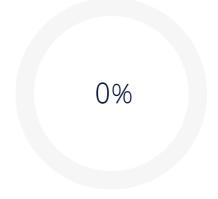
July 29th, 2025 at 02:23 UTC



Plagiarism Detection Report

Sidekicker

Plagiarism Detection



Plagiarism Types	Text Coverage	Words
Identical	0%	0
Minor Changes	0%	0
Excluded		
Omitted Words		0

Total Pages

Total Words

3575









Plagiarism

0%

Results (0)



Plagiarism Types	Text Coverage	Words
O Identical	0%	0
Minor Changes	0%	0
Excluded		
Omitted Words		0

About Plagiarism Detection

Our Al-powered plagiarism scans offer three layers of text similarity detection: Identical, Minor Changes, and Paraphrased. Based on your scan settings we also provide insight on how much of the text you are not scanning for plagiarism (Omitted words).

Identical

One to one exact word matches. Learn more

Minor Changes

Words that hold nearly the same meaning but have a change to their form (e.g. "large" becomes "largely"). <u>Learn more</u>

Omitted Words

The portion of text that is not being scanned for plagiarism based on the scan settings. (e.g. the 'Ignore quotations' setting is enabled and the document is 20% quotations making the omitted words percentage 20%) <u>Learn more</u>

Copyleaks Shared Data Hub

Our Shared Data Hub is a collection of millions of user-submitted documents that you can utilize as a scan resource and choose whether or not you would like to submit the file you are scanning into the Shared Data Hub. Learn more

Filtered and Excluded Results

The report will generate a complete list of results. There is always the option to exclude specific results that are not relevant. Note, by unchecking certain results, the similarity percentage may change. <u>Learn more</u>

Current Batch Results

These are the results displayed from the collection, or batch, of files uploaded for a scan at the same time. Learn more

Abstract

Ambiguous prompts are one of the key issues in the real-world application of large language models (LLMs), which potentially generate incorrect outputs and consume immense computing resources. We propose a holistic three-stage approach to preemptively address this problem: (1) measuring prompt ambiguity by training an LLM-as-a-judge, (2) diagnosing the levels and effects of ambiguity (dimensions), and (3) generating clarified prompts automatically. This prevents users from acting on misaligned responses and also shortens the progression from fully crafted prompts to well-calibrated clarify-only prompts. Experiments demonstrate that ambiguous prompts involve an average waste of 275.8 tokens per interaction, and our framework consumes only 139.4 tokens, achieving a 136.4 token efficiency saving per prompt. We make available an annotated benchmark of ambiguous prompts to facilitate future research in prompt engineering. Summary: Our method presents a lightweight yet effective approach to mitigating failure cases in LLM applications without relying on model fine-tuning.

#1. Introduction

Although LLMs can perform miracles, they fail when a user prompt is vague or not sufficiently specified. Standard LLMs generally continue to produce responses when given wildly ambiguous prompts, siphoning computation into responses that are not aligned with user intent. This inefficiency leads to the natural question of why LLMs continue to generate outputs for vague prompts rather than signal uncertainty or prompt clarification. This paper fills this crucial gap with a proactive framework for timely disambiguation. Our approach systematically quantifies prompt clarity prior to inference, provides an ambiguity score, discovers ambiguity dimensions, and offers ambiguity-aware automatic prompt clarification. By doing so, we hope to enhance the efficiency and reliability of LLM-powered pipelines, as well as empower users with actionable, instantaneous feedback on prompt quality.

A major advantage of our approach lies in its architecture-agnostic design: it does not rely on any details of a specific LLM or even on any

individual architecture or training set. This allows it to be enabled in the large variety of currently and future available LLMs. In this paper, we describe our framework and experimental validation, showing its capacity in reducing computational wastage and enhancing usability for users.

#2. Objectives

The objectives of the study are as follows: 1.**Anticipatory Prompt

Ambiguity Detection:** Design an LLM scoring scheme to anticipate prompt

ambiguity before inference. 2.**Reasoning about Sources of

Ambiguities:** Discover and summarize unknown parameters as concepts of

ambiguity dimensions. 3.**Automated Task Formulation:** Automatically

derive clarified prompts by resolving the measured ambiguity dimensions.

4.**Computational Efficiency:** Solve the misalignment problem by

upstream LLM inference to be more efficient in terms of CPU usage and

have fewer retries. 5.**Learning Feedback:** Supply users of differing

technical-experience levels with clear, real-time feedback to help them

improve their prompting skills and user abilities.

#3. Related Work

Recently, the problem of prompt ambiguity and the induced impact on large language models (LLMs) has been studied extensively. The research community has already contributed to some frameworks that are mostly based on the reactive clarification approach to address this issue.

ClarifvGPT\

ClarifyGPT suggests a reasoning-based pipeline to identify and eliminate ambiguity in the tasks of code generation. Prompt ambiguity is detected with the code interpretation sampling step, for which one needs training of various code interpretations and examination of output inconsistencies, which is (a) computationally expensive and (b) specific to coding tasks. When identifying such differences, ClarifyGPT posits targeted clarifying questions according to the presumed reasons behind the behavioral disparities observed between the samples of generated code. Although successful, ClarifyGPT is by nature reactive and restricted to binary ambiguity detection. On the other hand, our method tries to predict ambiguity in advance of any form of reasoning with a graded ambiguity score and a structured identification of the sources of

ambiguity and their level of harm. Moreover, our approach eliminates the use of expensive execution-based consistency checks and thus can be applied at scale for various other non-programming domains.

CLam\

The CLam framework proposes a refined LLM-based model that selectively inquires clarifying questions for ambiguous user queries. CLAM uses a binary classifier to decide whether a query is ambiguous or not, where the classifier is based on fine-tuning a language model on the AmbigQA dataset for question generation. However, CLAM's reliance on labeled supervision constrains its portability and also generates clarifying questions one by one rather than offering the whole diagnostic picture of the problem that still persists.

ECLAIR\

The scope of ambiguity handling in the domain of enterprise dialogue systems has been extended by ECLAIR, which makes use of static classifiers, heuristics, and domain-specific agents for identifying and resolving ambiguities in user inputs. Based on dialogue state tracking and predefined patterns, its architecture generates clarification question templates, which are quite dependent on enterprise ontologies. Strong in its domain, ECLAIR is caught in domain lock-in and is stoic to generalization. It is also unspecified in a measurable degree of ambiguity. By contrast, our approach adds a measurable ambiguity score (between, you decide) that can be used for standard comparison across a range of domains. We further enable LLMs as self-assessors (LLM-as-a-judge), going beyond static classification into the realm of dynamic, zero-shot ambiguity detection and resolution.

Clarify When Necessary\

This first task-independent approach addresses whether clarification is needed, guided by uncertainty estimation using the INTENT-SIM metric. By generating simulated user intents and grouping them together, ambiguity is measured via entropy. However, this approach is still reactive, making a decision only after output sampling and using a binary decision threshold. In contrast, our proactive approach assesses prompt ambiguity in a direct manner and provides detailed feedback through dimension-specific analysis. Moreover, in contrast to merely requesting prompt clarification, our approach can rewrite unclear prompts into more effective ones that help users write better prompts. What's more, it

serves as a tutorial on prompt engineering as well.

PRewrite\

PRewrite employs reinforcement learning (RL) to produce prompt rewriting automatically to facilitate the performance of downstream tasks. An RL agent trained using Proximal Policy Optimization generates new prompts using preparatory claims that are more likely to improve any task-specific metric (such as accuracy or F1 score).

3.1 A Summary of New Contributions by Our Work

Our model moves the state-of-the-art in several aspects:

1.**Proactive Ambiguity Detection:**\

Unlike reactive approaches like ClarifyGPT and INTENT-SIM that only react after the facts, our process is here to approve prompts prior to any inference occurring. Surely we could detect the ambiguity problems a priori, and we don't just save resources and time, but we could even be more ecological as we don't generate unnecessary responses that may waste resources. In practice, it's a fantastic scenario with huge time and resource savings for all.

2.**Dimension-Based Analysis:**\

Most current systems will let you know in binary form whether something is wrong or not. Our technique is, however, more informative in that we provide the user with an enumeration of the types of ambiguity -- whether it comes from an unclear task definition, missing input specifications, underdetermined desired outputs, or inadequate context. This comprehensive explanation can help users diagnose issues with their prompts.

3. **Educational Component: **\

Other systems are purely reactive to emerging problems, but our approach is proactive and empowering in that it helps users get their prompts right the first time around. We not only tell them how to fix certain issues, but we give them choices as they make headway utilizing the prompting established in the examples.

4.**Domain Flexibility:**\

Both ECLAIR and ClarifyGPT are restricted and designed to tackle

only business-related problems and programming domains, respectively. Our approach does not depend on any specific model architecture and any special training data to be implemented, and can, therefore, be used for any application of LLM.

5.**Efficiency:**\

Also, unlike CLAM, which presents one clarification question at a time, we bring everything together in one place---a table---which is far easier to work with. As a result, the whole situation is quickly understood by users, which makes the entire process time-saving.

In summary, our work is a departure because it provides an active, quantifiable, educational framework for the resolution of ambiguity.

Leveraging the LLM's self-assessment capability to guide users for self-reviewing and giving clear feedback on the ambiguous parts, we hope to improve the overall quality of LLM outputs and enhance the prompting skills of users.

#4. Methodology

Our framework composes a frontend to preprocess user requests and checks their validity before forwarding them to the interested LLM. This component uses a formatted 'meta-prompt' to an analyzer LLM providing an extended optionality test and prompt clarification.

4.1 System Architecture

The system is illustrated as a three-stage pipeline, which is displayed to the user (see Figure 1), known in the following order:

Input Prompt \rightarrow Ambiguity Scorer \rightarrow Dimension Detector \rightarrow Clarification Generator \rightarrow Final Prompt

Stage 1: Ambiguity Scoring A score (ranging from 0 to 100) reflects how ambiguous a prompt is and hence indicates the quality of a prompt.

Stage 2: Dimension Breakdown The following is a table that the framework lists: - Dimension name - Level of impact (High, Medium, Low) - A couple of hypotheses LLM could adopt

Stage 3: Clarified Prompt Suggestion Based on the determined ambiguity facets, the system automatically selects a single, completely resolved (clarified) prompt that specifies the resolved doubts. This, after all is said and done, gives the users a place for editing for further refining.

! [System Architecture](images/Figure1.jpeg)

4.2 Core Implementation

4.2.1 Frontend Service Integration With Puter.js

A key novelty of our approach is the use of Puter.js, a serverless
JavaScript SDK allowing inference from a browser directly, without
server-side authentication, cloud storage, and database access, as well
as performing powerful LLM inference. For a developer, that opens the
doorways to lots of things including multi-model AI chat, user
authentication, and KeePassCloud storage---from the client-side
scripting only by adding a single `<script>` tag.

Advantages of Puter.js Integration:

- **True Serverless: ** All prompt analysis, ambiguity scoring, clarifications, and inferencing occur in your browser---no backend to deploy or manage.

- **Unified API for Cloud & AI:** Storage, user-session management, LLM calls all have similar, user-friendly APIs.

- **Bet on User Privacy & Security:** Data is kept at the client; it will only be shared/stored via cloud APIs if the customer agrees.

- **Rapid Prototyping:** No hassle setup means you can easily create demos and scale all the way to production.

Example usage:

``` html

<script src='https://js.puter.com/v2/'></script>

<script>

puter.ai.chat('Hello Al! This is Adnan & Dr. Aiman Abu Samra.');

</script>

•••

This seamless integration allows any LLM-based workflow to be executed natively in the browser and massively increases the accessibility of the technology to developers and end-users.

### 4.2.2 Modular LLM Processing: 'generateLLMResponses' and 'analyzeLLMResponses'

The heart of both our systems is two modular JavaScript functions:

#### `generateLLMResponses`

Manages sending prompts to the chosen LLM (GPT-4 or otherwise), which controls model selection and API requests through Puter.js. A prompt generator is employed to express both baseline (potentially ambiguous) prompts and disambiguated prompts output by the analyzer. - It accepts user/system prompts and the model choice. - Invokes the LLM API via Puter.js. - It returns the raw model output to the UI.

#### `analyzeLLMResponses`

Structured core analysis is triggered by wrapping the user prompt in a highly structured meta-prompt sent to the analysis LLM. Within this meta-prompt, the LLM is directed to: - Assign a score reflecting the degree of uncertainty (range between 0 and 100) - Record all uncertain dimensions, their labels, impact levels, as well as two (or more) relative assumptions. - Create clear rewrites of the prompt.

The LLM returns tabulated results in a strict JSON schema, facilitating predictable UI rendering and post-processing tasks. The meta-prompt template we used is as follows:

`Where is the prompt ambiguous, given the prompt text?'\${promptText}'`
Return in JSON:

- ambiguityRating (0-100)
- analysisTable: [{ impactLevel, dimensionName, possibleAssumptions (two assumed values) }]
- suggestedClarifiedPrompt: clarified prompt assumption for the missing parameters`

\*\*Typical workflow:\*\*1. User submits a prompt. 2. `analyzeLLMResponses` measures ambiguity and provides analysis and feedback. 3. User chooses a refined prompt and changes it if necessary. 4. `generateLLMResponses`

fetches the LLM response for the clarified prompt.

### 4.2.3 Interactive Chat Interface

The frontend is a feature-rich, browser-based chat interface fluent in the language of JavaScript and written in Puter.js for all AI operations.

\*\*Key features:\*\*

- \*\*Model Selection: \*\* Users select from the LLMs in the supported list (default: GPT-4).

- \*\*Message Processing in Real Time:\*\* Processing messages in real time from the user and the AI, which can be copied or exported.

- \*\*Multimodal Support:\*\* It can take images uploaded or pasted by users as content for analysis, together with text prompts.

- \*\*Prompt Clarification Modal: \*\* The button 'Clarify' and its three sub-checkboxes are intended to immediately trigger the detect process of ambiguities and, thus, instead of only obtaining scores and ambiguity dimensions, also obtain proposed rewrites in a modal window with a single click to insert.

- \*\*Export/Copy:\*\* The whole chat can be saved (Word supported) or copied in a formatted manner.

- \*\*User Experience Improvements:\*\* Feedback messages, auto-size inputs, superior error handling.

\*\*Workflow:\*\* 1. User inputs a prompt and/or image. 2. The prompt is sent to the LLM through Puter.js; the AI reply is displayed. 3.

'Clarify' examines ambiguity and shows structured feedback and advice.

4. User can export and copy chat history.

\*For a live demonstration, visit:

[`amuammer/ai`](https://amuammer.github.io/ai).\*

### 4.3 Dataset Preparation and Processing Source Code

In order to systematically test and compare our ambiguity analysis framework, we implemented a set of scripts to create, analyze, and combine LLM responses. All preparation and pre-processing of the datasets were conducted in fully transparent client-side and Node.js

code, providing a framework that reproduces and can be extended.

- \*\*Key steps and scripts:\*\*
- \*\*Prompt and Baseline Generation\*\*
- 1. collect-generatorLLM-responses.html: Uses Puter.js itself in the browser to send each ambiguous prompt to the LLM and save the raw baseline response.
- \*\*Ambiguity Analysis Collection\*\*
- '2. collect-analyzerLLM-responses.html': Uses Puter.js in the browser to serve prompts over the ambiguity analysis meta-prompt and aggregate structured analyzer responses.
- \*\*CSV Export and Deduplication\*\*

'back-queried-LLM-to-csv-ambiguous.js': Node.js script that generates unique (field, prompt) pairs from a dataset and saves them into a CSV to be processed.

- -\*\*Deduplication, Grouping, and Indexing\*\*`ordering-dataset.js`:
  Node.js script that strips duplicates, gives entries a global
  identifier & sequential index in that stream, and groups entries by
  field for further analysis.
- \*\*Changing Generator and Analyzer Outputs\*\*
- '3. mergeLLMResponses.js': Node.js script that combines the baseline LLM predictions with the accompanying ambiguity analyses, matching by prompt and standardizing the indexing across the dataset.

Each script is documented and configurable for reusability with other jobs or LLMs. Output data sets are utilized in the experimental sections (refer to Section 5). Full code and reproducibility details are available at [our repository](https://github.com/amuammer/ai).

## # 5. Experiments

In order to validate the effectiveness of our pre-processing, an experimental protocol in terms of token-level cost analysis is developed.

#### ## 5.1 Dataset and Procedure

We composed a benchmark set of extremely ambiguous prompts across various tasks as follows: - Business Communication (15) - Code

Generation (15) - Data Analysis (15) - Education (15) - Technical
Writing (15) - Human Resources (15) - General Question Answering (15)

! [ambiguityRatingPerField](images/Figure3.jpeg)

For each ambiguous command, we performed the following: 1.\*\*Simulate a Wasted Cycle:\*\*\

The `original\_prompt` was submitted to a generation LLM (GPT-4) to get a `baseline\_response` and simulate the response of an initial, possibly unhelpful answer.

#### 2.\*\*User Clarification:\*\*\

A human expert authored a `followup\_prompt`---an even more explicit reformulation which, when given as input, should enable the user to provide a valid response.

#### 3.\*\*System Analysis:\*\*\

The same `original\_prompt` was processed by our model LLM, resulting in a structured `analyzer\_response` with ambiguity score, diagnosis, and suggestions for the prompt.

## 4.\*\*Token Counting:\*\*\

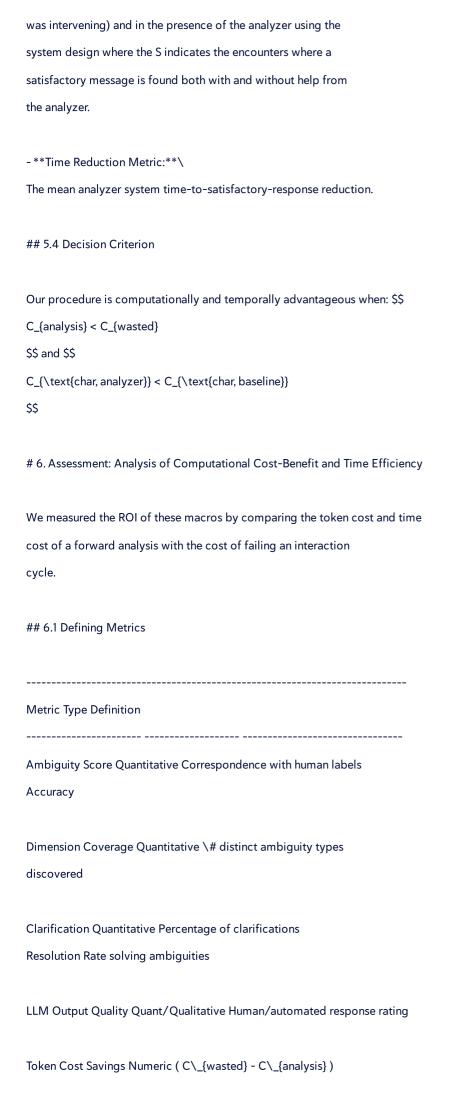
For each prompt, we reported token counts for:

- `T\_original\_prompt`
- `T\_baseline\_response`
- `T\_followup\_prompt`
- `T\_analyzer\_response`
- The hard-coded token count of the `T\_analyzer\_prompt` template is (31).

## ## 5.2 Evaluation Procedure

The authors have developed an automatic script that performs the following steps for each prompt in the dataset: 1. Send the 'ambiguous\_prompt' to a generation LLM Model (e.g., GPT-4), and a 'baseline\_response' is returned. 2. Pass 'ambiguous\_prompt' to the Analysis LLM Model and get 'ambiguityRating', 'analysisTable', and 'suggestedClarifiedPrompt'. 3. Transfer the 'suggestionClarifiedPrompt'

```
again to the generation LLM Model, and a 'clarified_response' is
generated in return.
5.3 Quantifying Reduction in Computation Waste and ROI
We introduce a token-driven cost-benefit model to evaluate the
effectiveness of our system.
Cost Metrics:
1.**Computational Cost Savings:**
- **The Cost of Preventive Analysis (`C_analysis`):**\
The complexity of finding an ambiguity includes:
- Tokens used in the analyzer meta-prompt template
('T_analyzer_prompt')
- Tokens for the original user prompt ('T_original_user')
- Tokens in the response from the analyzer (`T_analyzer_response`)
$$
C_{analysis} = T_{analyzer\prompt} + T_{original\prompt} + T_{analyzer\prompt}
$$
- **Cost of a Wasted Generation Cycle (`C_{wasted}`):**\
The penalty of a failed transaction due to ambiguity, where:
- Tokens in the source prompt
- Tokens in the non-effective baseline response
- Tokens in user follow-up clarification $$
C_{wasted} = T_{original_prompt} + T_{baseline_response} + T_{followup_prompt}
$$
- **Computational Cost Savings Metric:**\
Mean of the difference between `C_{wasted}` and `C_{analysis}` for
all prompts. $$
\text{Saving in Computational Cost} = C_{wasted} - C_{analysis}
$$
2.**Time-to-Satisfactory Response:**
-**Definition:**\
The time taken from the ambiguous prompt to the first useful LLM
response.
- **Measurement:**\
For each message, we log the time, in the baseline (no analyzer
```



Time-to-Resolution Discrete Number of words between prompt and correct answer User Satisfaction Qualitative User surveys, ratings Adoption Rate Quantitative \% users who adopt clarified prompts Failure Rate Quantitative System robustness, error rate The metrics above are all things we could go with, but we select: - \*\*Computational Cost Savings: \*\*\ The trivial saving in tokens associated with the application of the analyzer system as stated above. - \*\*Time-to-Satisfactory-Response: \*\*\ The amount of time saved between submitting the prompt and receiving an acceptable answer via the analyzer. ## 6.2 Results Over the complete set of prompts in the benchmark, the average costs were:  $C_{wasted} = 28,956$  $C_{analysis}$  sum = 14,641 Total entries = 105  $avg_{char}, baseline\_response$  = 1241.5 char  $avg_{char, analyzer\_response} = 179 char$ Metric Avg Value \*\*Avg. C\\_{wasted} (Fail, tokens)\*\* 275.8 tokens \*\*Avg. C\\_{analysis} (Prevention, tokens)\*\*139.4 tokens \*\*Net Computational Savings per Prompt\*\* 136.4 tokens \*\*Avg. Reading Baseline Response (Failure)\*\* 82.77 sec \*\*Avg. Reading Analyzer Response\*\* 11.93 seconds \*\*Net Time Spent in Reading Effort\*\* 70.83 sec

! [tokenAndTimeEfficiency](images/Figure4.jpeg)

The reward of engaging the analyzer LLM is, on average, half the token cost of an unsuccessful interaction. Preventative analysis using 139.4 tokens prevents 275.8 tokens of waste on average for each ambiguous prompt---a 49.5% reduction in computational cost per interaction. The analyzer eliminates much of the original thinking and typing time of writing a follow-up. But it does not remove user effort entirely---it offloads some of that effort to creating an edit clarification.

Without Analyzer:

User Prompt -> User reads reply  $\rightarrow$  recognizes failure  $\rightarrow$  writes up follow-up prompt  $\rightarrow$  hits send

With Analyzer:

User Prompt -> Special malware protect -> Analyzer creates cleaned prompt - User reviews/edits -> click sends

Reading Rates Assumptions (According to Research) Average adult reading speed:

200-250 words per minute 1000-1250 cpm (for approximately 5 characters per word)

So:

1000 characters 60 seconds = 16.7 characters/second

\*\*Let's presume a modest speed of reading:\*\*

15 characters per second

\*\*Average total baseline response characters to find ambiguity =  $\$ 1241.5 char and analyzer response =  $\$ 179 char\*\*

When analyzing failures, the users spend on average around 83 seconds reading the ambiguous baseline responses, while around 12 seconds for the structured analyzer outputs. This results in a 71% reduction in reading time, the potential for increased comprehension, as well as a decrease in cognitive load during the disambiguation process.

! [tokenAndTimePerField](images/Figure5.jpeg)

#### #7. Discussion and Future Work

Our results demonstrate that ambiguity analysis integration is beneficial in both computational and practical aspects when applied to LLM pipelines:

- High requests resolution ambiguity requires preprocessing a fast and cost-efficient method that minimizes trial and error for the user after solving.
- The approach is scalable where there are metered/limited LLMs (e.g., API services).
- \*\*Future Directions:\*\*
- Adaptive model design allowing for localization of the probability of ambiguity prior to analysis start-up.
- Measure the non-computational benefits like user satisfaction and task success.
- Broadening the application domain to multi-turn dialogues where ambiguity problems may accumulate.

## #8. Conclusion

In particular, our system can automatically score how ambiguous a prompt is and identify and generate clarified versions of prompts, thereby significantly reducing the waste of computation in LLM-based pipelines.

## **SCAN SETTINGS**

These features were chosen to create this report

| Omitsettings       | Private Cloud Hubs |  |
|--------------------|--------------------|--|
| References:        | Off                |  |
| Quotes:            | Off                |  |
| Citations:         | Off                |  |
| Titles:            | Off                |  |
| HTML Templates:    | Off                |  |
| Table Of Contents: | Off                |  |
| Code Comments:     | Off                |  |

#### **Plagiarism Detection Settings**

| Security Measures       |     |
|-------------------------|-----|
| Safe Search:            | Off |
| Hide Sensitive Data:    | Off |
| Character Manipulation: | Off |
| Similarity Level        |     |
| Identical:              | On  |
| Minor Changes:          | On  |
| Al Source Match:        | Off |

| Results Calibration |                |
|---------------------|----------------|
| Focused Results     | Sensitivity: 3 |