CARE: Enhancing LLM Instruction Following via Dual-Agent Prompt Refinement

Nardine Basta[a,*], **Benjamin Zi Hao Zhao**[a], **Muhammad Ikram**[a] **and Mohamed Ali Kaafar**[a]

[a]Macquarie University, Sydney, Australia

Abstract. Prompt engineering is crucial for optimizing the performance of Large Language Models (LLMs), yet it remains a manual and resource-intensive process that requires multiple iterations of trial and error. Current automated prompt enhancement approaches face key challenges in preserving component relationships, managing computational requirements, and maintaining optimization traceability. This paper introduces CARE (Comprehensive Analyzer & REfiner), an LLM-based dual-agent framework that models prompt enhancement as a staged transformation pipeline with explicit validation constraints. CARE tackles three fundamental challenges: prompt decomposition with interleaved dependencies, component interference in LLM processing, and semantic drift during refinement. The framework enables reliable prompt enhancement in a single iteration through systematic component extraction and rule-based transformations. Evaluation using established benchmarks demonstrates consistent improvements across diverse LLM architectures.

1 Introduction

While LLMs demonstrate remarkable capabilities across diverse natural language processing tasks, realizing their full potential requires carefully crafted prompts. For non-experts, prompt engineering remains resource-intensive, requiring multiple trial-and-error iterations, especially for complex tasks that require precise instruction following [8].

Automated prompt optimization has emerged as a crucial research direction, with several approaches recently proposed. However, significant challenges remain: (1) lack of systematic mechanisms to preserve the relationships of the prompt components during optimization, as seen in gradient-based [11] and evolutionary approaches [4]; (2) high computational cost from multiple LLM calls in iterative optimization [15]; and (3) insufficient traceability during optimization [13, 4], making it difficult to validate semantic preservation.

To address these challenges, we present CARE, an LLM-based multi-agent framework[1] that implements prompt optimization as a staged transformation pipeline with validation constraints. Our theoretical contribution conceptualizes prompt refinement as a constrained graph transformation, where instructions are nodes with typed dependency edges that must be explicitly preserved during enhancement. The framework addresses three challenges: (1) prompt decomposition, where instructions contain interleaved components with implicit dependencies; (2) component interference, where proximity between instructions affects LLM processing; and (3) semantic drift, where refinements can alter original requirements.

CARE implements a dual-agent architecture with distinct analysis and refinement phases. The *analysis agent* extracts components and maps dependency relationships through three tiers of analysis (instruction decomposition, constraint extraction, and linguistic pattern identification). The *refinement agent* then applies systematic enhancements based on this structured representation, enforcing semantic preservation through explicit validation rules. This approach provides verifiable prompt improvement while addressing fundamental limitations in current research. Our contributions include:

1. A graph-theoretic model for representing prompts as directed dependency networks that enables formal analysis of instruction relationships and their dependencies.
2. A systematic prompt enhancement method that transforms unstructured instructions into structured formats while preserving semantic intent through dependency-aware transformation.
3. A validation methodology with formal verification criteria that ensures equivalence between original and refined prompts through explicit constraint checking.
4. A dual-agent architecture that decouples prompt decomposition from enhancement, enabling traceable transformations and verification boundaries.
5. A deterministic refinement approach that achieves performance improvements in a single processing cycle, eliminating the computational cost of iterative optimization.

The effectiveness of this approach is demonstrated through rigorous evaluation using established benchmarks [21, 9], showing consistent improvements across different LLM architectures, particularly for complex, multi-step instructions.

2 Related Work

Recent innovations in prompt optimization have emerged through text-gradient approaches. LLM-as-Optimizer [18], TextGrad [12], and OPRO [19] conceptualize prompt modifications as differentiable operations, enabling optimization through estimated gradients. While elegant mathematically, these methods typically employ iterative optimization without explicit component tracking or dependency preservation, and require multiple iterations to converge.

ProTeGi [11] employs a gradient descent-inspired approach using LLMs for error feedback; however, its reliance on simple text-based mutations fails to capture higher-level prompt structure and domain knowledge, leading to inefficient exploration of the prompt space. While PromptAgent [15] introduces strategic planning via Monte Carlo Tree Search [6], it lacks mechanisms for effectively incorpo-

* Corresponding Author. Email: nardine.basta@mq.edu.au.

[1] The code is at https://anonymous.4open.science/r/CARE_ECAI-DBE4

rating domain expertise and struggles with balancing multiple competing objectives like instruction-following and linguistic quality.

EvoPrompt [4] applies evolutionary algorithms by representing prompts as gene sequences, but employs insufficiently granular genetic operations with limited modification transparency. SAMMO [13] addresses these limitations through function graph representations, introducing significant implementation complexity and relying on structural assumptions that lack generalizability.

3 The CARE Framework

Figure 1 illustrates CARE's sequential pipeline of agents, with each agent building upon the output of its predecessor. The prompt enhancement process begins with the *Analyzer* agent, which extracts structured prompt representations that capture atomic instructions, their dependencies, and constraints through a three-tier analysis. This analysis serves as the foundation for the *Refiner* agent that implements targeted enhancements through a staged transformation pipeline with explicit validation requirements. Both agents are implemented using Claude 3.5 Sonnet [1], leveraging its strong capabilities in context comprehension and nuanced text generation.

3.1 Multi-Agent Architecture

In the following, we present our CARE framework, which consists of a dual-agent architecture that addresses fundamental limitations in monolithic prompt refinement systems. Our design enables explicit validation boundaries and transformation transparency.

Formal Agent Architecture: The system consists of two distinct agents, each operating on separate computational domains:

$$\mathcal{M} = \langle A, R, \mathcal{S} \rangle \tag{1}$$

where $\mathcal{M}$ represents the complete multi-agent system, $A : \mathcal{P} \to \mathcal{S}$ is the Analyzer function mapping prompts from the prompt space $\mathcal{P}$ to an intermediate structured representations or schema, $\mathcal{S}$. Here, $R : \mathcal{S} \to \mathcal{P}$ is the Refiner function mapping structured representations to refined prompts.

The schema space $\mathcal{S}$ is formally defined as:

$$\mathcal{S} = \langle I, G, C \rangle \tag{2}$$

where I represents the set of extracted atomic instructions, $G = (V, E, \tau)$ encodes the dependency graph with vertex set V corresponding to instructions, edge set E representing dependencies, and typing function $\tau : E \to \{E_s, E_c, E_r\}$ categorizing edges as sequential, conditional, or reference dependencies. The component C specifies the hierarchy of constraints comprising explicit, implicit, and cross-instruction requirements.

Validation Architecture: The system implements three sequential validation functions:

$$V_1 : A \to \mathcal{S}(P_a) \qquad \text{(completeness verification)} \tag{3}$$

$$V_2 : \mathcal{S}(P_a) \to R \qquad \text{(structural integrity)} \tag{4}$$

$$V_3 : R \to P_r \qquad \text{(semantic preservation)} \tag{5}$$

At checkpoint V_1, the Analyzer agent implements completeness verification (see Listing 3.1), ensuring all instructions from P are identified and represented in $\mathcal{S}(P_a)$. This checkpoint prevents information loss during decomposition. Checkpoint V_2 operates at the agent boundary (see Listing 3.7), validating schema conformance and dependency graph consistency before processing. Checkpoint V_3 executes post-refinement validation (see Listing 3.8), verifying that P_r preserves all semantic requirements through constraint checking against original components.

3.2 Analyzer Agent

The Analyzer agent implements a systematic decomposition framework that transforms unstructured prompts into explicit instruction graphs with dependency tracking. Our approach builds on findings that structured decomposition significantly improves LLM instruction following [17]. Through a carefully engineered prompt template, the framework performs multi-dimensional analysis across three key aspects: instruction decomposition, constraint analysis, and linguistic pattern analysis.

Instruction Decomposition: Drawing inspiration from program dependency graphs [3], we guide the LLM to decompose complex prompts into atomic components with explicit relationships. Given an input prompt P, the model constructs a structured representation:

$$I(P) = \{(i_k, D_k, R_k) | k \in [1, n]\} \tag{6}$$

where i_k represents atomic instructions identified by the LLM with unique identifiers. To capture instruction relationships, we construct a dependency graph $G = (V, E)$ where vertices V represent instructions and edges E capture their dependencies. D_k encodes these relationships through a typed edge system comprising sequential edges (E_s) for strict ordering requirements between instructions (e.g., "first do X, then Y"), conditional edges (E_c) for execution dependencies where one instruction's output affects another's execution path, and reference edges (E_r) for information flow between instructions that share data or formatting requirements, where $E = E_s \cup E_c \cup E_r$. R_k defines the set of completion requirements and success criteria that must be satisfied for instruction i_k to be considered properly executed (e.g., formatting rules, numerical constraints).

The Analyzer is directed through a comprehensive prompt template that systematically extracts structured representations as shown in Listing 3.1. The dependency analysis, shown in Listing 3.2, extends this framework through enabling explicit tracking of instruction relationships and identification of potential interference points where multiple dependencies intersect.

Listing 3.1: Core structure section of Analyzer prompt

```
Analyze the manually crafted prompt. Understand the
     intent, context, and overall objective. Extract
     these key elements as a JSON Dictionary:
1. Role/Identity: The persona the AI is asked to
     emulate. Expected values: Specific role (e.g.,
     "Data Scientist"), or "N/A" if not specified.
2. Context: Background information or setting for the
     prompt. Expected values: Brief context
     description, or "N/A" if not provided.
3. Objective: Clarify user's objective (e.g., concise
     output, formatting rules, word limits).
4. Prompt/Query: The specific question/task asked.
```

Constraint Analysis: Building on the dependency structure, the framework implements a three-tier constraint analysis through a carefully engineered prompt, shown in Listing 3.3, that guides the Analyzer in identifying and categorizing requirements. Through systematic analysis, the LLM extracts explicit constraints that are directly stated in the prompt (e.g., "use less than 10 capital words", "wrap in quotes"), implicit constraints derived from context (e.g., formatting consistency, logical ordering), and cross-instruction constraints that emerge from component interactions (e.g., maintaining consistent style across sections). While dependency analysis focuses

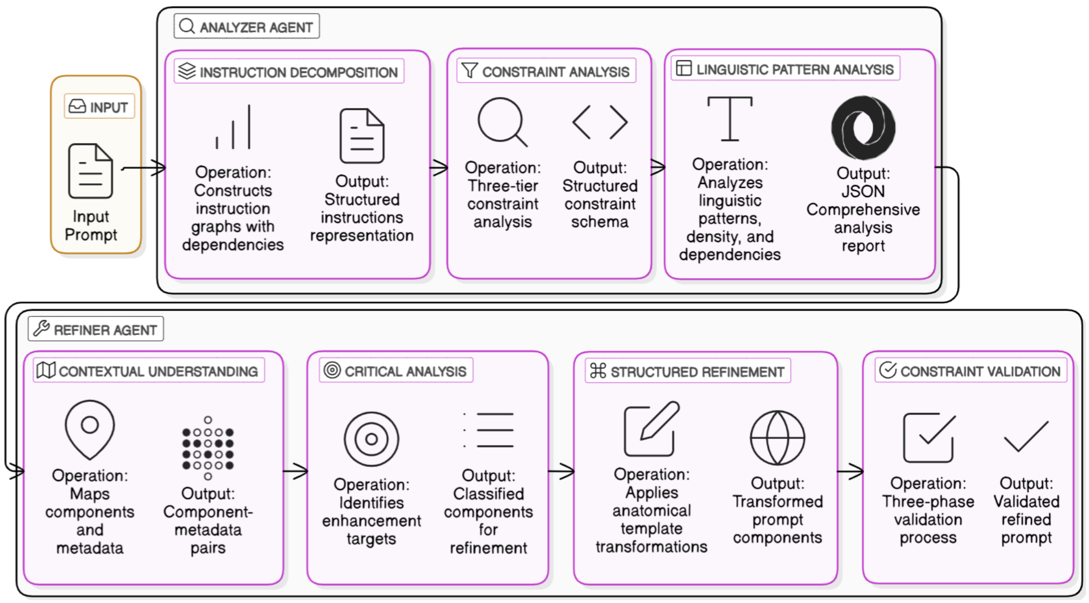
ANALYZER AGENT
INPUT
Input Prompt
INSTRUCTION DECOMPOSITION
Operation: Constructs instruction graphs with dependencies
Output: Structured instructions representation
CONSTRAINT ANALYSIS
Operation: Three-tier constraint analysis
Output: Structured constraint schema
LINGUISTIC PATTERN ANALYSIS
Operation: Analyzes linguistic patterns, density, and dependencies
Output: JSON Comprehensive analysis report
REFINER AGENT
CONTEXTUAL UNDERSTANDING
Operation: Maps components and metadata
Output: Component-metadata pairs
CRITICAL ANALYSIS
Operation: Identifies enhancement targets
Output: Classified components for refinement
STRUCTURED REFINEMENT
Operation: Applies anatomical template transformations
Output: Transformed prompt components
CONSTRAINT VALIDATION
Operation: Three-phase validation process
Output: Validated refined prompt

Figure 1: CARE Architecture: The Analyzer agent performs instruction decomposition, constraint analysis (explicit, implicit, and cross-instruction), and linguistic pattern analysis. The Refiner agent implements contextual understanding, critical analysis, identifying refinement targets, structured refinement, and constraint validation (presence verification, structural validation, and constraint adherence verification).

Example: Prompt refinement

Original prompt

Write a resume for Matthias Algiers. Use words with all capital letters to highlight key abilities. Make sure that words with all capital letters appear less than 10 times. Wrap the entire response with double quotation marks.

Refined prompt

Role: You are a resume writer.
Objective: Your objective is to create a resume for Matthias Algiers that highlights key abilities using words with all capital letters, but with fewer than 10 such words.
Context: Matthias Algiers is a professional seeking a new job opportunity.
Instructions:1. Gather relevant information about Matthias Algiers' background, experience, and skills. 2. Identify key abilities to highlight using words with all capital letters. 3. Structure the resume with the following sections: a. Header with Matthias Algiers' name and contact information. b. Professional Summary highlighting Matthias Algiers' key strengths and qualifications. c. Work Experience detailing previous job roles, responsibilities, and achievements. d. Education listing academic qualifications and degrees. e. Skills and Certifications showcasing relevant skills and certifications. 4. Use no more than 9 words with all capital letters to emphasize key abilities. 5. Ensure the resume is well-formatted, concise, and tailored to Matthias Algiers' target industry or job role. 6. Wrap the resume content with double quotation marks.

Figure 2: Example demonstrating CARE's prompt refinement process, showing transformation from an unstructured prompt to a structured format while preserving original requirements.

on execution relationships, constraint analysis captures specific requirements and validation criteria for each instruction.

Linguistic Pattern Analysis: The framework implements systematic linguistic analysis, shown in Listing 3.4, to guide the agent in examining structural patterns and semantic relationships. At the syntactic level, the LLM analyzes quantitative metrics including instruction density (instructions per sentence), text complexity (through readability scores and sentence length distribution), and specialized vocabulary usage. At the semantic level, it examines instruction interactions by identifying points where requirements overlap. This complements the previous analyses by focusing on expression patterns rather than execution dependencies or validation requirements.

The analysis, captured in a JSON schema, provides the Refiner with a comprehensive understanding of the prompt's operational dependencies, validation requirements, and linguistic structure.

3.3 Refiner Agent

Given an analyzed prompt P_a with its structured JSON representation, the refinement process executes through four stages. Figure 2 shows an example of how an unstructured prompt is transformed through this process into a structured format with explicit components and preserved requirements.

Contextual Understanding: The first stage establishes a comprehensive refinement scope by processing the Analyzer's output:

$$I(P_a) = \{(c, \text{meta}) | c \in \text{Components}(P_a)\} \tag{7}$$

where c represents atomic components (role, context, objective, instructions) and meta encapsulates their attributes, including source text, constraints, dependencies, and formatting specifications. This structured mapping enables precise transformation tracking while

Listing 3.2: Instruction decomposition section

```
5. Instructions_Analysis: {
  "atomic_instructions": [
    { "id":"unique identifier",
      "text":"instruction text",
      "type":"primary | secondary | conditional",
      "dependencies":[
        { "instruction_id": "string",
          "dependency_type": "sequential |
              conditional | reference",
          "description": "string" }],
      "interleaved_with":["interleaved instructions
          ids"],
      "completion_requirements":["Requirements
          list"]}],
  "dependency_graph":{"nodes":["instruction ids"],
    "edges":[{ "from":"source instruction id",
        "to":"target instruction id",
        "type":"dependency type"}]}}
```

Listing 3.3: Constraint analysis section

```
6. Constraints: {
  "explicit": ["directly stated constraints"],
  "implicit": ["constraints derived from context or
      dependencies"],
  "cross_instruction": [
    { "instructions": ["related instruction ids"],
      "constraint": "shared constraint
          description"}]}
```

preserving essential component relationships and validation requirements. Listing 3.7 shows the section of the Refiner prompt implementing the contextual understanding.

Critical Analysis: The second stage systematically identifies required refinements based on the Analyzer's structured output. Through carefully engineered prompts, the agent analyzes transformation requirements across multiple dimensions where $c \in \text{Components}(P_a)$:

$$A(c) = \begin{cases} \text{Dependency}(c) & \text{order/concurrency} \\ \text{Constraint}(c) & \text{validation} \\ \text{Structure}(c) & \text{organization} \\ \text{Semantic}(c) & \text{content preservation} \end{cases} \quad (8)$$

For dependency requirements, the LLM analyzes the instruction dependency graph to determine the execution order, group concurrent instructions, and handle interleaving points. For constraint requirements, it processes both explicit (directly stated rules) and implicit constraints (derived from context), ensuring compatibility and preservation. Structural requirements focus on component organization following the anatomical template: role definition, context establishment, objective specification, and instruction sequencing. Semantic requirements ensure the preservation of all atomic instructions and their completion criteria.

The critical analysis provides transformation directives that guide the subsequent refinement stage. Each directive includes required modifications, preservation rules, and validation criteria, ensuring that refinements maintain semantic fidelity while improving instruction clarity. This comprehensive requirements analysis enables systematic and verifiable prompt enhancement. This stage is operationalized through prompt directives as specified in Listing 3.6.

Listing 3.4: Linguistic analysis section

```
7. Linguistic_Analysis: {
  "word_count": "integer",
  "sentence_count": "integer",
  "avg_words_per_sentence": "float",
  "readability_score": "float",
  "tone": "tone description",
  "specialized_vocabulary": ["domain-specific terms"],
  "instruction_density": "instructions per sentence",
  "interleaving_patterns": [
    { "instructions": ["involved instruction ids"],
      "pattern_type": "pattern description",
      "description": "detailed explanation"}]}
```

Listing 3.5: Contextual understanding of Refiner prompt

```
You are an expert prompt engineer tasked with
    optimizing prompt effectiveness through a
    structured transformation while maintaining
    semantic integrity.
INPUT:
Original Prompt: {original_prompt}
Analyzer Output:
- Role: {analysis.get('Role/Identity', 'N/A')}
- Context: {analysis.get('Context', 'N/A')}
- Objective: {analysis.get('Objective', 'N/A')}
- Instructions_Analysis:
    {analysis.get('Instructions_Analysis', 'N/A')}
- Constraints: {analysis.get('Constraints', 'N/A')}
- Linguistic_Analysis:
    {analysis.get('Linguistic_Analysis', 'N/A')}
```

Listing 3.6: Transformation requirements section

```
TRANSFORMATION REQUIREMENTS:
1. Dependency-Aware Restructuring:
  - Analyze the dependency graph from
      Instructions_Analysis
  - Order instructions to respect sequential
      dependencies
  - Group concurrent instructions appropriately
  - Preserve critical path integrity
  - Handle instruction interleaving points explicitly
2. Constraint Integration:
  - Address all explicit constraints
  - Incorporate implicit constraints
  - Maintain cross-instruction constraints
  - Validate constraint compatibility
3. Structural Enhancement:
  a. Component Organization:
    - Role definition
    - Context establishment
    - Objective specification
    - Instruction sequence (following dependency
        graph)
  b. Section Demarcation:
    - Clear section boundaries
    - No cross-section repetition
    - Explicit transition markers
4. Semantic Preservation:
  - Maintain all atomic instructions
  - Preserve completion requirements
  - Respect instruction relationships
  - Ensure constraint satisfaction
```

Structured Refinement: The third stage implements transformations identified in the critical analysis through a structured template validated by empirical studies of LLM instruction following [21] and prompt design principles [7]. The refinement process follows an anatomical structure:

$$E(C) = \text{Role}\|\text{Context}\|\text{Objective}\|\text{Instructions} \quad (9)$$

where C represents the complete set of analyzed components from the previous stages and $\|$ represents structured concatenation with explicit boundary preservation. For each component, the LLM applies targeted transformations guided by the previous stage's requirements: standardizing role definitions into explicit persona statements, consolidating context elements into prerequisite information blocks (following constraint specifications), reformulating objectives as concrete task specifications, and transforming instructions into enumerated requirements. This structured approach is implemented through a specific prompt as shown in Listing 3.7.

Constraint Validation: This stage implements comprehensive

Listing 3.7: Output format section

```
OUTPUT FORMAT:
Generate a refined prompt that:
1. Follows the hierarchical structure: Role ->
     Context -> Objective -> Instructions
2. Preserves all dependencies and relationships
3. Makes instruction sequences explicit
4. Maintains clear section boundaries
5. Integrates all constraints systematically
```

validation through constraint embedding in the refinement template:

$$V(P_r, P_a) = \bigwedge_{c \in C} \text{Preserved}(c, P_r) \tag{10}$$

where Preserved(c, P_r) executes a three-phase validation process to verify component preservation. The process encompasses: (1) presence verification through exact string matching between original and refined instructions, (2) structural validation ensuring maintained section boundaries and formatting requirements, and (3) constraint adherence verification for explicit requirements, including formatting rules and enumeration patterns. This validation mechanism ensures semantic fidelity throughout the refinement process through explicit template directives rather than post-hoc verification.

The validation is implemented through prompt directives that enforce semantic preservation requirements, as shown in Listing 3.8.

Listing 3.8: Validation requirements section

```
VALIDATION REQUIREMENTS:
1. All atomic instructions from the original prompt
     are preserved
2. Dependency graph integrity is maintained
3. Explicit and implicit constraints are satisfied
4. No new requirements are introduced
5. Instruction relationships are preserved
```

4 Experimental Setup

4.1 Dataset

We evaluate our framework using the dataset introduced by Google Research's instruction following evaluation framework [21], which consists of verifiable instruction prompts designed to assess the instruction-following capabilities of LLMs. It comprises 25 types of verifiable instructions and is constructed around 500 prompts, with each prompt containing one or more verifiable instructions.

The dataset is structured as a JSON file, where each line represents a prompt with associated metadata. The format of each entry is as follows: { **key:** <*unique_identifier*>, **prompt:** <*instruction_text*>, **instruction_id_list:** [<*list_of_instruction_types*>], **kwargs:** [<*list_of_instruction_parameters*>] }, where the `instruction_id_list` field contains identifiers for the types of instructions in the prompt, such as `punctuation:no_comma` or `length_constraints:number_words`. The `kwargs` field provides additional parameters for each instruction, such as the number of required placeholders or word count constraints.

4.2 Models

We selected a diverse set of state-of-the-art LLMs that represent different architectures, training paradigms, and accessibility levels:

Anthropic's Claude 3.5 Sonnet [1]: A proprietary model known for its strong instruction-following capabilities.

OpenAI's GPT-4 [10]: A capable, multi-modal model that has demonstrated exceptional performance across various tasks.

Meta's LLaMA 3.1 70B Instruct [14]: An advanced version of open-source LLaMA, fine-tuned for instruction following.

Mistral AI's Mixtral 8x7B Instruct [5]: A sparse mixture-of-experts model optimized for instruction-based tasks.

DeepSeek AI's DeepSeek-67B [2]: A recently released open-source model trained on a diverse corpus of data.

4.3 Baselines

We compare CARE against SAMMO [13], a compile-time prompt optimization framework using dynamic function graphs with mutation-based optimization. Using SAMMO's public implementation with default parameters, comparisons are limited to Anthropic's Claude and OpenAI's GPT models due to implementation constraints, while CARE extends to additional models including open-source LLMs. Both target pre-deployment optimization but differ in approach: CARE employs specialized agents for analysis and refinement, while SAMMO uses graph-based transformations with evolutionary search. SAMMO serves as a suitable baseline due to its output compatibility with the instruction-following dataset's complex metadata structure.

4.4 Evaluation

4.4.1 Instruction Following Assessment

For evaluating instruction adherence, we adopt Google Research's instruction verification framework [21], which implements a systematic approach through categorical classification and deterministic rule-based verification. This framework, along with its associated verifiable prompts dataset, enables precise quantification of refinement impact on instruction clarity and compliance. We employ three key metrics from this framework to assess refinement effectiveness:

- *Prompt-level Accuracy*: Measures the proportion of prompts where all verifiable instructions are followed, evaluating CARE's ability to maintain complete instruction adherence.
- *Instruction-level Accuracy*: Captures the proportion of individual instructions followed across all prompts, providing insight into CARE's effectiveness at handling specific instruction types.
- *Category-specific Accuracy*: Evaluates performance across different instruction categories, enabling analysis of CARE's effectiveness for specific instruction categories.

4.4.2 Linguistic Quality Assessment

To evaluate linguistic quality, we utilize G-Eval's linguistic assessment framework [9], which employs GPT-4 for analysis of output quality across multiple linguistic dimensions. The framework implements carefully crafted prompts to guide GPT-4 in analyzing text across four aspects:

- *Coherence*: Assesses logical flow and idea connectivity, ensuring refinements maintain clear instruction relationships (1-5 scale).
- *Consistency*: Measures uniformity in style and factual alignment (1-5 scale) to verify that refinements preserve original intent.
- *Relevance*: Evaluates alignment with prompt objectives (1-5 scale) and validates that refinements maintain task focus.
- *Fluency*: Examines grammatical correctness (1-3 scale), ensuring refinements enhance rather than degrade linguistic quality.

4.4.3 Ablation Study

We evaluated CARE through component-level experiments to assess the individual contributions of its modules: (a) an independent evaluation of the Analyzer to measure decomposition quality, (b) an assessment of the Refiner's performance *with* and *without* structured input from the Analyzer, and (c) an evaluation of the validation stage's effectiveness using redundant data extraction mechanisms. The results from our experiments demonstrate that each component makes a substantive contribution to the overall system performance.

4.5 Experiments

We begin by preparing our dataset, which involves generating responses to both original and CARE-refined prompts across all five selected models (cf. § 4.2).

5 Results and Analysis

We evaluated CARE's computational efficiency by measuring token usage and response latency across benchmark tasks. CARE employs a deterministic single-pass approach, requiring only 8,000–16,000 tokens per prompt, in contrast to SAMMO's 20,000–50,000 tokens over 5–10 iterations. This design yields a latency of 3–4 seconds, compared to SAMMO's 15–20 seconds, thereby eliminating the overhead associated with iterative convergence.

5.1 Cross-model Performance

Next, we evaluated CARE's effectiveness through cross-model performance analysis, comparing instruction-following between the *original* and *refined* prompts across all evaluated models. As shown in the upper portion of Table 1, this analysis provides a nuanced perspective on CARE's impact across different LLM architectures. The results indicate that CARE generalizes effectively, delivering substantial gains for both proprietary and open-source models. Unlike SAMMO, whose evaluation is limited to Anthropic and GPT models, CARE demonstrates consistent improvements across all tested architectures, including Mixtral (+7.4%), LLAMA (+3.1%), and DeepSeek (+1.6%). Moreover, the magnitude of improvement appears inversely correlated with the base model's initial performance, suggesting that CARE is particularly effective in addressing limitations in less optimized models.

Notably, GPT-4 exhibits the largest (+10. 16%), suggesting that larger and more sophisticated models can benefit disproportionately from refined prompts. This gain may stem from GPT-4's advanced contextual understanding, allowing it to more effectively leverage the additional structure and clarity introduced by CARE.

Moreover, LLaMA-3's significant improvement in instruction-following performance is particularly noteworthy given its open-source nature, highlighting the accessibility of CARE as a cost-effective way to enhance performance in resource-constrained scenarios. Mixtral's 7.4% increase, despite its smaller size, suggest that sparse mixture-of-experts architectures can efficiently utilize refined prompts, potentially through the activation of distinct subnetworks tailored to varied instruction types.

The consistent improvements in instruction-level accuracy across all evaluated models indicate that CARE's refinements capture fundamental aspects of instruction clarity that transcend specific model architectures. This universality suggests that our refinement strategies are tapping into core principles of natural language understanding and task specification.

5.2 Instruction Category Analysis

Instruction category analysis in the lower portion of Table 1 reveals several key patterns. CARE shows particular strength in complex instruction categories like Combination tasks (improving Anthropic from 0.862 to 0.969) and Length Constraints (0.580 to 0.874), significantly outperforming SAMMO in these areas. For simpler categories like Case Change and Detectable Content, both CARE and SAMMO achieve comparable improvements, suggesting these tasks may be closer to saturation. Notably, CARE maintains or improves performance across all categories, while SAMMO shows occasional performance degradation (e.g., Case Change in GPT: 0.888 to 0.786).

The significant improvement in the 'combination' category, which includes tasks like repeating prompts or providing multiple responses (e.g., GPT-4's jump from 44.62% to 92.31%) reveals a weakness in current LLM prompt processing: handling compound instructions [20]. This improvement suggests that LLMs may process instructions sequentially, struggling to maintain multiple objectives simultaneously. CARE's success in this category can be attributed to its strength in disambiguating complex, multi-step instructions into clearer, sequential steps—a strategy that aligns well with the transformer architecture's strength in processing sequential information.

The mixed results in the 'punctuation' category, particularly Mixtral's slight decrease in accuracy, reveal a nuanced interaction between prompt refinement and model behavior. This indicates that some refinements can introduce competing priorities for the model, perhaps by overemphasizing other aspects of the instruction at the expense of punctuation adherence. It suggests that LLMs may have a finite "instruction bandwidth" [16], requiring balance in prompt design that maintains emphasis across all instruction categories.

The persistent challenge in the 'length constraints' category across all models points to a fundamental difficulty in precise numerical control of language model outputs. This could be related to the tension between adhering to strict word counts and maintaining coherent, natural language generation. This is rooted in the core training objective of LLMs, which typically focuses on next-token prediction rather than holistic output planning.

5.3 Impact on Linguistic Quality

The linguistic quality evaluation presented in Table 2 demonstrates consistent improvements across models and metrics, with both CARE and SAMMO enhancing baseline prompt quality. For Anthropic, CARE achieves improvements in coherence (4.563 to 4.747), consistency (4.492 to 4.651), and relevance (4.479 to 4.705), outperforming SAMMO's more modest gains (coherence: 4.563 to 4.639, consistency: 4.492 to 4.571, relevance: 4.479 to 4.483). Similarly, for GPT, CARE shows stronger improvements across metrics compared to SAMMO, particularly in coherence (4.642 to 4.753 vs 4.642 to 4.601) and relevance (4.429 to 4.692 vs 4.429 to 4.524).

The modest' meaningful linguistic improvements are particularly noteworthy given that the evaluation dataset consists of carefully crafted, high-quality examples specifically designed for instruction following. This challenging evaluation context demonstrates CARE's ability to enhance even well-formulated prompts, suggesting the potential for more substantial improvements when applied to typical user-generated prompts.

5.4 Ablation Study Results

Component analysis reveals each stage's contribution. The Analyzer alone improves prompt structure clarity by 15–20% in readabil-

Table 1: *Instruction following* results for the original and CARE refined prompts across different LLM architectures.

Model	**Anthropic**			**Mixtral**		**LLAMA**		**GPT**			**DeepSeek**	
Prompt	**Original**	**SAMMO**	**Refined**	**Original**	**Refined**	**Original**	**Refined**	**Original**	**SAMMO**	**Refined**	**Original**	**Refined**
Prompt-level	0.741	0.767	0.889	0.508	0.582	0.806	0.837	0.750	0.765	0.852	0.771	0.787
Instruction-level	0.820	0.836	0.927	0.612	0.689	0.868	0.886	0.827	0.830	0.896	0.841	0.851
Case Change	0.820	0.820	0.921	0.798	0.730	0.910	0.910	0.888	0.786	0.910	0.831	0.854
Combination	0.862	0.862	0.969	0.231	0.385	0.569	0.877	0.446	0.738	0.923	0.569	0.800
Detectable Content	1.000	1.000	1.000	0.717	0.755	0.981	0.925	0.962	0.981	0.962	0.981	0.943
Detectable Format	0.930	0.936	0.904	0.764	0.822	0.930	0.911	0.924	0.930	0.936	0.955	0.924
Keywords	0.761	0.859	0.926	0.724	0.828	0.908	0.865	0.834	0.853	0.871	0.816	0.834
Language	1.000	0.968	0.935	0.548	0.806	0.968	1.000	0.935	0.968	1.000	0.968	1.000
Length Constraints	0.580	0.629	0.874	0.524	0.615	0.720	0.734	0.692	0.713	0.776	0.657	0.692
Punctuation	0.712	0.985	0.955	0.242	0.288	0.985	1.000	0.864	0.667	0.894	0.985	0.864
Start-End	0.955	0.940	0.970	0.597	0.731	0.925	0.985	0.970	0.925	0.970	0.985	0.955

Table 2: Linguistic feature comparison between the original and CARE refined prompts for different LLMs.

Eval.	**Anthropic**			**Mixtral**		**LLAMA**		**GPT**			**DeepSeek**	
Metric	**Original**	**SAMMO**	**Refined**	**Original**	**Refined**	**Original**	**Refined**	**Original**	**SAMMO**	**Refined**	**Original**	**Refined**
Coherence	4.563	4.639	4.747	4.231	4.244	4.505	4.571	4.642	4.601	4.753	4.606	4.575
Consistency	4.492	4.571	4.651	4.043	4.052	4.470	4.516	4.534	4.510	4.638	4.421	4.515
Fluency	2.856	2.922	2.943	2.867	2.919	2.859	2.942	2.892	2.927	2.919	2.841	2.900
Relevance	4.479	4.483	4.705	3.949	3.949	4.351	4.351	4.429	4.524	4.692	4.384	4.489

ity metrics but shows minimal instruction-following gains (+2–3%) without the Refiner. The Refiner without structured input achieves only 40–50% of its full performance improvement, demonstrating the critical importance of systematic decomposition. Validation stages prevent 8–12% of potential semantic drift cases, though occasionally exhibit over-conservative behavior with roughly 5–8% false rejection rate in complex multi-constraint scenarios where valid paraphrasing is flagged as semantic drift.

5.5 *Robustness and Reliability*

To evaluate the robustness of CARE, we conducted consistency analyses across multiple runs of structured analysis tasks. The goal was to assess the stability of the model outputs in the face of inherent stochasticity in language models. Our methodology involved executing repeated trials with the evaluated models and measuring the consistency of their outputs under identical prompt conditions.

The results demonstrate a high degree of consistency, with approximately 90–95% agreement observed across runs. This level of stability significantly exceeds that of typical free-form generation approaches, which are more susceptible to variability. We attribute this improvement to CARE's structured design and explicit validation constraints, which effectively reduce output variability while mitigating–but not entirely eliminating–the intrinsic limitations of language models. Furthermore, CARE's modular architecture enhances reliability by allowing flexible adaptation through targeted modifications of analysis and refinement prompts without disrupting the overall framework. This adaptability supports continued robustness as models evolve or as new use cases arise.

6 Conclusion

This paper introduces the multi-agent prompt refinement framework, CARE, a novel approach to automated prompt refinement that demonstrates significant potential for enhancing LLM performance across diverse architectures. Our comprehensive evaluation, utilizing a dataset of verifiable prompt instructions and the G-Eval framework, reveals consistent improvements in instruction-following while maintaining or enhancing linguistic quality.

Our analysis uncovers several key phenomena in LLM behavior, including "refinement saturation", and "instruction affinities". These insights not only validate CARE's effectiveness but also contribute to the broader understanding of LLM functionality and prompt engineering principles. The observed "instruction interference" effect and CARE's success in refining compound instructions highlight critical areas for future LLM development and prompt design strategies.

7 Limitations

CARE fails for approximately 5-8% of cases due to: (a) overstructuring simple prompts by adding unnecessary complexity, (b) difficulty with domain-specific jargon that breaks dependency analysis, and (c) occasional reordering of instructions that changes subtle emphasis. Specific examples include: (1) A technical prompt about database optimization where CARE incorrectly restructured domain-specific workflows, (2) A prompt with cultural references that CARE misinterpreted due to a lack of contextual knowledge. The framework's effectiveness depends on the underlying LLM's knowledge base and capabilities, inheriting these limitations for specialized domains where LLMs lack expertise.

While CARE demonstrates consistent performance improvements across multiple LLMs, several limitations warrant acknowledgment. Our evaluation relies on a dataset of verifiable prompt instructions crafted by researchers, which may not fully represent the diversity and potential flaws of prompts encountered in real-world applications. This could potentially overestimate CARE's effectiveness on less optimally constructed prompts.

Our evaluation methodology faces constraints from the complex dataset specifications required by the instruction-following evaluator, necessitating unique identifiers, instruction types, and structured parameters. This complexity confined evaluation to a pre-existing dataset of 500 prompts, as creating additional conformant datasets presents significant challenges. Further, our English-focused evaluation leaves cross-lingual capabilities unexplored.

References

[1] Anthropic. Model card and evaluations for claude models. Technical report, Anthropic, July 2023.

[2] DeepSeek-AI, X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[3] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349, 1987.

[4] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

[5] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. Bou Hanna, F. Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[6] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[7] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[8] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9), 2023.

[9] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, Dec. 2023. Association for Computational Linguistics.

[10] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

[11] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[12] R. Rafailov, A. Sharma, A. Raghunathan, P. Liang, and T. Hashimoto. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[13] T. Schnabel and J. Neville. Symbolic prompt program search: A structure-aware approach to efficient compile-time prompt optimization. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

[14] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[15] X. Wang, C. Li, Z. Wang, F. Bai, H. Luo, J. Zhang, N. Jojic, E. Xing, and Z. Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

[16] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

[17] J. Wei et al. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[18] C. Yang, X. Shen, J. Zhao, S. Liu, Y. Wang, O. Huang, Z. Ye, C.-Y. Lee, S. Liu, Y. Wang, M. Lewis, Y. Yue, and H. Lipson. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

[19] S. Zhang, I. Schlag, N. Malkin, J. Choi, T. Brown, T. Goldstein, A. Gesmundo, J. Gao, A. Steiner, J. Kaplan, C. Finn, I. Sutskever, S. Welleck, M. Bellagente, and J. Sohl-Dickstein. Optimizing generative ai by backpropagating language model feedback. *Nature*, 627(8004):110–118, 2024.

[20] Z. Zhao, W. S. Lee, and D. Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, number 1387 in NIPS '23, 2023.

[21] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.