Style Guide: Writing for AI

Role: Technical Writer / Content Strategist
Deliverables: Style guide, LLM optimization checklist, anti-pattern reference
Audience: Technical writers, content developers, documentation engineers

Overview

With AI chatbots and RAG systems becoming an integral part of content development, it is important to provide clear and explicit instructions in prompts and to structure documents for better consumption. This goes beyond the baseline of good grammar and proper syntax. These guidelines are designed to strip away the ambiguity that can trip up LLMs. Well-crafted content with specific language is more likely to ensure the LLM gets the technical facts right instead of hallucinating a better-sounding answer.

Guidelines

1. Use Descriptive, Searchable Headers

Rule: Every heading (H1-H4) must be a self-contained summary that includes searchable keywords.

Rationale: RAG systems retrieve content in chunks. Headers must provide enough context to stand alone.

> Bad: ## Setup
>
> LLM-Optimized: ## Set Up the Python SDK for API Authentication

2. Eliminate Directional References

Rule: Do not reference position-based context (e.g., "above," "below," "on the right").

Rationale: LLMs process text linearly, not visually. Directional references break when content is chunked.

> Bad: "See the table below."
>
> LLM-Optimized: "Refer to the Authentication Parameters table."

3. Enforce Strict Entity Consistency

Rule: Use one canonical name per concept. Do not use synonyms or variations.

Rationale: Inconsistent naming weakens semantic linking and causes entity fragmentation.

Bad: "The Portal," "Dashboard," and "Console"

LLM-Optimized: "Management Console" (used consistently)

4. Define Entities at First Use

Rule: Explicitly define each key entity when it first appears.

Rationale: Retrieved chunks may not include earlier explanations.

Example: "The Management Console is the web interface used to configure API settings."

5. Prefer Simple, Parseable Data Structures

Rule: Use flat tables or key-value pairs. Avoid merged cells, nested tables, or complex HTML.

Rationale: Complex structures often break during parsing and ingestion.

Bad: Nested or merged HTML tables

LLM-Optimized: API_KEY: Your 32-character API key | REGION: Deployment region (e.g., us-west-1)

6. Use the Action-Object Procedure Format

Rule: Write steps using an imperative verb followed by a clear object.

Rationale: Improves readability and aligns with task-oriented interpretation.

Bad: "Once you're on the settings page, use the button to generate a key."

LLM-Optimized: 1. Navigate to the API Settings page. 2. Click the Generate Key button.

7. Include Explicit Input and Output Examples

Rule: Provide at least one concrete example for every feature or workflow.

Rationale: Examples anchor meaning and reduce hallucination.

Example: Request: POST /v1/keys | Response: { "api_key": "abc123" }

8. Define Negative Constraints and Limitations

Rule: Clearly state what is not supported or cannot be done.

Rationale: Prevents LLMs from inferring nonexistent capabilities.

Example: "The SDK does not support asynchronous calls in Python 3.7."

9. Document Failure Modes and Errors

Rule: Include expected errors, causes, and resolutions.

Rationale: Helps both users and LLMs diagnose issues correctly.

Example: "Returns 401 Unauthorized if the API Key is invalid or expired."

10. Ensure Self-Contained Context

Rule: Each section must be understandable without external context.

Rationale: Retrieved chunks are often isolated from the full document.

Bad: "Use the SDK client to send requests."

LLM-Optimized: "Use the Python SDK client (Client) to send authenticated API requests."

11. Avoid Pronoun Ambiguity

Rule: Replace ambiguous pronouns ("it," "this," "they") with explicit nouns.

Rationale: Reduces ambiguity in reference resolution.

Bad: "If it fails, retry it."

LLM-Optimized: "If the API request fails, retry the API request."

12. Include Metadata Headers

Rule: Add a structured metadata block at the top of each document.

Rationale: Improves indexing, filtering, and retrieval accuracy.

Example: title: Webhook Integration | audience: backend_developers | topic_type: tutorial | keywords: webhook, events, API

13. Specify Versioning and Time Context

Rule: Include version numbers and deprecation timelines where applicable.

Rationale: Prevents outdated or incompatible guidance.

Example: "Available in Python SDK v2.3+" | "Deprecated as of March 2025"

14. Use Consistent Markdown Formatting

Rule: Follow standard Markdown conventions for structure and code.

Rationale: Ensures reliable parsing across ingestion pipelines.

Guidelines: Use triple backticks for code blocks. Use ordered lists for procedures. Avoid nonstandard formatting.

15. Optimize for Semantic Density (Not Just Brevity)

Rule: Remove unnecessary filler while preserving clarity and keywords.

Rationale: LLMs depend on meaningful tokens, not just fewer tokens.

Bad: "In this section, we will explore how you can begin to..."

LLM-Optimized: "This section explains how to authenticate using OAuth 2.0."

16. Use Standardized Section Templates

Rule: Follow consistent templates for common content types.

Rationale: Predictable structure improves retrieval and comprehension.

Example (Tutorial Template): ## Goal | ## Prerequisites | ## Steps | ## Expected Result | ## Troubleshooting

LLM Optimization Checklist

Use this checklist during writing and code reviews:

- □ Do all headers include searchable, descriptive keywords?
- □ Can each section stand alone without external context?
- □ Are all entities clearly defined and consistently named?
- □ Are examples included for each feature or workflow?
- □ Are limitations and unsupported cases documented?
- □ Are common errors and failure modes explained?
- □ Is formatting simple, consistent, and machine-parseable?
- □ Are versioning and time-sensitive details included?

Common Anti-Patterns

Avoid these:

- Vague headers ("Overview," "Details")
- Synonym swapping for the same concept
- Missing examples
- Overly complex tables
- Directional references ("see below")
- Ambiguous pronouns
- Long, low-information introductions