence depends on adjacent content (e.g., "Use the following format"), highlighting the need for context-aware classification. Finally, the absence of publicly available benchmark datasets tailored for prompt dissection impedes both method development and standardized evaluation.

Limited Context-Aware Prompt Dissection To address these challenges, we develop a two-stage dissection pipeline (Algorithm 1) that combines rule-based segmentation with LLM-based classification under contextual guidance. First, each raw prompt is segmented into individual sentences using punctuation delimiters (e.g., periods, question marks) to enable fine-grained analysis. For each sentence, we construct a local context window by collecting up to two preceding and two following sentences, which helps disambiguate meaning in context-dependent cases. Each target sentence and its context are then passed to a language model (e.g., GPT-4o) using a structured prompt that instructs the model to classify the sentence into one or more of five canonical components: <Role>, <Directive>, <Additional Information>, <Output Formatting>, and <Examples>. The model's responses are aggregated and post-processed to ensure consistency, with an additional verification step to identify and reclassify any missing sentences. The final output is a structured representation of the prompt, where each sentence is wrapped in XML-style tags corresponding to its identified component(s).

Algorithm 1: Prompt Dissection Pipeline

Require: Prompt p from dataset $\mathcal{D}$
Ensure: Component tags $\mathcal{A}(p)$
$S \leftarrow \text{Split}(p, [\texttt{"."}, \texttt{"?"}])$
$\mathcal{A}(p) \leftarrow \emptyset$
for $i = 1$ to $|S|$ **do**
  $C_i \leftarrow \text{ContextWindow}(S, i, k = 2)$
  $r_i \leftarrow \text{LLMClassify}(s_i, C_i)$
  $\mathcal{A}(p) \leftarrow \mathcal{A}(p) \cup \{r_i\}$
end for
$S_{\text{miss}} \leftarrow \text{FindUnlabeled}(S, \mathcal{A}(p))$
for $s_j \in S_{\text{miss}}$ **do**
  $C_j \leftarrow \text{ContextWindow}(S, j, k = 2)$
  $r_j \leftarrow \text{LLMClassify}(s_j, C_j)$
  $\mathcal{A}(p) \leftarrow \mathcal{A}(p) \cup \{r_j\}$
end for
return $\mathcal{A}(p)$

As shown in Table 7 in the Appendix 8.5, PROMPTANATOMY shows strong reliability, achieving 100% accuracy on GPT-4o and over 90% on other open-weight models across diverse domains. We applied it to a diverse set of existing prompt-based datasets across multiple domains, including biomedical question answering (*PubMedQA* (Wang et al. 2022)), translation (*EMEA* (Wang et al. 2022)), calculating password strength (*Leetcode* (Wang et al. 2022)), and code generation (*CodeGeneration* (nceyda 2025)). For each dataset, we automatically dissect the raw prompts into structured components, which result in enhanced datasets *PubMedQA-PA*, *EMEA-PA*, *Leetcode-PA*, *and CodeGeneration-PA* (see Appendix 8.2 for details).

3.2 COMPERTURB: Component-wise Perturbation Method on LLM Prompt

Grounded in our PROMPTANATOMY dissection framework, we propose a novel **com**ponent-wise **perturb**ation method COMPERTURB. Formally, given a prompt p dissected into a set of k labeled components $\{c_1, c_2, \ldots, c_k\}$, COMPPERTURB defines an adversarial transformation function $\mathcal{T}_i$ for each component c_i, resulting in a perturbed prompt $p^{(i)} = \{c_1, \ldots, \mathcal{T}_i(c_i), \ldots, c_k\}$. Each $\mathcal{T}_i$ is instantiated with a perturbation strategy tailored to the semantics of component c_i. We then evaluate the robustness of an LLM f under each component-wise perturbation by measuring the change in output $\Delta f(p, p^{(i)})$ using task-specific metrics.

To instantiate $\mathcal{T}_i$ for each component c_i, we design a suite of multi-level perturbation operations that introduce varying degrees of semantic distortion: (i) special character insertion, which injects noise symbols (e.g., '@', '#', '!') into words to simulate typos or obfuscation; (ii) synonym replacement, which substitutes words with semantically similar alternatives using `nltk`; (iii) word deletion, which removes selected tokens to mimic underspecified or incomplete prompts; (iv) sentence rewriting, which replaces a full sentence with a paraphrased variant to simulate natural reformulations; and (v) component deletion, which removes an entire functional segment of the prompt (e.g.,

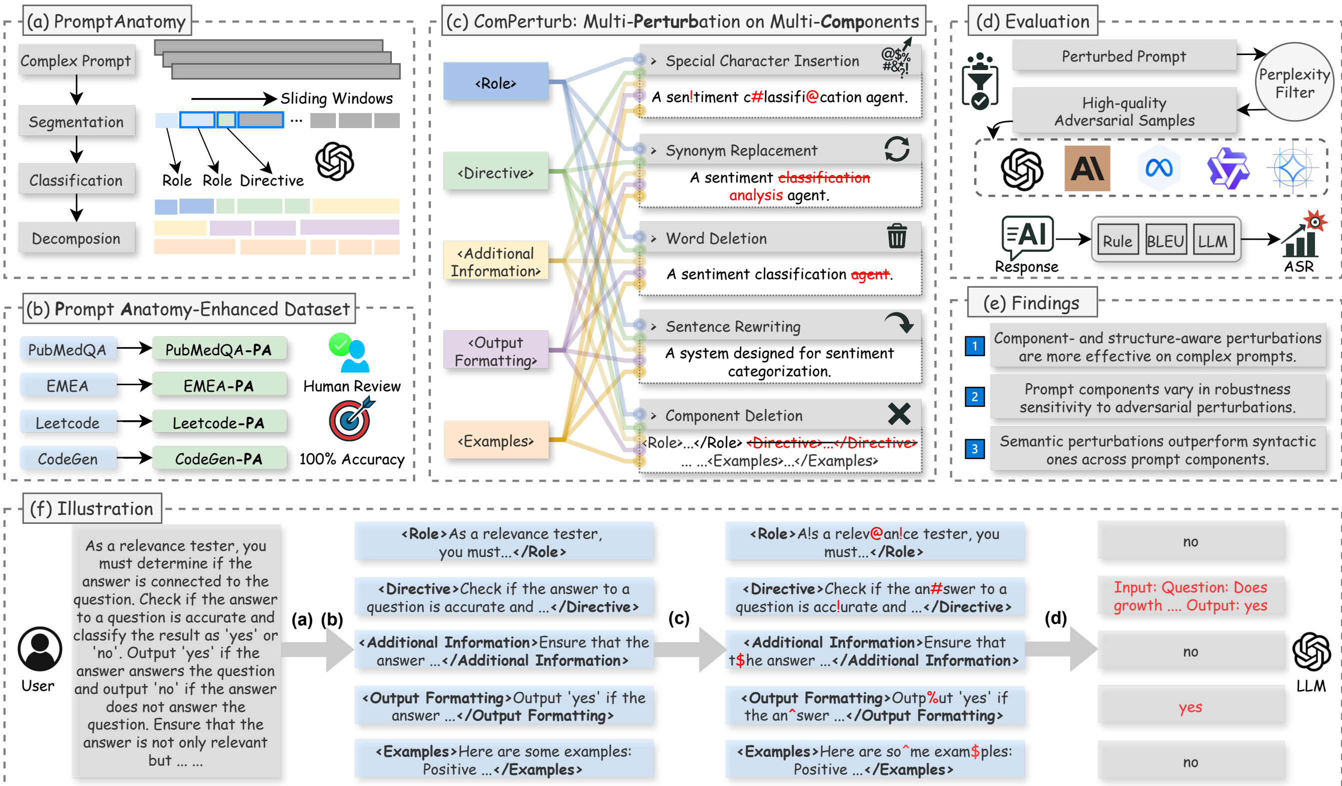

Figure 2: Overview of PromptAnatomy and ComPerturb. (a) Origin prompts are dissected into components using a sliding window LLM classifier. (b) Dissection is applied to benchmarks to create annotated datasets. (c) ComPerturb applies perturbations to components. (d) Perturbed prompts are enhanced and use to evaluate LLMs. (e) Analysis reveals component-wise vulnerabilities. (f) A working example shows dissection, perturbation, and output shifts.

`<Directive>` or `<Examples>`) based on our structural decomposition.

To ensure the effectiveness and representativeness of adversarial examples, we apply a perplexity-guided filtering strategy. Specifically, for each perturbed prompt $p^{(i)}$, we compute its perplexity $\mathrm{PPL}(p^{(i)})$ using a reference language model. To quantify the relative semantic disturbance, we calculate the perplexity ratio between the perturbed prompt and its corresponding clean version $p^{(0)}$ as:

$$\mathrm{Ratio}^{(i)} = \frac{\mathrm{PPL}\left(p^{(i)}\right)}{\mathrm{PPL}\left(p^{(0)}\right)} \tag{1}$$

where $\mathrm{Ratio}^{(i)}$ denotes the perplexity ratio of the i-th perturbed prompt, $p^{(i)}$ is the perturbed version of the prompt, and $p^{(0)}$ is the original unperturbed prompt.

We then rank all perturbed samples in descending order of $\mathrm{Ratio}^{(i)}$ and retain the top 20% as our final high-quality adversarial set, effectively filtering out low-impact or semantically trivial perturbations.

4 Experiment

Setup. We evaluate the effectiveness of ComPerturb across a diverse suite of language models and task domains. For target models, we select five widely-used LLMs with varying capacities and architectures: *GPT-4o* (M1), *Claude3.7* (M2), *Qwen2.5-14B* (M3), *LLaMA3.3-70B* (M4), and *Gemma3-12B* (M5). For adversarial prompt generation, we employ both commercial and open-weight models, including *GPT-4o*, *GPT-3.5*, *Qwen3-8B*, *Qwen2.5-14B*, *LLaMA3.3-70B*, and *Gemma3-12B*. Our evaluation uses datasets from distinct domains: *PubMedQA-PA* (PM-PA), *EMEA-PA* (EM-PA), *Leetcode-PA* (LC-PA), and *CodeGeneration-PA* (CG-PA). Inference was performed using 4 NVIDIA RTX™ A6000 GPUs for open-weight models, and through official APIs for commercial models. We compare our method against baselines *MTTM* (Wang et al. 2023b) and *PromptRobust* (Zhu et al. 2023a).

Metrics. We adopt task-specific metrics. For classification-style tasks such as PubMedQA and Leetcode, we define an attack as successful if the LLM's output no longer contains the correct answer. For generation tasks such as translation, we calculate Bilingual Evaluation Understudy (BLEU) scores (Papineni et al. 2002) to measure the semantic preservation or degradation after perturbation, an attack is considered a success if BLEU is less than 20, which indicate poor translation quality (Koehn and Knowles 2017). For code generation tasks (evaluated on the Code dataset), we employ GPT-4o as an automatic reference

Method	GPT-4o	Claude3.7	Qwen2.5-14B	LLaMA3.3-70B	Gemma3-12B	Average
			PubMedQA-PA (PCM = 34.883; AVG ↑=17.76%)			
MTTM	42.50 (40.70±0.02)	46.88 (44.23±0.02)	44.67 (39.33±0.03)	45.74 (41.74±0.02)	47.32 (39.08±0.01)	45.42 (41.02±0.02)
PromptRobust	44.37 (41.56±0.03)	47.34 (45.34±0.07)	46.75 (41.71±0.02)	49.03 (47.03±0.05)	49.75 (46.25±0.03)	47.45 (44.38±0.04)
ComPerturb (Ours)	64.98 (53.63±0.07)	67.01 (60.02±0.04)	60.94 (50.76±0.06)	65.04 (52.38±0.07)	63.00 (51.73±0.07)	64.19 (53.70±0.06)
			EMEA-PA (PCM = 32.100; AVG ↑=10.24%)			
MTTM	62.30 (60.01±0.02)	62.24 (60.26±0.01)	74.20 (69.97±0.02)	73.52 (72.00±0.01)	76.30 (72.89±0.02)	69.72 (67.03±0.02)
PromptRobust	69.30 (58.74±0.04)	67.50 (61.56±0.03)	72.30 (70.34±0.05)	75.70 (72.87±0.06)	74.60 (73.81±0.03)	71.88 (67.46±0.04)
ComPerturb (Ours)	75.90 (67.10±0.05)	78.00 (69.11±0.05)	84.90 (78.09±0.04)	82.12 (75.93±0.04)	84.30 (78.66±0.03)	81.04 (73.78±0.04)
			Leetcode-PA (PCM = 41.495; AVG ↑=20.16%)			
MTTM	57.20 (53.45±0.02)	52.30 (49.10±0.01)	47.90 (47.96±0.03)	54.70 (51.89±0.03)	54.40 (50.88±0.02)	53.30 (50.66±0.02)
PromptRobust	56.30 (54.98±0.05)	55.20 (54.32±0.02)	56.70 (55.12±0.05)	51.20 (50.32±0.03)	56.90 (54.30±0.02)	55.26 (53.81±0.03)
ComPerturb (Ours)	81.50 (75.40±0.04)	72.08 (65.74±0.04)	65.30 (62.57±0.02)	78.80 (76.10±0.03)	74.50 (67.42±0.04)	74.44 (69.45±0.03)
			CodeGeneration-PA (PCM = 44.524; AVG ↑=29.6%)			
MTTM	31.20 (27.98±0.02)	29.30 (27.77±0.01)	40.20 (38.40±0.03)	40.30 (38.17±0.03)	41.70 (38.78±0.02)	36.54 (34.22±0.02)
PromptRobust	29.08 (23.46±0.05)	29.30 (27.32±0.02)	40.20 (35.78±0.05)	41.20 (36.21±0.03)	38.40 (33.21±0.02)	35.78 (31.20±0.03)
ComPerturb (Ours)	66.96 (61.48±0.04)	64.00 (57.09±0.04)	66.36 (59.35±0.05)	65.40 (60.72±0.03)	66.08 (60.19±0.03)	65.76 (59.77±0.04)

Table 1: Comparison of ASR (%) of COMPERTURB and baselines across five LLMs and four datasets. Each cell reports the best ASR, followed by the mean ASR and standard deviation from different perturbation strategies of each method.

model to assess the correctness of the generated code.

Based on various successful metrics for different tasks, we compute Attack Success Rate (ASR) as:

$$\text{ASR} = \frac{N_{\text{success}}}{N_{\text{total}}} \times 100\% \tag{2}$$

where $N_{success}$ is the number of samples attacked successfully, N_{total} is the total number of samples.

We introduce the **Prompt Complexity Metric (PCM)**, a quantitative measure designed to evaluate the inherent complexity of LLM prompts. PCM integrates five core dimensions: lexical rarity, syntactic depth, semantic dispersion, structural richness, and task difficulty. Each component captures a distinct aspect of prompt formulation. The PCM score is computed as a weighted sum of these dimensions:

$$\begin{aligned} \text{PCM}(p) = \alpha C_{\text{lexical}} + \beta C_{\text{syntactic}} + \gamma C_{\text{semantic}} \\ + \delta C_{\text{structural}} + \epsilon C_{\text{task}}, \end{aligned} \tag{3}$$

where C_{lexical} denotes the average inverse document frequency of tokens, $C_{\text{syntactic}}$ reflects syntactic tree depth, C_{semantic} measures pairwise embedding-based semantic distance, $C_{\text{structural}}$ captures the organization of prompt components, and C_{task} encodes the inherent difficulty of the task. Detail could be found at Appendix 8.4.

5 Results and Analysis

Table 1 provides a comparison of proposed COMPERTURB with baselines. For each method, we report both the highest ASR and the mean ASR with standard deviation, aggregated over multiple perturbation strategies. The results demonstrate that COMPERTURB achieves the highest ASRs across all settings. It achieves significant gains in complex scenarios by leveraging a component-aware design that targets semantically critical prompt segments, highlighting the advantage of structurally and semantically informed perturbations.

Finding #1: Component and Structure-Guided Perturbations Result in a Better Effect in Complex Prompts.

Based on this finding, we derive a practical guideline to improve LLM robustness for developers.

Guideline #1: Enhancing robustness requires clear task specification by users and component-aware adversarial data augmentation by developers.

Table 2 reports the ASRs of five LLMs across four domain-specific datasets, categorized by prompt components and perturbation types, revealing critical patterns in adversarial robustness. The DIR and ADI consistently yield higher ASRs, highlighting their centrality in conveying task-critical semantics. In contrast, the ROL shows greater resilience, likely due to its auxiliary function. Semantic perturbations, including SYR and SER, are generally more effective than syntactic ones like SCI, underscoring how meaning-level disruptions more severely compromise model behavior. Furthermore, model-specific vulnerabilities emerge, with *LLaMA3-70B* and *Gemma3-12B* being particularly susceptible to component-wise attacks, possibly due to architectural or training differences. Heatmaps in Figures 4 to 8 (See in the Appendix 8.6) visualize these patterns, showing that datasets such as *EMEA-PA* and *Leetcode-PA* exhibit higher ASRs. These prompts depend heavily on semantically dense instructions and multi-step reasoning—especially within DIR and ADI components—making them more fragile under perturbation. By contrast, prompts from *PubMedQA-PA* and *CodeGeneration-PA*, which are more syntax-oriented or retrieval-based, display greater robustness, as superficial semantic alterations have less impact

Model	COM	PPL	PubMedQA-PA					EMEA-PA					Leetcode-PA					CodeGeneration-PA				
			SCI	SYR	WOD	SER	COD	SCI	SYR	WOD	SER	COD	SCI	SYR	WOD	SER	COD	SCI	SYR	WOD	SER	COD
GPT-4o	ROL	w/o	46.0	40.0	42.5	46.0	42.5	58.5	58.5	57.5	57.5	61.0	57.4	60.0	60.5	65.4	65.0	43.5	43.5	42.5	40.5	41.5
		w/	47.0	45.5	52.5	50.0	47.4	65.0	59.0	60.0	57.5	67.5	70.5	70.5	75.0	69.0	75.0	60.0	52.5	57.4	60.0	62.5
	DIR	w/o	56.0	52.0	59.5	50.0	77.5	62.5	62.0	72.0	64.5	99.5	72.5	70.0	72.0	72.5	95.0	50.5	47.0	49.5	47.5	64.0
		w/	60.0	57.4	64.0	65.0	78.5	67.5	67.5	73.0	72.5	99.5	75.0	77.5	80.0	77.5	97.5	67.0	62.5	67.0	65.0	72.5
	ADI	w/o	46.0	43.0	47.5	45.0	52.5	59.0	61.0	59.5	62.5	59.5	62.5	67.5	71.0	70.0	77.5	44.0	43.5	45.0	45.0	45.5
		w/	55.0	45.0	57.4	43.5	75.0	62.5	63.5	68.5	67.5	68.0	70.5	68.5	80.0	72.5	87.0	67.5	60.0	62.5	65.0	62.5
	OFT	w/o	45.0	46.0	45.5	48.0	47.0	59.0	61.0	58.5	58.0	58.0	57.4	66.0	65.5	66.0	66.5	41.5	45.5	43.0	44.0	43.5
		w/	47.5	50.0	46.0	50.0	52.5	62.5	75.0	65.0	62.0	65.0	71.0	75.0	77.5	69.0	85.0	65.0	57.5	57.4	62.5	55.0
	EXA	w/o	45.5	43.5	42.0	47.5	46.0	57.5	59.5	58.0	57.5	61.5	50.0	55.0	69.0	65.0	69.5	45.5	43.5	44.0	44.0	42.0
		w/	47.5	44.0	55.0	55.0	47.0	62.5	62.5	65.0	62.0	77.5	68.0	64.0	69.0	67.0	97.5	62.5	57.4	62.5	57.4	55.0
	CRT	w/o	44.0	43.5	43.0	48.5	52.5	60.5	59.5	59.5	59.5	69.5	60.0	69.0	70.5	69.5	76.0	45.0	43.5	44.0	43.0	54.0
		w/	55.0	45.0	60.0	50.0	55.0	62.5	62.5	60.0	70.0	70.5	71.0	70.0	72.5	70.0	87.0	62.0	60.0	65.0	55.0	62.0
Claude3.7	ROL	w/o	50.0	48.0	48.0	52.0	52.5	57.5	58.0	50.0	58.5	59.0	50.4	53.0	50.0	50.5	50.5	39.0	40.0	42.0	39.0	39.5
		w/	55.0	52.5	53.5	65.5	57.5	67.5	62.5	59.0	70.0	65.0	57.4	53.5	72.5	55.0	75.0	60.0	57.4	55.0	60.0	50.0
	DIR	w/o	58.5	56.0	57.0	59.5	77.5	67.0	61.0	68.5	64.5	99.0	60.5	59.0	61.0	60.0	90.0	46.0	49.0	49.0	46.0	67.0
		w/	65.0	60.0	60.0	62.5	87.5	77.5	65.0	75.0	72.5	100	65.0	62.5	74.0	67.4	92.0	60.0	62.5	62.5	60.0	72.5
	ADI	w/o	56.5	54.0	52.0	55.0	64.0	61.5	59.0	60.0	59.5	59.5	53.0	60.5	57.4	53.5	76.5	43.0	44.0	42.0	43.0	49.0
		w/	58.0	57.5	55.0	65.0	65.0	67.5	72.0	69.5	70.0	70.0	53.5	60.0	65.0	56.0	96.5	55.0	57.0	55.5	55.0	60.0
	OFT	w/o	52.0	50.5	54.0	58.5	66.5	58.5	57.0	60.0	59.5	58.5	50.0	50.5	51.5	55.0	51.5	42.0	40.0	42.0	42.0	40.0
		w/	55.0	57.5	54.5	60.0	66.0	70.0	67.0	60.0	68.5	60.0	54.5	60.0	67.5	60.0	80.0	57.5	50.0	57.4	57.5	57.5
	EXA	w/o	52.0	52.0	56.0	52.5	57.4	60.0	59.5	59.0	58.0	59.0	53.0	54.5	55.5	56.4	52.5	40.0	45.0	43.0	40.0	39.0
		w/	55.0	60.0	57.5	57.5	58.0	65.0	65.0	67.5	68.5	72.5	57.5	55.7	56.5	57.0	90.0	55.0	57.4	55.0	55.0	47.5
	CRT	w/o	55.5	52.5	52.0	52.5	59.0	59.0	60.0	60.0	59.0	70.5	57.4	57.4	56.9	55.0	60.5	40.0	41.0	42.0	40.0	49.0
		w/	57.0	55.0	52.5	55.0	70.0	67.0	70.0	67.5	67.0	70.5	62.5	61.0	59.0	61.0	87.5	50.0	57.5	55.5	50.0	55.0
Qwen2.5-14B	ROL	w/o	39.2	44.2	42.0	46.0	38.7	71.1	74.5	72.5	75.7	73.2	51.6	50.3	50.2	50.5	51.7	44.0	43.0	42.5	39.5	43.0
		w/	43.7	43.7	45.0	48.5	47.5	74.5	76.5	76.2	77.5	75.0	49.5	53.5	62.5	64.0	70.5	60.5	57.4	57.0	57.4	60.5
	DIR	w/o	41.5	50.0	47.0	47.5	96.5	74.8	77.3	87.0	81.2	99.5	56.8	54.2	55.7	53.5	93.0	59.5	49.5	49.5	49.5	67.0
		w/	50.0	51.2	55.0	51.2	97.5	75.0	80.0	88.7	81.3	99.5	60.5	54.5	64.0	54.5	93.0	70.0	57.4	68.0	59.5	77.4
	ADI	w/o	41.4	47.5	45.2	46.0	40.7	74.7	75.2	75.0	75.5	74.7	51.9	54.2	54.6	54.3	92.1	49.5	45.0	45.0	41.0	45.5
		w/	50.0	52.0	51.5	51.2	47.5	74.8	73.8	80.0	72.5	89.0	52.0	56.4	57.9	55.6	98.0	62.5	50.0	62.8	56.0	62.5
	OFT	w/o	42.0	45.0	43.7	44.5	45.0	74.2	74.7	75.2	74.5	75.0	50.4	50.2	51.6	50.6	52.0	40.5	43.0	44.5	40.8	44.0
		w/	52.5	52.5	50.0	47.5	48.7	72.5	76.3	80.0	75.0	80.0	45.5	51.5	74.5	52.0	85.0	72.5	52.5	57.4	58.0	53.0
	EXA	w/o	41.0	45.7	45.0	45.5	37.2	71.2	76.2	75.0	73.0	74.2	50.7	50.1	52.3	51.5	54.6	43.5	42.0	44.0	40.5	43.0
		w/	38.7	47.5	50.0	48.5	47.5	75.0	80.0	75.2	74.0	70.5	50.9	51.5	62.5	56.0	88.5	57.4	55.0	60.0	50.0	50.0
	CRT	w/o	39.5	47.0	43.0	45.4	47.0	73.8	75.7	75.7	74.7	79.0	51.7	53.7	51.5	52.5	59.6	45.5	42.5	44.5	40.5	49.5
		w/	41.2	43.7	43.5	47.0	50.0	75.0	76.0	75.7	76.3	78.8	56.0	53.7	62.0	54.0	90.0	60.5	62.5	60.0	52.7	60.0
LLaMA3-70B	ROL	w/o	45.0	40.0	43.5	44.0	45.0	70.5	70.0	70.7	72.5	71.2	70.0	60.0	66.0	66.5	78.5	40.0	42.0	45.0	41.0	43.0
		w/	48.0	48.5	46.5	47.5	52.5	70.0	70.0	71.7	73.7	78.7	70.0	69.5	75.0	66.0	80.0	57.4	55.0	58.0	60.0	60.0
	DIR	w/o	49.0	49.5	50.0	49.5	100	76.0	74.0	83.2	75.5	99.7	70.0	73.0	72.5	68.5	93.5	51.5	51.0	50.0	52.5	68.5
		w/	57.5	52.5	58.0	57.5	100	78.7	76.0	83.5	73.7	99.7	75.0	72.0	75.0	78.5	93.5	62.5	62.5	65.0	62.0	75.0
	ADI	w/o	47.0	45.5	46.5	47.0	47.0	72.0	71.0	72.2	71.2	70.0	72.5	71.5	72.0	72.5	80.0	42.2	46.0	43.0	47.0	53.0
		w/	50.0	47.5	55.0	50.0	50.0	75.0	71.3	73.2	75.0	82.5	75.0	77.5	72.5	72.5	90.0	70.0	62.5	62.5	52.5	60.5
	OFT	w/o	48.5	45.0	43.0	46.5	46.5	72.0	71.5	71.2	70.2	71.5	68.5	71.0	71.0	64.5	71.0	42.0	42.0	45.0	41.0	44.0
		w/	48.5	47.0	49.5	50.0	50.0	71.4	77.8	72.5	75.0	75.0	72.5	75.0	87.5	65.0	80.5	65.0	57.4	60.0	62.0	57.4
	EXA	w/o	46.5	44.0	46.5	45.5	38.0	70.5	70.2	72.0	69.2	73.2	71.5	70.0	67.0	66.5	66.5	40.5	43.5	44.5	46.0	40.0
		w/	47.0	45.0	57.4	52.0	42.5	71.5	76.5	70.0	73.7	85.0	72.5	70.0	80.0	65.0	92.5	57.4	62.5	62.5	56.0	52.5
	CRT	w/o	47.5	47.5	43.4	46.0	54.0	73.7	71.0	72.0	70.7	77.8	69.5	66.5	66.5	68.0	84.0	41.0	42.5	45.0	41.5	50.0
		w/	50.0	51.5	55.0	50.0	67.5	77.5	71.0	72.5	71.3	78.8	72.5	70.0	69.0	87.5	90.0	59.0	67.5	65.0	52.5	55.0
Gemma3-12B	ROL	w/o	39.5	39.5	41.5	40.0	41.0	71.0	70.0	72.5	72.0	73.5	60.0	57.5	59.5	59.5	55.0	40.0	44.0	42.5	43.5	43.0
		w/	40.0	45.0	55.0	40.0	57.4	75.0	73.5	75.0	80.5	75.0	60.0	65.0	75.0	60.0	60.0	55.0	55.0	62.5	55.0	60.0
	DIR	w/o	60.5	63.0	48.5	60.0	56.5	74.0	77.5	83.5	79.5	100	70.5	67.0	64.5	67.0	82.2	51.0	51.0	47.0	50.0	67.5
		w/	73.0	60.0	53.0	67.5	62.5	75.0	81.5	85.0	82.5	100	72.5	72.0	75.0	70.0	83.0	65.0	65.0	62.5	60.4	77.5
	ADI	w/o	41.0	40.0	44.0	42.0	44.0	74.5	73.5	73.0	73.0	73.0	59.5	57.4	62.0	71.5	88.5	43.5	44.0	42.5	44.5	44.0
		w/	60.0	47.5	47.5	57.5	51.0	75.0	80.0	85.0	75.0	73.5	59.0	57.5	60.0	75.0	94.0	65.0	52.5	59.5	57.4	67.5
	OFT	w/o	40.5	43.0	38.5	39.5	40.0	72.0	72.0	72.0	73.0	72.0	57.5	56.0	55.5	59.5	54.5	42.0	42.5	43.5	43.5	44.0
		w/	47.5	47.5	45.0	57.4	40.0	77.5	75.0	82.5	73.0	75.0	62.5	52.5	75.0	55.0	77.5	67.5	52.5	62.5	55.0	52.5
	EXA	w/o	40.5	40.5	42.0	41.5	44.5	72.0	73.0	72.0	71.5	71.5	60.0	53.5	56.5	59.5	81.5	42.0	43.5	45.0	43.5	44.5
		w/	40.0	40.0	50.0	55.0	47.5	72.5	80.0	82.5	72.5	85.0	60.0	60.0	52.5	59.5	97.5	55.0	62.5	65.0	60.0	52.5
	CRT	w/o	40.5	40.5	41.5	43.5	45.0	73.5	73.5	76.0	72.0	77.5	59.5	62.5	61.0	68.5	64.0	46.0	44.0	45.0	44.0	49.5
		w/	42.5	42.5	52.5	60.0	47.5	77.0	77.5	80.0	75.0	80.0	67.4	65.0	62.5	75.0	80.0	57.4	62.5	62.5	52.5	60.0

Table 2: ASR (%) on five LLMs across four datasets, grouped by prompt components and perturbation types. *Note:* Prompt Components (COM): ROL = Role, DIR = Directive, ADI = Additional Information, OFT = Output Formatting, EXA = Examples. Perturbation Types: SCI = Special Character Insertion, SYR = Synonym Replacement, WOD = Word Deletion, SER = Sentence Rewriting, COD = Component Deletion. CRT = Control Group, AVG = Average Result.

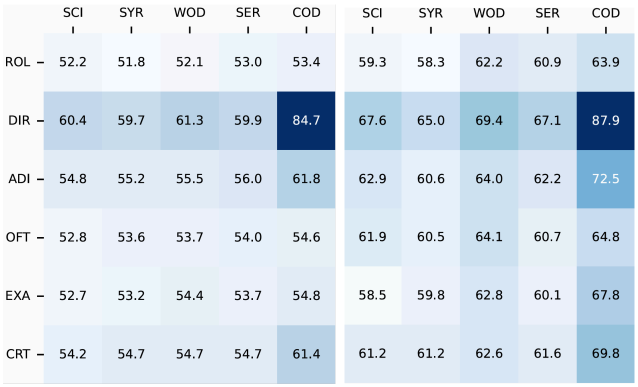

Figure 3: Heatmap of ASR across different prompt components and perturbation types. (left: w/o PPL, right: w/ PPL)

on task execution. These patterns underscore another observation:

Finding #2: Prompt Components Exhibit Heterogeneous Robustness to Adversarial Perturbations.

This leads to the following actionable insight for further improving robustness for prompt engineers and developers.

Guideline #2: Prompt engineers should prioritize protecting high-impact components, while developers should focus defenses on semantically critical elements.

Figure 3, extracted from Table 2, shows that attack success rates (ASRs) vary significantly depending on which prompt component is targeted and by which perturbation type. The DIR component consistently exhibits the highest ASRs—particularly under COD—reaching up to 87.9%, underscoring its critical role in guiding model behavior. This heightened vulnerability stems from the semantic function of DIR, which conveys task-defining instructions; altering or removing it (e.g., via COD or SYR) disrupts task understanding. Similarly, ADI, which provides essential context or constraints, shows moderate susceptibility, especially to COD. In contrast, ROL and OFT serve more auxiliary purposes, such as setting tone or format, and thus display lower ASRs across perturbations. The EXA component yields moderate effects, suggesting limited impact in zero-shot settings. The Control group (CRT), where no semantically meaningful component is perturbed, yields the lowest ASRs, reinforcing that perturbation effectiveness is tightly coupled with a component's functional importance. These results highlight the component-sensitive nature of LLMs, where disrupting core semantic components leads to disproportionately degraded performance, and also suggest that the type of perturbation—not just the target component—plays a key role in adversarial effectiveness:

Finding #3: Semantic Perturbations Are More Effective Than Syntactic Ones Across Prompt Components.

PA	CP	Attack Success Rate (%)					AVG ↑
		M1	M2	M3	M4	M5	
✗	✗	55.9±0.12	54.1±0.08	54.6±0.15	58.9±0.15	56.4±0.15	–
✓	✗	64.8±0.11	62.8±0.09	64.5±0.13	67.8±0.13	66.0±0.13	+9.2
✗	✓	64.0±0.09	61.8±0.06	60.8±0.13	66.7±0.11	65.0±0.11	+7.7
✓	✓	72.3±0.08	70.3±0.06	69.4±0.11	72.8±0.09	72.0±0.10	+15.4

Table 3: Ablation study showing the effect of PROMPTANATOMY (PA) and COMPERTURB (CP) on ASR.

Building on this finding, we recommend the following best practice for both users and developers.

Guideline #3: Users should avoid vague rephrasing, and developers should train models to withstand meaning-level changes for improved robustness.

We conduct an ablation study to evaluate the impact of PROMPTANATOMY (PA) and COMPERTURB (CP) across five LLMs, testing four configurations with and without each component. As shown in Table 3, using PA or CP individually boosts ASR by +9.2% and +7.7%, respectively, demonstrating their independent effectiveness. Combining both yields the highest gain, with a +15.4% average ASR improvement, confirming their complementary strengths in enhancing attack performance.

6 Discussion and Takeaways

Our findings show that prompt components vary in adversarial robustness, challenging the notion that prompts are structurally uniform. (i) For security researchers, focusing on semantically critical components like Directive and Additional Information enables more effective red and blue teaming. (ii) For prompt engineers and developers, adopting a component-aware design enhances robustness and control in high-stakes scenarios. (iii) For general users, clearly specifying the Directive and relevant context improves model understanding and response quality.

7 Conclusion

We reveal that different prompt components exhibit heterogeneous adversarial vulnerabilities in large language models, challenging the prevalent assumption that prompts are uniformly robust. To facilitate fine-grained robustness analysis, we introduced PROMPTANATOMY, a novel framework that structurally dissects prompts into canonical components, and COMPERTURB, a component-wise perturbation method. Extensive experiments show that vulnerabilities vary by component, with perturbation effectiveness tied to semantics and increasing with prompt complexity. These findings highlight the importance of structure-aware adversarial evaluation and the effectiveness of component-guided design in improving prompt robustness. We advocate for structure-aware prompt engineering and training-time defenses to build safer, more robust language models.

References

Aguilera-Martínez, F.; and Berzal, F. 2025. LLM Security: Vulnerabilities, Attacks, Defenses, and Countermeasures. *arXiv preprint arXiv:2505.01177*.

Das, N.; Raff, E.; and Gaur, M. 2024. Human-Interpretable Adversarial Prompt Attack on Large Language Models with Situational Context. *arXiv preprint arXiv:2407.14644*.

Demszky, D.; Yang, D.; Yeager, D. S.; Bryan, C. J.; Clapper, M.; Chandhok, S.; Eichstaedt, J. C.; Hecht, C.; Jamieson, J.; Johnson, M.; et al. 2023. Using large language models in psychology. *Nature Reviews Psychology*, 2(11): 688–701.

Dong, Q.; Li, L.; Dai, D.; Zheng, C.; Ma, J.; Li, R.; Xia, H.; Xu, J.; Wu, Z.; Liu, T.; et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Ghosh, S.; Frase, H.; Williams, A.; Luger, S.; Röttger, P.; Barez, F.; McGregor, S.; Fricklas, K.; Kumar, M.; Bollacker, K.; et al. 2025. Ailuminate: Introducing v1. 0 of the ai risk and reliability benchmark from mlcommons. *arXiv preprint arXiv:2503.05731*.

Han, T.; Kumar, A.; Agarwal, C.; and Lakkaraju, H. 2024. Medsafetybench: Evaluating and improving the medical safety of large language models. *Advances in Neural Information Processing Systems*, 37: 33423–33454.

Hou, X.; Zhao, Y.; Liu, Y.; Yang, Z.; Wang, K.; Li, L.; Luo, X.; Lo, D.; Grundy, J.; and Wang, H. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8): 1–79.

Hu, J.; Dong, Y.; Ding, Z.; and Huang, X. 2025. Enhancing Robustness of LLM-Driven Multi-Agent Systems through Randomized Smoothing. *arXiv preprint arXiv:2507.04105*.

Hu, Z.; Wang, C.; Shu, Y.; Paik, H.-Y.; and Zhu, L. 2024. Prompt perturbation in retrieval-augmented generation based large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1119–1130.

Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Khot, T.; Trivedi, H.; Finlayson, M.; Fu, Y.; Richardson, K.; Clark, P.; and Sabharwal, A. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Koehn, P.; and Knowles, R. 2017. Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872*.

Li, L.; Dong, B.; Wang, R.; Hu, X.; Zuo, W.; Lin, D.; Qiao, Y.; and Shao, J. 2024a. Salad-bench: A hierarchical and comprehensive safety benchmark for large language models. *arXiv preprint arXiv:2402.05044*.

Li, T.; Lu, J.; Chu, C.; Zeng, T.; Zheng, Y.; Li, M.; Huang, H.; Wu, B.; Liu, Z.; Ma, K.; et al. 2024b. Scisafeeval: a comprehensive benchmark for safety alignment of large language models in scientific tasks. *arXiv preprint arXiv:2410.03769*.

Li, X.; Wang, R.; Cheng, M.; Zhou, T.; and Hsieh, C.-J. 2024c. Drattack: Prompt decomposition and reconstruction makes powerful llm jailbreakers. *arXiv preprint arXiv:2402.16914*.

Li, Y. 2023. A practical survey on zero-shot prompt design for in-context learning. *arXiv preprint arXiv:2309.13205*.

Long, D.; Zhao, Y.; Brown, H.; Xie, Y.; Zhao, J.; Chen, N.; Kawaguchi, K.; Shieh, M.; and He, J. 2024. Prompt optimization via adversarial in-context learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7308–7327.

Lu, H.; Fang, L.; Zhang, R.; Li, X.; Cai, J.; Cheng, H.; Tang, L.; Liu, Z.; Sun, Z.; Wang, T.; et al. 2025. Alignment and Safety in Large Language Models: Safety Mechanisms, Training Paradigms, and Emerging Challenges. *arXiv preprint arXiv:2507.19672*.

Mei, L.; Yao, J.; Ge, Y.; Wang, Y.; Bi, B.; Cai, Y.; Liu, J.; Li, M.; Li, Z.-Z.; Zhang, D.; et al. 2025. A Survey of Context Engineering for Large Language Models. *arXiv preprint arXiv:2507.13334*.

Naveed, H.; Khan, A. U.; Qiu, S.; Saqib, M.; Anwar, S.; Usman, M.; Akhtar, N.; Barnes, N.; and Mian, A. 2023. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*.

nceyda. 2025. YAP470 Code Generation Dataset. https://huggingface.co/datasets/nceyda/YAP470_Code_Generation_Dataset. Accessed: 2025-07-05.

Nie, E.; Yuan, S.; Ma, B.; Schmid, H.; Färber, M.; Kreuter, F.; and Schütze, H. 2024. Decomposed prompting: Unveiling multilingual linguistic structure knowledge in english-centric large language models. *arXiv preprint arXiv:2402.18397*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 311–318.

Schulhoff, S.; Ilie, M.; Balepur, N.; Kahadze, K.; Liu, A.; Si, C.; Li, Y.; Gupta, A.; Han, H.; Schulhoff, S.; et al. 2024. The prompt report: a systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608*.

Shivagunde, N.; Lialin, V.; Muckatira, S.; and Rumshisky, A. 2024. Deconstructing in-context learning: Understanding prompts via corruption. *arXiv preprint arXiv:2404.02054*.

Thirunavukarasu, A. J.; Ting, D. S. J.; Elangovan, K.; Gutierrez, L.; Tan, T. F.; and Ting, D. S. W. 2023. Large language models in medicine. *Nature medicine*, 29(8): 1930–1940.

Wang, B.; Xu, C.; Wang, S.; Gan, Z.; Cheng, Y.; Gao, J.; Awadallah, A. H.; and Li, B. 2021. Adversarial glue: A multi-task benchmark for robustness evaluation of language models. *arXiv preprint arXiv:2111.02840*.

Wang, J.; Hu, X.; Hou, W.; Chen, H.; Zheng, R.; Wang, Y.; Yang, L.; Huang, H.; Ye, W.; Geng, X.; et al. 2023a. On the robustness of chatgpt: An adversarial and out-of-distribution perspective. *arXiv preprint arXiv:2302.12095*.

Wang, W.; Huang, J.-t.; Wu, W.; Zhang, J.; Huang, Y.; Li, S.; He, P.; and Lyu, M. R. 2023b. Mttm: Metamorphic testing for textual content moderation software. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2387–2399. IEEE.

Wang, Y.; Mishra, S.; Alipoormolabashi, P.; Kordi, Y.; Mirzaei, A.; Naik, A.; Ashok, A.; Dhanasekaran, A. S.; Arunkumar, A.; Stap, D.; et al. 2022. Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 5085–5109.

Xiong, Q.; Huang, Y.; Jiang, Z.; Chang, Z.; Zheng, Y.; Li, T.; and Li, M. 2025. Butterfly Effects in Toolchains: A Comprehensive Analysis of Failed Parameter Filling in LLM Tool-Agent Systems. *arXiv preprint arXiv:2507.15296*.

Xue, J.; Zheng, M.; Hua, T.; Shen, Y.; Liu, Y.; Bölöni, L.; and Lou, Q. 2023. Trojllm: A black-box trojan prompt attack on large language models. *Advances in Neural Information Processing Systems*, 36: 65665–65677.

Yin, S.; Pang, X.; Ding, Y.; Chen, M.; Bi, Y.; Xiong, Y.; Huang, W.; Xiang, Z.; Shao, J.; and Chen, S. 2024. Safeagentbench: A benchmark for safe task planning of embodied llm agents. *arXiv preprint arXiv:2412.13178*.

Zhang, Q.; Ding, K.; Lv, T.; Wang, X.; Yin, Q.; Zhang, Y.; Yu, J.; Wang, Y.; Li, X.; Xiang, Z.; et al. 2025. Scientific large language models: A survey on biological & chemical domains. *ACM Computing Surveys*, 57(6): 1–38.

Zhang, Z.; Cui, S.; Lu, Y.; Zhou, J.; Yang, J.; Wang, H.; and Huang, M. 2024. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*.

Zhang, Z.; Lei, L.; Wu, L.; Sun, R.; Huang, Y.; Long, C.; Liu, X.; Lei, X.; Tang, J.; and Huang, M. 2023. Safetybench: Evaluating the safety of large language models with multiple choice questions. *CoRR*.

Zhao, H.; Tang, X.; Yang, Z.; Han, X.; Feng, X.; Fan, Y.; Cheng, S.; Jin, D.; Zhao, Y.; Cohan, A.; et al. 2024. Chemsafetybench: benchmarking llm safety on chemistry domain. *arXiv preprint arXiv:2411.16736*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

Zhu, K.; Wang, J.; Zhou, J.; Wang, Z.; Chen, H.; Wang, Y.; Yang, L.; Ye, W.; Zhang, Y.; Gong, N.; et al. 2023a. Promptrobust: Towards evaluating the robustness of large language models on adversarial prompts. In *Proceedings of the 1st ACM workshop on large AI systems and models with privacy and safety analysis*, 57–68.

Zhu, K.; Wang, J.; Zhou, J.; Wang, Z.; Chen, H.; Wang, Y.; Yang, L.; Ye, W.; Zhang, Y.; Zhenqiang Gong, N.; et al. 2023b. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv e-prints*, arXiv–2306.

Zou, A.; Wang, Z.; Carlini, N.; Nasr, M.; Kolter, J. Z.; and Fredrikson, M. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

8 Appendix

8.1 Examples of LLM Fails to Dissect Prompt

Misdissection of Semantic Components by GPT-4o

Prompt to GPT-4o

You are now an expert in prompt engineering for large language models. Please classify the following prompt into the five components: Role, Directive, Additional Information, Output Formatting and Examples. Please classify the following prompt into its respective components using implicit tags such as <Role></Role>, <Directive></Directive>, <Additional Information></Additional Information>, <Output Formatting></Output Formatting> and <Examples></Examples>. Do not provide any explanations or additional content. Each sentence should remain unmodified. The classified sentences should remain the same as the original sentences. Below are the definitions of each component: Role: Also known as a persona, this component helps improve the writing style and tone of the text. Examples: As a helpful assistant, answer the following question concisely. Directive: The main instruction or question in the prompt. i.e. What are the benefits of renewable energy? Or Tell me five books. Additional Information: Provides additional background information or situational details to help the model better understand the task. Example: You are writing an email to a colleague about a meeting next week. Output Formatting: Defines how the response should be structured or presented. Example: Answer the question in 'yes' or 'no'. Examples: Provides sample inputs and outputs to guide the model's response. Example: Input: 'Night' Output: 'Noche' (Spanish). The prompt to classify is:
A seasoned Python developer like you has extensive experience in writing and refining Python code, and your expertise includes key domains such as software engineering, data processing, and algorithm optimization. Your task is to either create new Python scripts that fulfill specific functional requirements or modify existing Python code to enhance its performance, readability, or functionality. These tasks involve implementing basic functions, optimizing existing code, and ensuring that the code adheres to best practices in software development. The output should be a well-structured Python script. It should include clear comments explaining the purpose of each section of the code. The script should be formatted according to PEP 8 guidelines, ensuring readability and consistency. Below is an instruction that describes a task, Write a response that appropriately completes the request. Inspect the following Python file from murano/tests/unit/api/middleware/test_version_negotiation.py. Update the given code considering this change: ”Change assertTrue(isinstance()) by optimal assert” “‘python # Copyright 2016 AT&T Corp # All Rights Reserved.# # Licensed under the Apache License, Version 2.0 (the ”License”); you may # not use this file except in compliance with the License.You may obtain # a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an ”AS IS” BASIS, WITHOUT # WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the # License for the specific language governing permissions and limitations # under the License.import webob from murano.api import versions from murano.api.middleware import version_negotiation from murano.tests.unit import base class MiddlewareVersionNegotiationTest(base.MuranoTestCase): def test_middleware_version_negotiation_default(self): middleware_vn = version_negotiation.VersionNegotiationFilter(None) request = webob.Request.blank('/environments') result = middleware_vn.process_request(request) self.assertTrue(isinstance(result, versions.Controller)) “‘. Example : Input: Generate a program in Python that takes in a string and returns the reverse of the string.”Hello World” Output: def reverse_string(str): ””” Reverses a given string ””” rev_str = ”” for c in str: rev_str = c + rev_str return rev_str str = ”Hello World” rev_str = reverse_string(str) print(rev_str).

Ground Truth

<Role> A seasoned Python developer like you has extensive experience in writing and refining Python code, and your expertise includes key domains such as software engineering, data processing, and algorithm optimization.</Role>
<Directive> Your task is to either create new Python scripts that fulfill specific functional requirements or modify existing Python code to enhance its performance, readability, or functionality. Below is an instruction that describes a task, Write a response that appropriately completes the request. Inspect the following Python file from murano/tests/unit/api/middleware/test_version_negotiation.py. Update the given code considering this change: ”Change assertTrue(isinstance()) by optimal assert” “‘python # Copyright 2016 AT&T Corp # All Rights Reserved.# # Licensed under the Apache License, Version 2.0 (the ”License”); you may # not use this file except in compliance with the License. You may obtain # a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an ”AS IS” BASIS, WITHOUT # WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the # License for the specific language governing permissions and limitations # under the License. import webob from murano.api import versions from murano.api.middleware import version_negotiation from murano.tests.unit import base class MiddlewareVersionNegotiationTest(base.MuranoTestCase): def test_middleware_version_negotiation_default(self): middleware_vn

= version_negotiation.VersionNegotiationFilter(None) request = webob.Request.blank('/environments') result = middleware_vn.process_request(request) self.assertTrue(isinstance(result, versions.Controller)) "'. </Directive>
<Additional Information> These tasks involve implementing basic functions, optimizing existing code, and ensuring that the code adheres to best practices in software development.</Additional Information>
<Output Formatting> The output should be a well-structured Python script. It should include clear comments explaining the purpose of each section of the code. The script should be formatted according to PEP 8 guidelines, ensuring readability and consistency.</Output Formatting>
<Examples>Example : Input: Generate a program in Python that takes in a string and returns the reverse of the string."Hello World" Output: def reverse_string(str): """" Reverses a given string """" rev_str = "" for c in str: rev_str = c + rev_str return rev_str str = "Hello World" rev_str = reverse_string(str) print(rev_str).</Examples>

Response From GPT-4o
<Role> A seasoned Python developer like you has extensive experience in writing and refining Python code, and your expertise includes key domains such as software engineering, data processing, and algorithm optimization. </Role>
<Directive>Your task is to either create new Python scripts that fulfill specific functional requirements or modify existing Python code to enhance its performance, readability, or functionality. </Directive>
<Additional Information>These tasks involve implementing basic functions, optimizing existing code, and ensuring that the code adheres to best practices in software development. </Additional Information>
<Output Formatting>The output should be a well-structured Python script. It should include clear comments explaining the purpose of each section of the code. The script should be formatted according to PEP 8 guidelines, ensuring readability and consistency. </Output Formatting>
<Additional Information> Below is an instruction that describes a task, Write a response that appropriately completes the request. </Additional Information>
<Examples> Inspect the following Python file from murano/tests/unit/api/middleware/test_version_negotiation.py. Update the given code considering this change: "Change assertTrue(isinstance()) by optimal assert" "'python # Copyright 2016 AT&T Corp # All Rights Reserved.# # Licensed under the Apache License, Version 2.0 (the "License"); you may # not use this file except in compliance with the License.You may obtain # a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an "AS IS" BASIS, WITHOUT # WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.See the # License for the specific language governing permissions and limitations # under the License.import webob from murano.api import versions from murano.api.middleware import version_negotiation from murano.tests.unit import base class MiddlewareVersionNegotiationTest(base.MuranoTestCase): def test_middleware_version_negotiation_default(self): middleware_vn = version_negotiation.VersionNegotiationFilter(None) request = webob.Request.blank('/environments') result = middleware_vn.process_request(request) self.assertTrue(isinstance(result, versions.Controller)) "'. Example : Input: Generate a program in Python that takes in a string and returns the reverse of the string."Hello World" Output: def reverse_string(str): """" Reverses a given string """" rev_str = "" for c in str: rev_str = c + rev_str return rev_str str = "Hello World" rev_str = reverse_string(str) print(rev_str) </Examples>

Omission of Middle Sentences in Prompt Dissection by GPT-4o

Prompt to GPT-4o
You are now an expert in prompt engineering for large language models. Please classify the following prompt into the five components: Role, Directive, Additional Information, Output Formatting and Examples. Please classify the following prompt into its respective components using implicit tags such as <Role></Role>, <Directive></Directive>, <Additional Information></Additional Information>, <Output Formatting></Output Formatting> and <Examples></Examples>. Do not provide any explanations or additional content. Each sentence should remain unmodified. The classified sentences should remain the same as the original sentences. Below are the definitions of each component: Role: Also known as a persona, this component helps improve the writing style and tone of the text. Examples: As a helpful assistant, answer the following question concisely. Directive: The main instruction or question in the prompt. i.e. What are the benefits of renewable energy? Or Tell me five books. Additional Information: Provides additional background information or situational details to help the model better understand the task. Example: You are writing an email to a colleague about a meeting next week. Output Formatting: Defines how the response should be structured or presented. Example: Answer the question in 'yes' or 'no'. Examples: Provides sample inputs and outputs to guide the model's response. Example: Input: 'Night' Output: 'Noche' (Spanish). The prompt to classify is:
As a cybersecurity expert dedicated to password strength analysis and improvement, you are responsible for assessing

and adjusting passwords to meet stringent security criteria. You are given a password and you need to generate the number of steps required to convert the given password to a strong password. A password is considered strong if (a) it has at least 6 characters and at most 20 characters; (b) it contains at least one lowercase letter and one uppercase letter, and at least one digit; (c) it does not contain three repeating characters in a row. In one step you can: (1) Insert one character to password, (2) delete one character from password, or (3) replace one character of password with another character. Calculate the minimum number of operations needed to transform the provided password into a strong one. Make sure your approach is efficient and explores all potential scenarios for optimal results. Answer only returns the number of steps required. Now complete the question: password = KMxgCvxC4A1SxLS2r9OEQGx3MqRaqXX. Here are some examples: Positive Example 1 - Input: password = a Output: 5.Positive Example 2 - Input: password = aA1 Output: 3.Negative Example 1 - Input: password = 1337C0d3 Output: 1.

Ground Truth

<Role>As a cybersecurity expert dedicated to password strength analysis and improvement, you are responsible for assessing and adjusting passwords to meet stringent security criteria.</Role>
<Directive>You are given a password and you need to generate the number of steps required to convert the given password to a strong password. Calculate the minimum number of operations needed to transform the provided password into a strong one. Now complete the question: password = KMxgCvxC4A1SxLS2r9OEQGx3MqRaqXX.</Directive>
<Additional Information>A password is considered strong if (a) it has at least 6 characters and at most 20 characters; (b) it contains at least one lowercase letter and one uppercase letter, and at least one digit; (c) it does not contain three repeating characters in a row. In one step you can: (1) Insert one character to password, (2) delete one character from password, or (3) replace one character of password with another character. Make sure your approach is efficient and explores all potential scenarios for optimal results.</Additional Information>
<Output Formatting>Answer only returns the number of steps required.</Output Formatting>
<Examples>Here are some examples: Positive Example 1 - Input: password = a Output: 5.Positive Example 2 - Input: password = aA1 Output: 3.Negative Example 1 - Input: password = 1337C0d3 Output: 1.</Examples>

Response From GPT-4o

<Role> As a cybersecurity expert dedicated to password strength analysis and improvement, you are responsible for assessing and adjusting passwords to meet stringent security criteria. </Role>
<Directive> You are given a password and you need to generate the number of steps required to convert the given password to a strong password. </Directive>
<Additional Information> A password is considered strong if (a) it has at least 6 characters and at most 20 characters; (b) it contains at least one lowercase letter and one uppercase letter, and at least one digit; (c) it does not contain three repeating characters in a row. In one step you can: (1) Insert one character to password, (2) delete one character from password, or (3) replace one character of password with another character. Make sure your approach is efficient and explores all potential scenarios for optimal results. </Additional Information>
<Output Formatting> Answer only returns the number of steps required. </Output Formatting>
<Examples> Positive Example 1 - Input: password = a Output: 5. Positive Example 2 - Input: password = aA1 Output: 3. Negative Example 1 - Input: password = 1337C0d3 Output: 1. </Examples>
Missing Sentence:
Calculate the minimum number of operations needed to transform the provided password into a strong one.
Now complete the question: password = KMxgCvxC4A1SxLS2r9OEQGx3MqRaqXX.

8.2 Examples of Origin and Enhanced Datasets

Table 4: Dataset Examples from PubMedQA, EMEA, Leetcode, and CodeGeneration

Dataset	Example
PubMedQA	You are a response inspector, ensuring that the answer is a valid reply to the question. Check if the answer to a question is correct and classify the result as 'yes' or 'no'. Output 'yes' if the answer answers the question and output 'no' if the answer does not answer the question. Focus on determining if the answer is both relevant and complete in addressing the question. Now complete the question: Question: Is chk1 required for the metaphase-anaphase transition via regulating the expression and localization of Cdc20 and Mad2? Answer: These results strongly suggest that Chk1 is required for the metaphase-anaphase transition via regulating the subcellular localization and the expression of Cdc20 and Mad2. Here are some examples: Positive Example 1 - Input: Question: Are group 2 innate lymphoid cells (ILC2s) increased in chronic rhinosinusitis with nasal polyps or eosinophilia? Answer: As ILC2s are elevated in patients with CRSwNP, they may drive nasal polyp formation in CRS. ILC2s are also linked with high tissue and blood eosinophilia and have a potential role in the activation and survival of eosinophils during the Th2 immune response. The association of innate lymphoid cells in CRS provides insights into its pathogenesis. Output: yes Positive Example 2 - Input: Question: Does vagus nerve contribute to the development of steatohepatitis and obesity in phosphatidylethanolamine N-methyltransferase deficient mice? Answer: Neuronal signals via the hepatic vagus nerve contribute to the development of steatohepatitis and protection against obesity in HFD fed Pemt(-/-) mice. Output: yes Negative Example 1 - Input: Question: Is methylation of the FGFR2 gene associated with high birth weight centile in humans? Answer: We identified a novel biologically plausible candidate (FGFR2) for with BWC that merits further study. Output: no Negative Example 2 - Input: Question: Do tumor-infiltrating immune cell profiles and their change after neoadjuvant chemotherapy predict response and prognosis of breast cancer? Answer: Breast cancer immune cell subpopulation profiles, determined by immunohistochemistry-based computerized analysis, identify groups of patients characterized by high response (in the pre-treatment setting) and poor prognosis (in the post-treatment setting). Further understanding of the mechanisms underlying the distribution of immune cells and their changes after chemotherapy may contribute to the development of new immune-targeted therapies for breast cancer. Output: no.
EMEA	As an experienced Latin translation specialist, you have a deep knowledge of historical and religious contexts, which enables you to translate Latin texts into English with precision. Your skill lies in grasping the subtleties of classical languages and their cultural importance. Translate the provided Latin text into English, ensuring that the historical and religious nuances are preserved and accurately conveyed. The content you've provided is extensive, with a wide range of texts and references from various fields like religion, history, philosophy, and literature. The translation should be presented in a clear and readable English format, with any necessary annotations or explanations provided to clarify the context or meaning of specific terms or phrases. Now complete the question: ceterum cum et magni pretii et varii generis a legatis eius tam virorum quam feminarum apta usui munera circa domos ferrentur, nulla cuiquam dono ianua patuit, Tarentinaeque petulantiae animosus magis quam efficax defensor haud scio maiore cum gloria huius urbis moribus ¡an moenibus¿ repulsus sit. Here are some examples: Latin Text: Omnia mutantur, nihil interit. Translation: Everything changes, nothing perishes. Latin Text: Qui cum statuisset, nisi me per vos recuperasset, eamdem subire fortunam atque idem sibi domicilium et vitae et mortis deposcere, tamen numquam nec magnitudinem negotii nec solitudinem suam nec vim inimicorum ac tela pertimuit. Translation: He had made up his mind that, should he fail, through you, to win me back to himself, he would ask permission to meet the same fate and to share the same dwelling with me in life and in death; and yet, in spite of this, no toil however formidable, no loneliness, no threat nor weapons of foes, could daunt him.

Dataset	Example
Leetcode	As a cybersecurity expert dedicated to password strength analysis and improvement, you are responsible for assessing and adjusting passwords to meet stringent security criteria. You are given a password and you need to generate the number of steps required to convert the given password to a strong password. A password is considered strong if (a) it has at least 6 characters and at most 20 characters; (b) it contains at least one lowercase letter and one uppercase letter, and at least one digit; (c) it does not contain three repeating characters in a row. In one step you can: (1) Insert one character to password, (2) delete one character from password, or (3) replace one character of password with another character. Calculate the minimum number of operations needed to transform the provided password into a strong one. Make sure your approach is efficient and explores all potential scenarios for optimal results. Answer only returns the number of steps required. Now complete the question: password = KMxgCvxC4A1SxLS2r9OEQGx3MqRaqXX. Here are some examples: Positive Example 1 - Input: password = a Output: 5. Positive Example 2 - Input: password = aA1 Output: 3. Negative Example 1 - Input: password = 1337C0d3 Output: 1.
CodeGeneration	A seasoned Python developer like you has extensive experience in writing and refining Python code, and your expertise includes key domains such as software engineering, data processing, and algorithm optimization. Your task is to either create new Python scripts that fulfill specific functional requirements or modify existing Python code to enhance its performance, readability, or functionality. These tasks involve implementing basic functions, optimizing existing code, and ensuring that the code adheres to best practices in software development. The output should be a well-structured Python script. It should include clear comments explaining the purpose of each section of the code. The script should be formatted according to PEP 8 guidelines, ensuring readability and consistency. Below is an instruction that describes a task, Write a response that appropriately completes the request. Inspect the following Python file from murano/tests/unit/api/middleware/test_version_negotiation.py. Update the given code considering this change: ”Change assertTrue(isinstance()) by optimal assert” “‘python # Copyright 2016 AT&T Corp # All Rights Reserved.# # Licensed under the Apache License, Version 2.0 (the ”License”); you may # not use this file except in compliance with the License.You may obtain # a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an ”AS IS” BASIS, WITHOUT # WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the # License for the specific language governing permissions and limitations # under the License.import webob from murano.api import versions from murano.api.middleware import version_negotiation from murano.tests.unit import base class MiddlewareVersionNegotiationTest(base.MuranoTestCase): def test_middleware_version_negotiation_default(self): middleware_vn = version_negotiation.VersionNegotiationFilter(None) request = webob.Request.blank(’/environments’) result = middleware_vn.process_request(request) self.assertTrue(isinstance(result, versions.Controller)) “‘. Example : Input: Generate a program in Python that takes in a string and returns the reverse of the string.”Hello World” Output: def reverse_string(str): ””” Reverses a given string ””” rev_str = ”” for c in str: rev_str = c + rev_str return rev_str str = ”Hello World” rev_str = reverse_string(str) print(rev_str).

Table 5: Dataset Examples from PubMedQA-PA, EMEA-PA, Leetcode-PA, and CodeGeneration-PA

Dataset	Example
PubMedQA-PA	<Role>You are a response inspector, ensuring that the answer is a valid reply to the question.</Role> <Directive>Check if the answer to a question is correct and classify the result as 'yes' or 'no'. Now complete the question: Question: Is chk1 required for the metaphase-anaphase transition via regulating the expression and localization of Cdc20 and Mad2? Answer: These results strongly suggest that Chk1 is required for the metaphase-anaphase transition via regulating the subcellular localization and the expression of Cdc20 and Mad2. </Directive> <Additional Information>Focus on determining if the answer is both relevant and complete in addressing the question.</Additional Information> <Output Formatting>Output 'yes' if the answer answers the question and output 'no' if the answer does not answer the question. </Output Formatting> <Examples>Here are some examples: Positive Example 1 - Input: Question: Are group 2 innate lymphoid cells (ILC2s) increased in chronic rhinosinusitis with nasal polyps or eosinophilia? Answer: As ILC2s are elevated in patients with CRSwNP, they may drive nasal polyp formation in CRS. ILC2s are also linked with high tissue and blood eosinophilia and have a potential role in the activation and survival of eosinophils during the Th2 immune response. The association of innate lymphoid cells in CRS provides insights into its pathogenesis. Output: yes Positive Example 2 - Input: Question: Does vagus nerve contribute to the development of steatohepatitis and obesity in phosphatidylethanolamine N-methyltransferase deficient mice? Answer: Neuronal signals via the hepatic vagus nerve contribute to the development of steatohepatitis and protection against obesity in HFD fed Pemt(-/-) mice. Output: yes Negative Example 1 - Input: Question: Is methylation of the FGFR2 gene associated with high birth weight centile in humans? Answer: We identified a novel biologically plausible candidate (FGFR2) for with BWC that merits further study. Output: no Negative Example 2 - Input: Question: Do tumor-infiltrating immune cell profiles and their change after neoadjuvant chemotherapy predict response and prognosis of breast cancer? Answer: Breast cancer immune cell subpopulation profiles, determined by immunohistochemistry-based computerized analysis, identify groups of patients characterized by high response (in the pre-treatment setting) and poor prognosis (in the post-treatment setting). Further understanding of the mechanisms underlying the distribution of immune cells and their changes after chemotherapy may contribute to the development of new immune-targeted therapies for breast cancer. Output: no.</Examples>
EMEA-PA	<Role>As an experienced Latin translation specialist, you have a deep knowledge of historical and religious contexts, which enables you to translate Latin texts into English with precision.</Role> <Directive>Translate the provided Latin text into English, ensuring that the historical and religious nuances are preserved and accurately conveyed. Now complete the question: ceterum cum et magni pretii et varii generis a legatis eius tam virorum quam feminarum apta usui munera circa domos ferrentur, nulla cuiquam dono ianua patuit, Tarentinaeque petulantiae animosus magis quam efficax defensor haud scio maiore cum gloria huius urbis moribus ¡an moenibus¿ repulsus sit.</Directive> <Additional Information>Your skill lies in grasping the subtleties of classical languages and their cultural importance. The content you've provided is extensive, with a wide range of texts and references from various fields like religion, history, philosophy, and literature.</Additional Information> <Output Formatting>The translation should be presented in a clear and readable English format, with any necessary annotations or explanations provided to clarify the context or meaning of specific terms or phrases.</Output Formatting> <Examples>Here are some examples: Latin Text: Omnia mutantur, nihil interit. Translation: Everything changes, nothing perishes. Latin Text: Qui cum statuisset, nisi me per vos recuperasset, eamdem subire fortunam atque idem sibi domicilium et vitae et mortis deposcere, tamen numquam nec magnitudinem negotii nec solitudinem suam nec vim inimicorum ac tela pertimuit. Translation: He had made up his mind that, should he fail, through you, to win me back to himself, he would ask permission to meet the same fate and to share the same dwelling with me in life and in death; and yet, in spite of this, no toil however formidable, no loneliness, no threat nor weapons of foes, could daunt him.</Examples>

Dataset	Example
Leetcode-PA	<Role>As a cybersecurity expert dedicated to password strength analysis and improvement, you are responsible for assessing and adjusting passwords to meet stringent security criteria.</Role> <Directive>You are given a password and you need to generate the number of steps required to convert the given password to a strong password.Now complete the question: password = KMxgCvxC4A1SxLS2r9OEQGx3MqRaqXX.</Directive> <Additional Information>A password is considered strong if (a) it has at least 6 characters and at most 20 characters; (b) it contains at least one lowercase letter and one uppercase letter, and at least one digit; (c) it does not contain three repeating characters in a row. In one step you can: (1) Insert one character to password, (2) delete one character from password, or (3) replace one character of password with another character.Calculate the minimum number of operations needed to transform the provided password into a strong one. Make sure your approach is efficient and explores all potential scenarios for optimal results.</Additional Information> <Output Formatting>Answer only returns the number of steps required.</Output Formatting> <Examples>Here are some examples: Positive Example 1 - Input: password = a Output: 5. Positive Example 2 - Input: password = aA1 Output: 3. Negative Example 1 - Input: password = 1337C0d3 Output: 1.</Examples>
CodeGeneration-PA	<Role>A seasoned Python developer like you has extensive experience in writing and refining Python code, and your expertise includes key domains such as software engineering, data processing, and algorithm optimization.</Role> <Directive>Your task is to either create new Python scripts that fulfill specific functional requirements or modify existing Python code to enhance its performance, readability, or functionality. Below is an instruction that describes a task, Write a response that appropriately completes the request. Inspect the following Python file from murano/tests/unit/api/middleware/test_version_negotiation.py. Update the given code considering this change: ”Change assertTrue(isinstance()) by optimal assert” “‘python # Copyright 2016 AT&T Corp # All Rights Reserved.# # Licensed under the Apache License, Version 2.0 (the ”License”); you may # not use this file except in compliance with the License.You may obtain # a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, software # distributed under the License is distributed on an ”AS IS” BASIS, WITHOUT # WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the # License for the specific language governing permissions and limitations # under the License.import webob from murano.api import versions from murano.api.middleware import version_negotiation from murano.tests.unit import base class MiddlewareVersionNegotiationTest(base.MuranoTestCase): def test_middleware_version_negotiation_default(self): middleware_vn = version_negotiation.VersionNegotiationFilter(None) request = webob.Request.blank(’/environments’) result = middleware_vn.process_request(request) self.assertTrue(isinstance(result, versions.Controller)) “‘.</Directive> <Additional Information>These tasks involve implementing basic functions, optimizing existing code, and ensuring that the code adheres to best practices in software development.</Additional Information> <Output Formatting>The output should be a well-structured Python script. It should include clear comments explaining the purpose of each section of the code. The script should be formatted according to PEP 8 guidelines, ensuring readability and consistency. </Output Formatting> <Examples>Example : Input: Generate a program in Python that takes in a string and returns the reverse of the string.”Hello World” Output: def reverse_string(str): ””” Reverses a given string ””” rev_str = ”” for c in str: rev_str = c + rev_str return rev_str str = ”Hello World” rev_str = reverse_string(str) print(rev_str).</Examples>

8.3 Details of Prompt Perturbation Strategies

To evaluate the robustness of LLMs under structurally and semantically aligned adversarial prompts, we design perturbations at four linguistic granularity levels. Each strategy is applied to individual semantic components in prompts. Table 6 is an example of five perturbation types applied to the `Role` component in prompts.

1) Character-Level Perturbation. Simulates typos or symbol noise. For each semantic component, we randomly insert special characters (e.g., #, $, %, &, @) at 10% of token positions, evenly distributed across the span.
Original: a sentiment classification agent.
Perturbed: a senti!ment c#lass@ification agent.

2) Word-Level Perturbation (Deletion). To mimic incomplete or informal text, 10% of words in each component are randomly deleted.
Original: a sentiment classification agent.
Perturbed: a classification agent.

3) Word-Level Perturbation (Synonym Substitution). We substitute 10% of words with synonyms from WordNet to simulate paraphrasing.
Original: a sentiment classification agent.
Perturbed: a sentiment analysis agent.

4) Sentence-Level Perturbation. One sentence per component is randomly replaced with a semantically equivalent sentence from a paraphrase corpus.
Original: a sentiment analysis agent.
Perturbed: a system designed for sentiment categorization.

5) Component-Level Perturbation. An entire semantic component (e.g., `<Role>`, `<Directive>`, `<Examples>`) is removed to assess its marginal contribution.
Example (original): `<Role>...</Role> <Directive>...</Directive> <Examples>...</Examples>`
Example (perturbed): `<Directive>...</Directive> <Examples>...</Examples>` (Role component removed)

Table 6: Examples of five perturbation types applied to the `Role` component in prompts. Each example shows how a clean prompt is modified using one specific perturbation strategy: Special Character Insertion (SCI), Synonym Replacement (SYR), Word Deletion (WOD), Sentence Rewriting (SER), and Component Deletion (COD).

Scenario	Prompt
Clean	You are Assistant, a sentiment classification agent. You will identify the sentiment given an utterance or text.
Special Character Insertion (SCI)	You are Ass!istant, a sentiment c@lassification agent. You will identify the sentiment given an utterance or text.
Synonym Replacement (SYR)	You are Assistant, a sentiment analysis agent. You will identify the sentiment given an utterance or text.
Word Deletion (WOD)	You are Assistant, a sentiment classification. You will identify the sentiment given an utterance or text.
Sentence Rewriting (SER)	You are Assistant, a system designed for sentiment categorization. You will identify the sentiment given an utterance or text.
Component Deletion (COD)	You will identify the sentiment given an utterance or text.

8.4 Prompt Complexity Metric

This appendix details the Prompt Complexity Metric (PCM), which systematically quantifies the complexity of prompts used with Large Language Models (LLMs). The metric integrates lexical, syntactic, semantic, structural, and task-oriented dimensions into a unified measure.

Formulation The PCM for a prompt p is defined as:

$$\text{PCM}(p) = \alpha C_{\text{lexical}} + \beta C_{\text{syntactic}} + \gamma C_{\text{semantic}} + \delta C_{\text{structural}} + \epsilon C_{\text{task}} \tag{4}$$

where coefficients $\alpha, \beta, \gamma, \delta, \epsilon$ are tunable parameters.

Components

Lexical Complexity (C_{lexical}) Measures vocabulary complexity based on token rarity:

$$C_{\text{lexical}}(p) = \frac{1}{|p|} \sum_{w \in p} \text{IDF}(w) \tag{5}$$

where $\text{IDF}(w)$ is the inverse document frequency of token w.

Syntactic Complexity ($C_{\text{syntactic}}$) Represents sentence structure intricacy:

$$C_{\text{syntactic}}(p) = \frac{1}{|S|} \sum_{s \in S} \text{depth}(\text{ParseTree}(s)) \tag{6}$$

where S is the set of sentences in p.

Semantic Complexity (C_{semantic}) Measures semantic dispersion using embeddings:

$$C_{\text{semantic}}(p) = \frac{2}{|S|(|S|-1)} \sum_{s_i, s_j \in S; i<j} 1 - \cos(E(s_i), E(s_j)) \tag{7}$$

where $E(s)$ denotes the embedding of sentence s.

Structural Complexity ($C_{\text{structural}}$) Quantifies prompt decomposition into structural elements:

$$C_{\text{structural}}(p) = k \cdot \log\left(\frac{|p|}{k} + 1\right) \tag{8}$$

where k is the number of distinct structural components in the prompt.

Task Complexity (C_{task}) Evaluates the intrinsic complexity of the requested task, rated on a scale from 1 to 5:

- 1: Simple factual queries
- 3: Moderate reasoning or summarization tasks
- 5: Complex multi-step reasoning tasks

Interpretation Lower PCM scores correspond to simpler prompts, characterized by common vocabulary and straightforward structures, while higher PCM scores indicate more intricate, challenging prompts requiring deeper comprehension or complex reasoning abilities.

Customization The weighting coefficients $\alpha, \beta, \gamma, \delta, \epsilon$ can be adapted according to specific research contexts or task requirements, enhancing PCM's applicability and flexibility.

8.5 Evaluation of PROMPTANATOMY'S Component Classification Accuracy

In this appendix, we present a comprehensive evaluation to assess the effectiveness and reliability of the proposed PromptAnatomy (PA) framework in accurately identifying and classifying semantic components within free-form prompts. This analysis serves to validate whether the PA framework can correctly dissect prompts into meaningful substructures under our defined taxonomy.

Experimental Setup To rigorously evaluate the classification accuracy of PA, we curated a human-annotated dataset across four task domains—PubMedQA, EMEA, Leetcode, and CodeGeneration—spanning both knowledge-intensive and procedural prompts. Each prompt was manually segmented and labeled on a per-sentence basis by expert annotators according to the canonical component schema:<Role>, <Directive>, <Additional Information>, <Output Formatting>, and <Examples>. The annotated data served as ground truth for benchmarking model predictions.

We employed the PA dissection pipeline (Algorithm 1) using six representative LLMs: (1) ChatGPT-4o, (2) ChatGPT-3.5, (3) Qwen2.5-14B, (4) Qwen3-8B, (5) LLaMA3.3-70B, and (6) Gemma3-12B.

Metric: Classification Accuracy We define the classification accuracy as:

$$\text{Accuracy} = \frac{C_p}{N_p} \tag{9}$$

where C_p denotes the number of prompts in which all sentences are correctly classified (i.e., the predicted label set exactly matches the annotated labels for every sentence in the prompt), and N_p is the total number of prompts in the evaluation set.

A prompt is considered correct only if the model's prediction for each sentence in the prompt exactly matches the human-annotated label set. Partial or sentence-level mismatches result in the entire prompt being marked as incorrect.

Results and Analysis The results, summarized in Table 7 , demonstrate strong classification performance across all models. ChatGPT-4o achieves perfect accuracy (100%) on all four datasets, underscoring its exceptional semantic parsing capabilities. ChatGPT-3.5, Qwen2.5-14B, and LLaMA3.3-70B follow closely with average accuracies of approximately 93%, while Qwen3-8B and Gemma3-12B achieve slightly lower scores of 92% and 88%, respectively.

Table 7: Classification Accuracy (%) across different datasets and models

Model	PubMedQA	EMEA	Leetcode	CodeGeneration	AVG
ChatGPT-4o	100.00	100.00	100.00	100.00	100.00
ChatGPT-3.5	89.00	95.00	96.00	93.00	93.00
Qwen3-8B	91.00	94.00	93.00	89.00	92.00
Qwen2.5-14B	88.00	94.00	95.00	93.00	93.00
LLaMA3.3-70B	91.00	97.00	95.50	90.30	93.45
Gemma3-12B	86.00	89.00	94.50	82.50	88.00

These results confirm that the PA framework generalizes well across model scales and architectures, achieving consistent accuracy above 85% even in smaller models. The use of local contextual windows appears to significantly improve the model's ability to disambiguate component roles, especially for structurally ambiguous prompts.

Notably, all models perform robustly across datasets with differing domain styles, indicating that prompt structure is learnable and recognizable irrespective of topical content.

8.6 Visualization of Table 2

To facilitate a more intuitive understanding of the heterogeneous adversarial robustness across prompt components and perturbation types, we present heatmap visualizations of Table 2 for each evaluated LLMs. Each heatmap illustrates the attack success rate (ASR) under different component-wise perturbations, grouped by model and dataset.

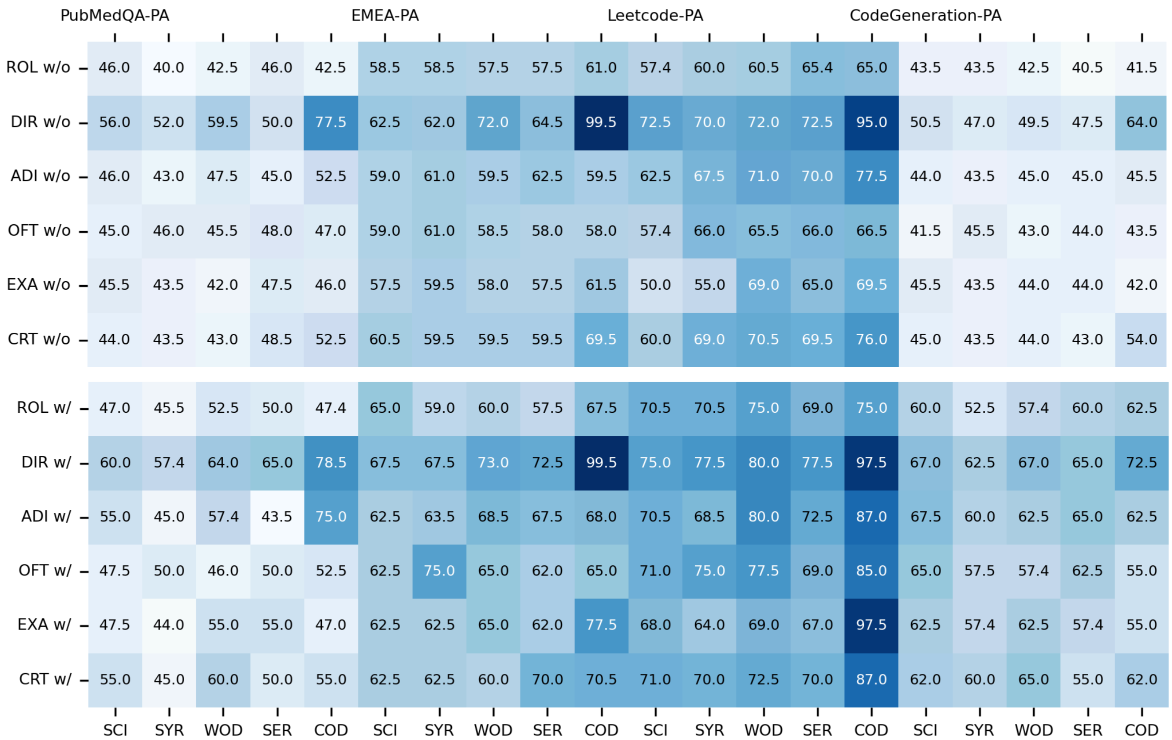

Figure 4: GPT-4o

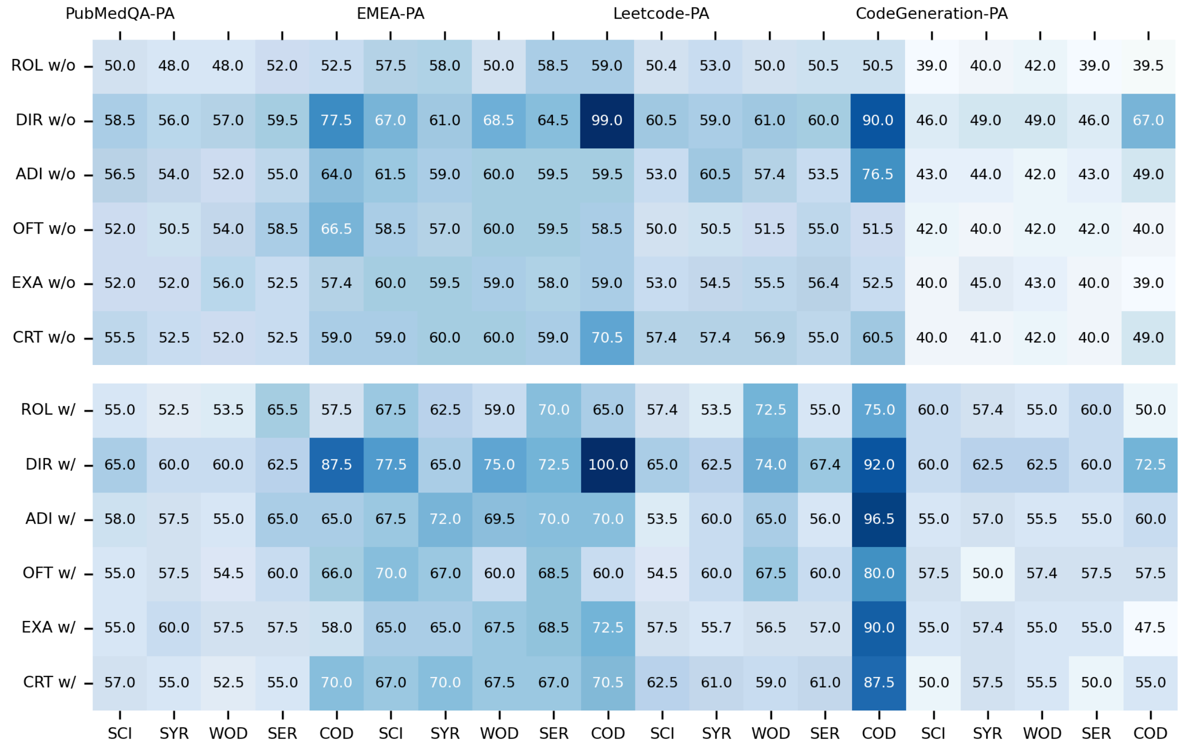

Figure 5: Claude3.7

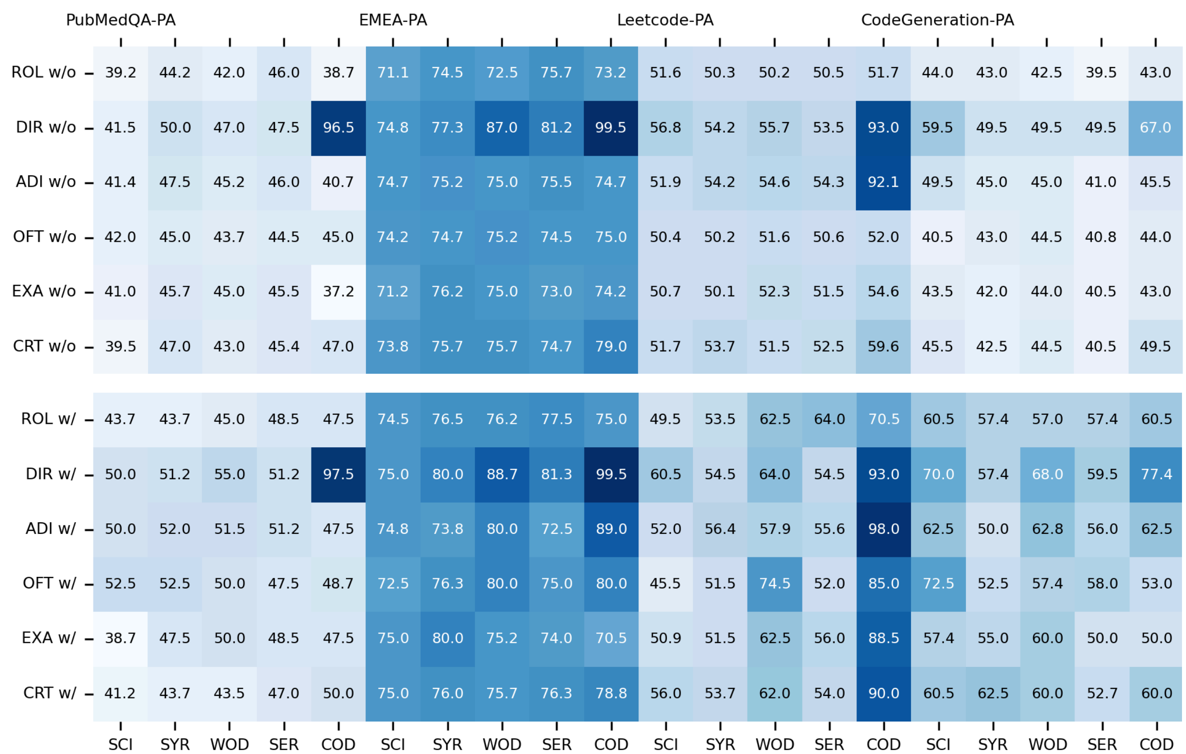

Figure 6: Qwen2.5

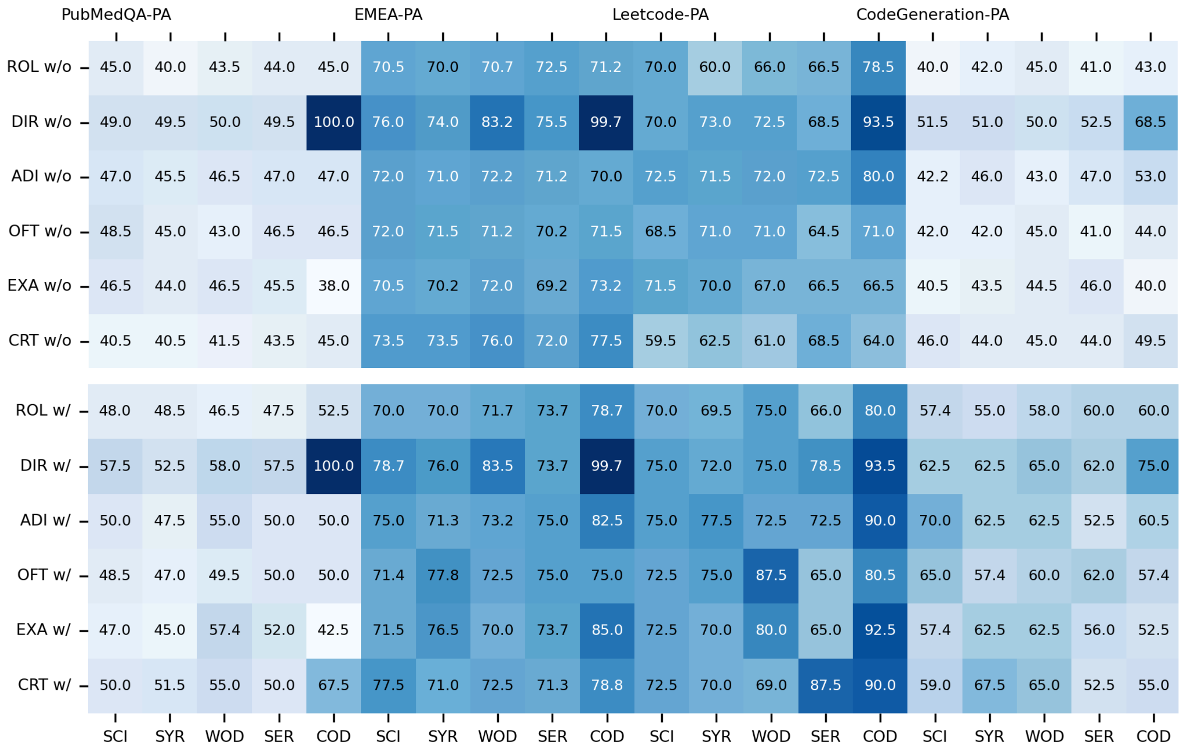

Figure 7: LLaMA3-70B

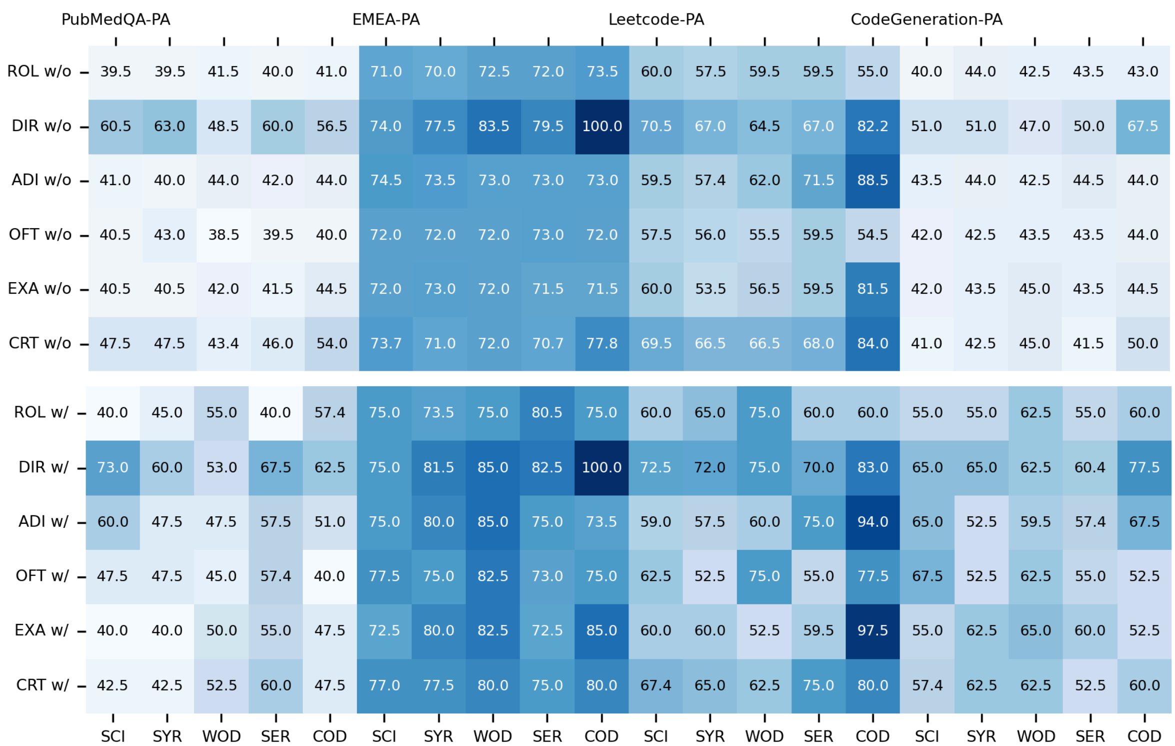

Figure 8: Gemma3-12B