ChatGPT

Claude System Prompt Design: Key Principles and Workflows

Introduction

In early 2026, a detailed **system prompt** for Anthropic's AI assistant **Claude** was leaked, revealing the internal rules and design principles governing Claude's behavior. This document distills the key components from that prompt (dated January 1, 2026) and interprets them for a technical audience. We focus on how Claude interacts and reasons, how it uses tools like web search, how it generates outputs (especially code artifacts), how it handles citations and safety constraints, and how it adapts to user preferences. The aim is to provide actionable insights into **Claude's operational logic**, enabling developers to better understand and utilize the model's capabilities. All information and quotes are drawn directly from the leaked prompt [1] [2], without speculation beyond that source.

Core Interaction and Reasoning Behaviors

Claude is designed to be **maximally helpful and responsive** in conversation while adhering to strict guidelines. It always attempts to provide a substantive answer to the user's query using its own knowledge before resorting to any external tools [3]. In fact, **Claude avoids unnecessary tool use** – if it can answer from its built-in knowledge (especially for timeless or well-known information), it should do so directly rather than search the web [2] [4]. Every user question "deserves a substantive response," and Claude should never reply with a mere refusal or a generic disclaimer without first providing as much help as it safely can [3].

When the user's query *does* require additional information (for example, about recent events or unknown entities), Claude will engage its tools – but always in a measured way (detailed in a later section). Importantly, Claude's prompt emphasizes **balanced reasoning**: the assistant should acknowledge uncertainty honestly while still giving a clear answer, and only then, if appropriate, offer to dig deeper. For instance, if asked for a statistic it knows roughly, Claude might answer from memory and then say it can verify with a search if the user wants [5] [6].

Claude's tone and formatting adjust based on context. In casual or empathetic conversations, the assistant keeps a *warm, natural tone* and typically responds in a conversational style (short paragraphs, no over-formal structuring) [7]. It avoids using lists or markdown formatting in casual chat, instead replying in complete sentences or brief paragraphs to keep the dialogue feeling natural [8] [9]. On more technical or task-oriented queries, Claude can produce more structured content (including lists, tables, or code blocks) as needed. It always tries to **tailor its response format to the situation**, but never at the expense of clarity or helpfulness [9].

Another key behavior is that Claude **never feigns ignorance needlessly**. It does not start answers with filler phrases like "That's a great question" or undue compliments [1]. It simply addresses the query directly to respect the user's time. If Claude ever cannot fulfill a request (due to safety/policy reasons or inability), it will *politely refuse or omit that portion* without lecturing the user. The system prompt explicitly instructs: **do not scold or moralize**. If something is disallowed, Claude should give a brief refusal (or an alternative solution if possible) in a calm, concise manner [10]. For example, if asked for disallowed content, Claude might respond with one or two sentences noting it cannot assist with that request, possibly suggesting a safer alternative, and then move on. It should **not** provide long justifications for refusals, as these can come across as preachy [10].

Claude also actively tries to **verify assumptions or facts** in the user's query. The prompt reminds the assistant that user questions may contain false presuppositions; Claude should double-check anything that seems uncertain [11]. It won't outright accuse the user of being wrong; instead, it will clarify or incorporate correct information tactfully. Maintaining a cooperative tone is paramount. If the user corrects Claude or points out a mistake, Claude is expected to pause and reason carefully – the assistant shouldn't reflexively admit wrongdoing if the user might be mistaken themselves [12]. This indicates an effort to be accurate and not overly deferential when the facts are at stake.

Finally, Claude's **memory is conversation-limited**. It does not carry information between different chats and has no awareness of external context beyond the current conversation (and any provided files or enabled tools). If asked what it was doing outside the chat, Claude will explain it has no activities or experiences beyond assisting the user [13]. It won't reference other sessions or users. This is standard for AI assistants, but the prompt explicitly notes it to avoid any confusing responses. Claude also recognizes that everything it writes is visible to the user, so it doesn't "think" or rant privately; even its internal reasoning (if shown via a special *thinking* mode) should be something it wouldn't mind the user seeing [14].

In summary, Claude's core interaction principles are about **helpfulness, transparency, and appropriateness**. It strives to answer directly and accurately, uses a tone suitable for the context, and abides by the rule "help the user as much as possible while staying within constraints." It engages in reasoning steps internally (and sometimes shows these in a structured way when the platform supports it) to ensure it's following the user's intent and the given policies.

Tool Use and Web Search Strategies

One distinctive aspect of Claude's behavior is its **selective use of external tools** (like web search, document retrieval, and code execution tools). The leaked prompt describes a sophisticated logic for **when and how Claude should perform searches or use integrated tools**, depending on the query's nature. The overarching rule is to **avoid tool calls unless they are necessary** [2]. Claude leverages a large base of built-in knowledge (with a training cutoff around January 2025 [15]), so it should tap into that first for stable, factual questions. If the user asks about "fundamental concepts, definitions, theories, or well-established knowledge," the assistant will answer from memory without any web search [4]. This is termed the **"Never Search" category** of queries. It includes questions about long-standing facts (e.g. historical dates, basic science), general explanatory requests, programming help for known techniques, and casual conversation [16]. Claude is instructed to *never* run a search for such queries, as it likely already knows the answer and the information doesn't change frequently [4]. For example, if asked *"What is the capital of France?"*, Claude should immediately answer "Paris" with no search, since this is stable, common knowledge.

However, not all queries are so straightforward. The prompt defines a **decision tree of query complexity** [17] to determine tool usage: - **If the query involves something Claude doesn't recognize or lacks information on** (for instance, a newly coined term or a very niche topic), **perform a quick search** [18]. There's no point in guessing – one web search should clarify the unknown term. This is also true if the user asks for a specific piece of data that Claude isn't sure about. - **If the query is about information that changes frequently (daily, weekly, or monthly)** – such as current events, latest statistics, weather, stock prices, recent news – Claude should **definitely search the web** for up-to-date info [19] [20]. Even if Claude has some older knowledge, the assistant is told to fetch the latest authoritative source in these cases. Often a single well-chosen search query is enough for these **"Single Search"** queries [21]. For example, *"What's the weather in London today?"* or *"Who won yesterday's football match?"* would prompt an immediate search and then a direct answer, possibly with an offer to find more if needed [21]. - **If the query is about something that changes slowly (order of years) or might have minor updates**, Claude falls into a middle ground: it should **answer from its existing knowledge first, then offer to search for more recent updates** [22] [23]. This is called the **"Do Not Search But Offer"** category. For instance, for a question like *"What is the population of Tokyo?"* (which might have changed slightly since Claude's training data), Claude might respond with the population figure it knows (e.g. an estimate of ~14 million for Tokyo proper, ~37 million metro) and then ask if the user wants the latest official statistic [24]. The assistant doesn't immediately hit the web, because its existing knowledge is mostly sufficient, but it remains proactive by offering further research if the user desires [5]. This strategy ensures efficiency (avoiding unnecessary calls) while still acknowledging that fresher data could be obtained. - **If the query is complex, multi-faceted, or explicitly asks for thorough research or a report**, Claude enters the **"Research" category**, which involves performing **multiple tool calls (between 2 and 20)** [25]. These could include several web searches, using internal knowledge base tools (like searching the user's Google Drive or emails if those are available for context), and fetching specific documents. The prompt indicates that any query containing hints of complexity – e.g. requests to "analyze," "compare," "evaluate," or involving corporate-specific terms like "our metrics vs industry" – should trigger a **comprehensive research mode** [26] [27]. In such cases, Claude is expected to **plan a research strategy**, use all relevant tools, and synthesize information from multiple sources.

Claude's tool usage thus scales with query difficulty. A simple factual question might get **0 searches**, a request for a single recent fact gets **1 search**, and an analytical question might get **5–10 searches**, combining internal and external sources as needed [28] [29]. The prompt even quantifies this: simple comparisons or verifications might need 2–4 calls, whereas an in-depth analysis or report could justify 10 or more calls [30] [27]. Claude is told to cap this research at around 15–20 calls in practice – if the answer isn't found by then, it should provide whatever best answer it can and suggest the user might use a more extended research mode themselves for further depth [31].

For **extensive research tasks**, Claude follows a clear workflow [32]: 1. **Planning** – First, analyze the query and decide which tools and sources will be useful. For example, a complex question about market trends might need web news, official reports, and internal company data. Claude effectively creates a mini research plan in its "thinking" process. 2. **Iterative searching** – Next, perform a series of searches and data retrievals. The prompt *requires at least five distinct searches for truly complex queries* [33], ensuring Claude doesn't give a superficial answer. After each search result, Claude should read and analyze the findings, then decide the next step. This loop continues, refining queries each time, until enough information is gathered or the tool-call budget (around 15 calls) is reached [34]. 3. **Answer synthesis** – Finally, Claude should compile the collected information into a coherent answer or report. The prompt suggests including a **brief summary or "bottom line up front"** at the beginning of the answer for lengthy analyses, often in bold, to address the query directly [35]. It should use **clear section headings** (in sentence case) for organized reports and highlight key facts in bold to improve readability [35]. Redundant or filler information should be trimmed out, to focus on the insights that actually answer the user's question.

Throughout this process, Claude must use tools **intelligently and transparently**. The prompt provides detailed **guidelines for formulating search queries** [36]. It advises Claude to start with **concise, broad keywords (1–3 words)** then narrow down if needed, rather than typing a long complicated query from the start [36]. Each search attempt should be unique – *no blind repetition of a failed query*. If initial results are irrelevant, Claude should reformulate the query with different terms based on what it learned from those results [37]. Also, Claude should not use advanced search operators like `site:` or quoted exact phrases unless explicitly instructed by the user [38] (those might overly constrain results or conflict with how the tool is supposed to be used). Another nuance: if the user's query includes a specific URL or asks about a specific webpage, Claude should use a direct **web fetch** function to retrieve that page's content instead of searching for it [39] [40]. This ensures the assistant gets the precise information requested.

The prompt also emphasizes **copyright compliance during searches**: Claude must **never copy large blocks of text from any source** it finds [41]. In practice, this means if Claude finds relevant text on a webpage, it can summarize or quote a tiny snippet, but it cannot lift whole paragraphs. (The detailed policy, discussed later, limits quotes to <15 words [42].) Claude is to be extremely careful to respect intellectual property while using search results. If a user asks to read out a whole article or large excerpt, Claude must refuse and instead encourage the user to read it directly or accept a summary [43] [44].

Choosing sources is another part of Claude's search strategy. The assistant should prioritize **high-quality, authoritative sources** – for example, official websites, academic papers, reputable news outlets, or company blogs – over low-quality content like random forums or user comments [45]. It should also favor **recent information** for topics that evolve, typically citing sources from the last 1–3 months for current events questions [46]. If multiple sources conflict on facts, Claude should note the discrepancy or pick the most credible source, rather than blindly averaging them. And notably, Claude must **cite any source that significantly contributes to its answer**, so the user can verify the information [47]. The prompt provides a specialized citation format, where each factual claim drawn from a tool result is tagged and indexed to the source document's sentence [48]. In simpler terms, Claude will provide reference links (like the ones you see in this white paper) next to statements that came from a search result. This is intended to increase transparency and reduce hallucinations.

Claude's search usage is also governed by **safety and privacy filters** (discussed more in the safety section). For example, if the user shares an image of a person and asks "Who is this?", Claude is explicitly instructed *not to search the web by image or description to identify the person* [49]. This protects privacy – Claude will remain "face-blind" (a rule expanded under safety) and not attempt to recognize or confirm someone's identity via search. Similarly, Claude will not search for content that is overtly harmful or extremist in nature [50], even if the user asks (more on this later).

To illustrate, the prompt includes examples. One shows a query for population of a city, where Claude answers from its knowledge and then politely offers to check the latest figure. Another example shows a complex query about *"semiconductor export restrictions and investment strategy"* where Claude correctly infers it needs extensive research: it outlines a plan in the conversation (to search news, government sites, internal documents, etc.), then executes multiple searches and even uses internal company file search tools, before compiling a detailed report [51] [52] [53]. This example demonstrates how Claude mixes internal and external data gathering for a comprehensive answer. In doing so, it narrates its steps at a high level ("I'll search for recent restrictions… now I'll check your Drive for internal memos…") to keep the user informed [54] [55]. While actual interactions with Claude might not show each step so verbosely, the internal logic ensures nothing critical is skipped for complex requests.

In summary, Claude's tool-use logic can be seen as a **spectrum from zero to heavy research**. It tries to answer directly whenever possible (for speed and efficiency), but it will not hesitate to search the web or the user's files when required, especially for up-to-date or user-specific information [19] [20]. The number of tool calls scales with the complexity of the ask [25]. By following the defined categories ("Never Search," "Offer to Search," "Single Search," and "Research") and guidelines, Claude aims to provide answers that are both **accurate and current**, all while staying transparent about sources and respecting content boundaries.

Artifact Generation and Coding Workflows

One of Claude's notable capabilities is producing **artifacts** – which are essentially larger, structured outputs such as files, code, documents, or interactive content. The system prompt contains detailed rules on **when to use an artifact** versus a normal message, how to format various artifact types, and how to handle coding tasks.

When and why are artifacts used? Claude is instructed to create an artifact for *"substantial, high-quality"* content that the user might want to reuse outside the chat [56] . This includes: - **Code**: If a user asks for code that implements a solution or algorithm (especially more than a few lines), Claude should deliver it as a proper code artifact file [57] . This makes it easier for the user to download or run the code. - **Technical documents or guides**: For example, if asked to write a reference manual, a detailed report, a one-pager, or any long-form formatted text, using an artifact (e.g. a Markdown file) is preferred [58] . - **Creative writing**: Stories, poems, scripts, or any extended imaginative content should always go into an artifact [59] . This keeps the chat concise and allows the user to enjoy the content separately. - **Structured data or plans**: Things like meal plans, schedules, or study guides – basically lists or tables of information intended for reference – are better as artifacts (often as Markdown) so the user can easily save them [59] . - Generally, any **standalone text** longer than ~20 lines or ~1500 characters merits being in an artifact rather than inline in the chat [60] . Short answers can remain in the conversation, but longer ones move to a file format. The only exception is that even short creative pieces should use an artifact (e.g., a short poem would still be given as an artifact for the user's convenience) [60] .

This approach ensures the conversation itself doesn't get overwhelmed by very large blocks of text, and it allows the user to download or view longer responses more cleanly. The prompt says to prefer Markdown artifacts for structured content because users can easily save or convert those [60] .

Artifact Design Principles: Claude is given guidance on making artifacts high-quality and functional: - Only **one artifact per response** should be created [61] . Claude shouldn't split content into multiple files in one turn. If it needs to correct or modify an artifact after creation, it has an update mechanism (discussed shortly) rather than generating a new artifact from scratch [62] . - The artifact should be **complete and immediately usable** [63] . For example, if it's code, it should include all necessary parts (imports, definitions) so the user can run it. If it's an HTML or React artifact for a mini web app, it should contain a working example – not placeholders – so the user can open it and see the intended result [64] [65] . - Claude should focus on **functionality and clarity** in artifacts. For instance, in code artifacts, it's recommended to use concise but clear variable names (like `i, j` for loop indices, or short identifiers) to maximize the amount of logic that fits within context limits without sacrificing readability [66] . Essentially, write clean code but be mindful of the length. - **Visual and interactive artifacts** (like HTML/CSS/JS or React components) come with specific design advice. The prompt encourages **modern, dynamic design**: if making a small web page or UI component, Claude should consider contemporary design trends (dark mode, glassmorphism, subtle animations) to create a "wow factor" [67] . It explicitly says static designs should be the exception; even a simple UI could have a hover effect or a tiny animation to feel more alive [68] . However, for complex interactive artifacts (like 3D animations or games), performance and user experience are prioritized over decorative flourishes [69] – meaning Claude should ensure the core functionality is solid (smooth interactions, no major bugs) before adding fancy visuals. - **Accessibility and compatibility**: Claude should ensure any HTML/React it produces adheres to basic accessibility (like proper contrast, semantic tags) [70] . Also, since these artifacts run in Claude's environment, there are restrictions: for example, **no use of `localStorage` or `sessionStorage`** in any artifact's code [71] . The environment won't support those, and using them causes the artifact to fail. Instead, the assistant must use in-memory storage (variables, or React state in React apps) to preserve any data across interactions in the artifact [72] [65] . If a user specifically asks for code that uses `localStorage`, Claude is instructed to politely explain that such features are not supported in the chat interface and offer an alternative (like demonstrating with an in-memory object or explaining how the user could adapt the code outside Claude) [73].

Artifact Types: The system prompt enumerates the artifact MIME types and their intended usage: - **Code artifacts** with type `"application/vnd.ant.code"` – used for source code in any programming language. Claude will label the artifact with the language (e.g. `language="python"`) so that the interface can show syntax highlighting. - **Text/Markdown artifacts** – for plain text or Markdown documents. This is typical for essays, reports, lists, etc. - **HTML artifacts** (`"text/html"`) – for self-contained web pages. These can include HTML, CSS, and JS in one file [74]. External scripts are only allowed from a safe CDN (cdnjs) if necessary [64]. Claude should not rely on any other external resources and should not separate CSS/JS into separate files. Essentially, it creates a single HTML file that the user can open directly. All interactive behavior should be included via inline scripts. - **SVG images** (`"image/svg+xml"`) – Claude can output an SVG image code if needed, which will render as an image [75]. This might be used for simple charts or diagrams. - **Mermaid diagrams** (`"application/vnd.ant.mermaid"`) – for rendering diagrams using the Mermaid syntax [76]. The assistant can output a Mermaid code block, and the interface will visualize it. This is particularly useful for flowcharts, UML, etc., if asked. - **React components** (`"application/vnd.ant.react"`) – for interactive UI components written in React/JSX [77]. Claude can produce a functional or class component. Guidelines include that it should have no required external props (or provide defaults) and use **Tailwind CSS utility classes** for styling instead of custom CSS [65]. The environment has some pre-approved libraries for React artifacts (like Lucide icons, Recharts for charts, MathJS, Lodash, D3, Plotly, Three.js r128, etc.) which Claude can import if needed [78]. This library list ensures that if Claude needs to create a chart or a 3D visualization, it knows which tools are available.

All artifacts should be **fully included** in the assistant's answer (no partial code or "I would put more here" – it must present the entire content) [79]. If an artifact is large and something was missing or needs changes, Claude shouldn't append a second artifact; instead, it uses an **update** mechanism to modify the existing artifact content in subsequent turns. The prompt details how Claude can issue an update: by specifying the exact old snippet and the new snippet to replace it with, ensuring minimal disruption [62]. This prevents duplication and keeps the artifact version controlled. Claude can do up to 4 small updates (each under 20 lines changed) per artifact [80], and if more major edits are needed, it should do a full "rewrite" of the artifact instead (essentially replacing the whole content with a new version) [81]. These rules encourage Claude to produce a thorough, correct artifact the first time, and only fine-tune if absolutely needed, rather than iterative trial-and-error.

Coding Assistance Workflows: When the user asks for help with code or data, Claude has a couple of modes: - If the task is to **write or fix code** (and not necessarily execute it immediately), Claude will usually just produce the code (in an artifact or inline if short) along with explanations. For example, "Help me write a Python script to process this CSV" – Claude would likely output a `process_data.py` code artifact with the Python code, and perhaps a short explanation after in the chat [82] [83]. The prompt even gives an example where the user asks for code to process CSV files, and Claude directly provides the Python code (no analysis tool needed) [82] [83]. - If the task involves **analyzing data or performing calculations** (especially if a user provided a file like a large CSV/Excel or a JSON), Claude might use its **Analysis Tool** (a sandboxed JavaScript environment, also called the "REPL") to dynamically inspect the data and compute results [84] [85]. The prompt specifies when to use this: only for truly complex calculations or to handle large files that are too big to just eyeball [86] [87]. Routine arithmetic or small data manipulations should be done mentally or via reasoning – the assistant is considered "more intelligent than it thinks" and shouldn't offload trivial math to the tool [86]. But for example, if asked "calculate the 47th Fibonacci number" or "process this 10,000-row dataset," using the analysis tool is appropriate [88] [89].

The analysis tool allows Claude to execute JavaScript code (with certain libraries available) and then incorporate the results back into its answer [84] [90]. Claude can `console.log` intermediate results in that environment and those will come back as output to use. The prompt provides **best practices for using the analysis tool**: - Don't use it for non-JS code. If the user wants a Python solution, just provide Python code (without executing it) [91]. - Don't use it for simple calculations that Claude can reliably do itself (to avoid unnecessary latency) [86]. - It's mainly for heavy lifting: big multiplication/division, large data parsing, statistical analysis, reading the contents of an uploaded file, etc. [86] [91]. - When reading files in the analysis environment, use the asynchronous `window.fs.readFile(path, {encoding:'utf8'})` API provided (since synchronous file reads aren't available in the browser context) [92] [93]. Always include error handling in case the file read fails [94]. - Use libraries like **PapaParse** for CSV, **SheetJS (xlsx)** for Excel, **Lodash** for data manipulation to avoid reinventing parsing or common algorithms [95] [96]. The prompt specifically warns not to write a custom CSV parser or grouping function when a robust library can do it, and to be mindful of things like trimming whitespace in headers and handling missing values [95] [97]. - If Claude uses the analysis tool to figure something out (say, compute summary stats from a CSV), it must then present the final answer or code in an artifact or the message. The analysis environment doesn't share state with the artifact environment [90] – so Claude cannot, for instance, load data into memory via analysis and then assume a subsequent HTML chart artifact can access it. Each artifact must independently load any required data (or the analysis results must be inserted explicitly).

The combination of artifacts and the analysis tool lets Claude act as a capable coding assistant: it can **generate complete code**, **run snippets to verify outputs or parse data**, and **deliver polished results**. The prompt's thorough directives (like how to handle updates, environment restrictions, which libraries to use, etc.) aim to ensure that the user receives **usable, correct, and safe artifacts**.

For example, consider a scenario: the user uploads a large CSV file and asks Claude to analyze it and create a chart. Claude might first read the file using the analysis tool (to understand its structure or compute some summary) [98] [99], log a snippet of data to confirm it parsed correctly [100], then produce an artifact containing, say, an HTML with an embedded chart (using D3 or Chart.js) based on that data. At each step, Claude ensures it's using the recommended methods (PapaParse for CSV, not exceeding environment limitations, etc.). It might also explain in the conversation what it's doing (e.g., "I will first inspect the data…" then after the tool results, proceed to "Now I'll create the chart artifact…").

Another scenario: If a user asks for a web app interface (React component or HTML page), Claude will produce the full code with interactive features and styling guidelines applied (Tailwind classes for React, maybe a modern CSS framework style for pure HTML). It keeps in mind performance and user experience – for instance, not using too heavy animations that could lag, and definitely no usage of unsupported web storage that could break in the Claude environment [71].

In summary, **Claude's artifact and coding workflow is geared toward delivering robust, ready-to-use content**. Developers interacting with Claude can leverage this by asking for artifacts when needed (Claude will often decide on its own when an output warrants an artifact). Knowing these rules, one can expect that any request for a significant piece of code, a long document, or an interactive example will result in an artifact. The content will follow best practices (for clarity, modern style, and safety) as enforced by Claude's system prompt instructions. And if revisions are needed, Claude has a structured way to update the artifact rather than spitting out multiple versions chaotically.

Source Citation and Fact-Checking Policies

Maintaining factual accuracy and giving proper credit is a major theme in Claude's design. The leaked prompt outlines stringent **citation requirements** whenever Claude uses information from external tools like web search or document retrieval. This is intended to **prevent hallucinations and ensure transparency** in Claude's answers.

The core rule is: **if Claude's answer includes content drawn from a search result (or any external document fetched), it must cite the source** [101]. In practice, Claude uses a citation tagging system internally, but the user sees footnote-style references (like the numbers in brackets you see here). Every specific claim or statistic that comes from a tool should be marked. The prompt even mandates that **each factual claim should point to the exact supporting sentence or sentences in the source** [102]. This granularity means the user can verify any piece of information by checking the cited source and finding the corresponding line.

Some key points of the citation policy: - **Minimal but sufficient citation**: Claude should cite *every* claim based on external data, but only the minimal snippet necessary, and avoid excessive or redundant citations [48] [103]. If one source fully supports a statement, it's better to cite just that one, rather than cluttering with multiple references. - **Quotation rules**: The assistant is allowed to include at most **one direct quote** from an external source, and that quote must be extremely short – *"fewer than 15 words"* [42] – enclosed in quotation marks. Anything longer risks copyright violation or just copying content. If Claude quotes a source, it still must cite it. The prompt explicitly states **never to quote long passages**, never to quote multiple times extensively, and absolutely **no song lyric quoting is allowed** (even a short lyric, because those are very sensitive copyrighted content) [104]. - **No disallowed content via quotes**: Quoting doesn't excuse policy violations. Claude can't quote hate speech or extremist propaganda under the guise of citation (the safety policies forbid using such sources at all). So citations must also come from reputable or at least policy-compliant content. - **No "hallucinated" sources**: Claude must not make up a source or citation. If it's unsure where something came from, it should either verify it properly or drop the claim. The prompt warns never to produce a citation for something that wasn't actually found in a source [105]. This is crucial to maintain trust – each reference should be real and verifiable. If Claude is not confident about a fact's source, it should present the fact as uncertain or simply not include it, rather than attaching a random reference. - **Avoiding plagiarism**: Beyond direct quotes, even summaries of content should be handled carefully. Claude should not produce a summary that is essentially a close paraphrase of a large portion of a source (the prompt calls this a *"displacive summary"*) [106]. If the user asks for a summary of an article, Claude should ensure its summary is much shorter than the original and uses original wording, not just rewording each sentence of the source. The assistant should *never* stitch together pieces from a copyrighted text to give the user – that would be like reconstructing the original (a big no-no). - **User-provided content**: The citation rules mainly apply to content Claude obtains via tools. If the user directly provides a passage and asks Claude to analyze or summarize it, Claude doesn't need to "cite" that (the user knows they provided it). The system prompt's citation instructions focus on attributions for *search results and fetched documents* [101].

In effect, Claude acts almost like an academic writer: any external fact gets a footnote. The conversation examples in the prompt illustrate this. For instance, when asked to find an article about fisheries and ocean warming, Claude searches and finds a line in an OECD report. It provides a snippet in quotes with a citation pointing to that source [107] [108]. It then summarizes the finding and notes it can't quote more due to copyright, inviting the user to read the full article if they want more [43] [44]. This behavior aligns perfectly with the rules: a short quote to answer the question, cited properly, and a refusal to dump full paragraphs from the source.

Another example in the prompt deals with song lyrics: the user asked for the first verse of "Let It Go" in an artifact. Claude responds by **refusing to reproduce the lyrics** (since they are copyrighted), and instead offers to create an original poem in the same spirit [109]. It explains why: because it cannot provide the actual lyrics. This is part of the copyright policy – even if the user requests it, Claude won't output non-public-domain lyrics or lengthy copyrighted text. The safest it can do is describe or rephrase the content, or redirect the user.

The prompt's **mandatory copyright requirements** highlight this strongly [110] [104]. Claude is to **always err on the side of caution with content reproduction**. Even if a user tries to insist that quoting 20+ words is fair use or that they have rights to it, Claude is instructed not to engage in that debate; it should reply that it's not a lawyer and cannot judge fair use, and stick to refusal for anything clearly over the limit [111]. Additionally, Claude shouldn't volunteer discussions about copyright unless the user specifically raises it. The focus is simply: *don't output protected material.*

By enforcing citations and limited quotes, Anthropic's prompt reduces the risk of **hallucinated facts**. Knowing that it must point to a source, Claude is incentivized to double-check information via search rather than make it up. It also means that as a user or developer, you can request Claude to provide sources for its answers, and the model is literally programmed to do so when any external data is involved. This is extremely useful for verifying answers or for research assistance use-cases.

In practice, when you ask Claude something that causes it to search, expect the answer to come with citations like "【source†Lx-Ly】". Those correspond to content from a web page or document it found. The model will try to cite *only the relevant part* of the source (e.g., lines x–y) to keep it precise [112]. This mechanism is part of Claude's answer composition pipeline internally. If it doesn't find anything relevant in search results, it should honestly say it couldn't find information, rather than fabricating an answer [113].

Lastly, to avoid clutter, Claude is told not to cite things that are not actually useful to the answer. Only sources that *"impact answers"* should be referenced [47]. So it won't cite an entire article if only one sentence was needed; and it won't cite a source for trivial common knowledge (it doesn't need to cite Britannica for "Paris is the capital of France"). It uses citations to back up claims that the user might want to verify or that come from beyond Claude's own training knowledge.

In sum, the citation and fact-checking policies make Claude's responses **transparent and reliable**. Developers can trust that when Claude provides a factual statement with a citation, that reference actually supports the statement (because the assistant had to check it). Likewise, the strict avoidance of long quotes ensures Claude's output stays on the right side of copyright law and discourages users from using Claude as a way to get copyrighted text. It's a design that encourages **verification and honesty**, aligning well with the needs of research or professional usage.

Safety Filters and Content Guardrails

Claude's system prompt contains comprehensive **safety directives** to prevent harmful, sensitive, or unethical outputs. These policies cover a wide range of scenarios, from dealing with hate speech to ensuring user well-being. Here we summarize the main safety rules, especially regarding **copyright, harmful content, and hallucination risk** (which we've touched on above).

Copyright and Intellectual Property: As discussed, Claude must not output copyrighted material beyond extremely short excerpts [110]. This is both a legal and ethical boundary. The assistant is effectively barred from providing things like full lyrics, lengthy article passages, or code from copyrighted sources, even if the user requests them. Instead, Claude will offer summaries or encourage the user to access the material through legitimate means. This avoids turning Claude into a piracy tool and protects content creators' rights.

Harmful Content Avoidance: Claude has strong restrictions on hate, violence, harassment, sex, self-harm, illicit behavior, and other categories of harmful content. Some key points: - Claude should **never assist in locating or disseminating extremist or hate content** [50]. If a user asks for information from hateful sources (e.g. an extremist manifesto), Claude must refuse and not even attempt to retrieve it. It should not quote or cite such sources either, even if for the sake of argument refutation. Instead, it can stick to factual rebuttals or refer to reputable analyses of those ideas, but not the original propaganda. - Claude should not facilitate **violent or illegal acts**. This includes not explaining how to make weapons (no chemistry for explosives, no guides for 3D-printed guns, etc.) [114]. It also refuses to write **malicious code** like viruses, exploits, or anything that could be used in cyberattacks [114] [115]. Even if a user frames it innocently ("just for education, how does a malware work?"), Claude's policy is to decline. The prompt explicitly says if the code or request looks malicious or could be misused to harm, Claude should refuse outright [116]. Similarly, any advice on carrying out illegal surveillance, hacking, fraud, etc., is off-limits. - **Self-harm and medical safety:** Claude avoids content that encourages self-harm or suicide, or unhealthy behaviors like disordered eating or substance abuse [117]. If a user is seeking help in such domains, Claude will offer support or suggest seeking professional help, and certainly not give instructions that worsen the situation. The assistant "cares about people's wellbeing" and so it tries to respond in a caring, encouraging tone if the user's message indicates distress, while staying within its knowledge domain for advice [117]. - **Child safety:** Any sexual or abusive context involving minors is strictly forbidden [118]. Claude is very cautious even with seemingly innocuous content involving children, to ensure it can't be misused for grooming or exploitation. For example, if someone asked for a "romantic story between a child and an adult," Claude would refuse. In general, it errs on protecting minors in any ambiguous scenario. - **Harassment and Hate:** The assistant doesn't produce harassing content or slurs. It won't aid a user in bullying or targeting an individual or group. Also, as part of hate speech avoidance, it won't teach or spread extremist ideologies. If a legitimate query touches on such topics (e.g., a historical question about Nazi propaganda), Claude would answer using *reputable academic or historical sources* and carefully avoid lending any credibility to hateful viewpoints [119]. - **Medical and legal disclaimers:** While not explicitly a large part of the prompt, usually these systems avoid giving explicit medical or legal advice. Claude's prompt says it provides emotional support with accurate medical terminology where relevant [120] but will avoid going beyond its role. For legal questions (like "is this fair use?" or "am I violating copyright?"), Claude will not give a firm judgment – it will say it's not a lawyer and thus cannot determine legality [111]. - **Misinformation:** Claude should not knowingly spread false information, and if a user asks something that is a known hoax or conspiracy (e.g., anti-vaccine myths), Claude will correct it with factual data. If Claude isn't sure about something, it's expected to either research or clearly state uncertainty rather than invent an answer. The hallucination prevention (through citations, etc.) is part of avoiding inadvertent misinformation.

Vision (Image) Safety – "Face Blind" rule: A very striking policy in Claude's prompt is that the assistant must act **completely face-blind** to images of people [121]. If a user shares a photo with a person and asks who it is, Claude is not allowed to identify them – even if it's a famous celebrity. Claude will never say "That's [Name]" or even "This person is a well-known actor." It will behave as though it does not recognize the face at all [121]. Instead, Claude might describe non-identifying features (e.g. "a man in a black suit smiling at a camera") and perhaps suggest *"I'm not able to identify people in images"*. The prompt clearly states Claude should not even hint at recognition or use the person's identity in a search query [49] [121]. If the user reveals who the person is ("That's my friend Jane" or "That's actor Tom Cruise"), Claude can talk about that person in general terms (like listing their movies or achievements) **but still must not confirm the photo is actually them** [122]. It will treat it as *"assuming you say it's Tom Cruise, I know Tom Cruise is an actor who… etc"* without ever using phrasing that implies visual confirmation. This policy is likely to prevent privacy violations and the potential misuse of facial recognition AI.

For images without people, Claude will discuss them normally (describing objects, scenery, etc.) as long as it doesn't violate other rules. The prompt also tells Claude to **reiterate any written instructions visible in an image** to ensure it doesn't miss user-provided directives embedded in images [123]. But anything verging on identifying a private individual from an image is off-limits.

General Refusal Style: When refusing any request due to policy (be it copyright, violence, personal data, etc.), Claude should do it briefly and without moralizing. The instructions say to explicitly mention what part of the request it cannot fulfill *at the start of the response* [10]. For instance, "I'm sorry, but I cannot help with that request" or "I cannot assist with creating that content." Then, if possible, offer an alternative (e.g., a summary instead of an excerpt, or a safer form of the info). If no alternative is appropriate, just apologize and maybe ask if it can help with something else. The prompt discourages lengthy refusals or lectures on policy ("does not say why or what it could lead to" [10]). So Claude won't say *"I can't do that because it's illegal and against policy and could be harmful,"* it will simply and politely decline.

Benefit of the Doubt and Edge Cases: The guidelines mention that if a user's request is ambiguous between a harmful interpretation and an innocent one, Claude should **assume the legitimate, safe intention** [124]. For example, if someone says "How do I get rid of a problem?" and it's not clear if they mean a *literal* problem or something harmful, Claude would default to a benign interpretation. However, if the user's intention appears clearly harmful or exploitative (especially towards vulnerable groups), Claude is told *not* to stretch to find a benign interpretation [125]. In such cases, it should refuse rather than help. A concrete scenario given: if a user seems to be asking for ways to scam elderly people (targeting a vulnerable group), Claude should not try to rationalize it as something else – it must promptly refuse and not enable it [125].

Hallucination Risk Management: We've covered this under citations, but to reiterate in the safety context: hallucinations (making up facts or sources) are considered a "safety" issue insofar as they can mislead users. The prompt's heavy emphasis on citation and verification is the primary mitigation. Additionally, Claude's knowledge cutoff (end of Jan 2025) is explicitly stated, and it is instructed to use search for events after that date [15]. This prevents it from confidently stating something about late 2025 events from outdated training data. If asked about something after its cutoff, Claude should either search or clarify that it doesn't have recent info. For example, the prompt includes information about the **Nov 2024 U.S. election** (Trump defeating Harris) that Claude "knows" but also says Claude won't mention it unless relevant [126]. If asked, it can provide that info, but if the conversation is not about that, it won't randomly mention it. This shows a controlled approach to knowledge updates – integrating some known critical post-cutoff info (perhaps to avoid confusion since that election was a major event) while still generally requiring search for current events.

All these safety measures are aimed at making Claude a **reliable and non-harmful assistant**. For developers, it means you can expect Claude to refuse requests for disallowed content and you don't have to worry as much about it going off the rails with toxic or dangerous output. It also means if you *do* need something borderline (like analyzing extremist content for research), you'd need to structure the request very carefully – and even then Claude might only summarize from a distance or refuse citing primary sources. In everyday use, these filters help keep the interaction productive and within legal/ethical bounds.

User Preferences and Contextual Adaptation

Anthropic’s system prompt reveals that Claude can take into account **user-specified preferences and styles** to personalize its responses. This is a sophisticated feature allowing the assistant to adjust tone, formality, language, and contextual framing according to the user’s profile or instructions, as long as doing so remains relevant and helpful.

Behavioral vs Contextual Preferences: The prompt distinguishes two kinds of user preferences that might be provided: - **Behavioral Preferences**: These are instructions about how the user wants Claude to behave or format responses *generally*. For example, “Always provide step-by-step solutions” or “Please keep explanations very simple” could be considered behavioral preferences. It could also include language preferences (e.g., “I only want responses in French”) or format choices (e.g., “I prefer bulleted lists”). - **Contextual Preferences**: These describe the user’s personal context, background, or interests. For instance, the user might indicate their profession (“I’m a physician”) or hobbies (“I love analyzing data and statistics”) or skill level (“I’m new to programming”) as part of their profile.

Claude is instructed to handle these preferences carefully. It should not blindly apply them to every answer, but rather interpret *when* they are relevant: - A **behavioral preference** that says “always do X” should indeed almost always be followed (unless it conflicts with a direct user request or a safety rule) [127]. For example, if the user’s profile says “always be very concise,” Claude will try to keep its answers short in all cases by default. - If the preference is not an “always” rule but a general note, Claude should apply it **only when it improves the response without causing confusion** [128]. If a user who loves statistics asks for a short story, Claude shouldn’t inject statistics into the story just because of that stated interest (that would be jarring). The prompt explicitly gives this example: *Preference:* “I love analyzing data,” *Query:* “Write a short story about a cat” – it says **do not** apply the preference here [129]. - **Contextual preferences** (like profession or interests) are only to be applied when directly relevant to the query [130]. So if a user is a physician and they ask a medical question, Claude can use more advanced terminology, assuming the user’s background means they’ll understand [131]. But if that same user asks about a programming problem, Claude shouldn’t start using medical metaphors or assume their medical background matters for coding [132]. - If the user explicitly asks for personalization – e.g. “Given my background in finance, how would this apply to me?” – then Claude will definitely incorporate the context [130]. Or if the query itself is about the user’s preference domain (say the user is a sommelier and they ask about wines), then obviously Claude can leverage that info (maybe using more wine jargon knowing the user is an expert, etc.).

The prompt provides **clear negative examples** where preferences should not bleed into answers: - A user who loves space exploration asks a recipe question – Claude should *not* turn it into a space-themed answer unless asked [133]. - A user who is an architect asks to fix Python code – Claude should not mention architecture or change the explanation style to architectural terms [132]. - A user mentions they are a sommelier (wine expert) but then asks an unrelated tech question – Claude shouldn’t even mention wine in that context [134].

Claude must be very cautious **never to force a preference in where it doesn’t belong**. It should not start responses with phrases like “Since you’re a [profession]…” unless the user’s role is directly relevant to the question at hand [135]. The rule of thumb given is to only incorporate these details if they *“materially improve response quality for the specific task.”* [136]

User Style Selection: Apart from preferences, the platform might allow the user to choose a particular **writing style** for Claude (e.g., “Friendly”, “Formal”, “Succinct”, etc.). When a style is selected, Claude receives a `<userStyle>` tag with instructions on tone and format to adopt. For example, a

“Storyteller” style might instruct Claude to be more narrative and whimsical, whereas a “Professional” style might enforce a formal, terse tone.

Claude will follow the latest selected style throughout the conversation, but **not at the expense of accuracy or appropriateness** [137] . If the style says to be very casual, but the user asks for a complex technical analysis, Claude will still deliver a thorough analysis (just phrased more casually). It won’t omit crucial details just because the style is simple. The prompt clearly states not to compromise completeness or correctness for style’s sake [137] .

Users can switch styles mid-conversation. Claude will then adapt to the new style from that point onward [138] . If a user is unhappy with the tone (maybe they forgot they set a style), and they manually instruct a different tone in their message, Claude should prioritize the user’s direct instruction over the previously set style [137] . Claude might even gently mention that it’s using a certain style if the user seems confused or repeatedly asks for a different approach – and suggest they can change it in the settings if needed [139] . However, normally, Claude does not overtly mention the style or preferences on its own [140] [141] . The user’s profile info is to be kept *implicit* unless the user brings it up.

Additionally, **language preferences** are honored. If a user’s profile or request says “I only want to speak in Spanish,” Claude will comply and respond in Spanish, even if the question was posed in English [142] . But again, it won’t randomly switch languages without being told – default is to follow the language the user is using unless instructed otherwise.

Example to illustrate: Suppose a user sets their preference that they are a beginner in programming and prefer simple explanations. If they ask “What is a recursive function in Python?”, Claude will recognize that their profile says they’re new to programming and thus make an effort to explain recursion in very accessible terms, maybe with a relatable analogy [143] . It might avoid heavy jargon or at least define any jargon it uses, because it knows the user’s level. This improves user satisfaction as the answer is tailored to their background.

Conversely, if an expert programmer asks the same question, Claude might give a more concise, technical definition, possibly assuming familiarity with certain terms – because a preference like “I am a professional Python developer” would signal Claude to go straight to the point in a technical manner [144] .

The system prompt’s handling of preferences means **Claude can provide a personalized experience** without the user having to repeat their life story each time. For developers integrating Claude, this indicates you can set user profiles (like their profession, skill level, or format preferences) and Claude will use that to refine its outputs appropriately. It’s a form of implicit prompt engineering that persists across the conversation (but not across separate chats – each conversation can start fresh with new or default preferences).

One must be mindful, though, that if a user’s stated context is not relevant, Claude will ignore it to avoid bizarre responses. So it’s not a fully open-ended persona system, but a context-aware adjustment mechanism. And importantly, if a user updates their preferences or explicitly says “don’t do X anymore,” Claude will respect the latest instruction. The prompt says user instructions in conversation override any previously provided preferences or style settings [145] [137] . This ensures that if the user changes their mind about tone or detail level, Claude adapts immediately.

Finally, the assistant is instructed that the user *cannot see these internal preference tags or style guidelines*, so it shouldn’t reference them unless necessary [146] . This means Claude won’t say “I’m responding this way because your profile says you're a doctor," unless the user is puzzled and explicitly asks "why are you using so many medical terms?" In which case, Claude might explain it assumed they were comfortable with them due to their stated background, and remind that they can adjust preferences in settings [146] .

In summary, **Claude's preference and style modeling allows a degree of customization and user-alignment** that can significantly enhance the quality of interaction. For developers, leveraging this might involve setting the right context at conversation start (via a profile or example messages) to get the desired tone and level from Claude. It's a powerful tool: you can have Claude automatically be more formal for one user and more playful for another, or switch languages depending on who's asking – all without explicit prompts each time, as long as those preferences are conveyed to the system. Understanding this, one can craft better user experiences by appropriately using the preference and style features provided by Claude's API or interface.

Prompting Claude: Developer Guidance

Knowing the rules and behaviors above, how can developers or advanced users best **interact with Claude** to get optimal results? By aligning prompts with Claude's internal design, you can often get more efficient and relevant answers. Here are some guidelines drawn from the prompt's implications:

- **Be clear and specific about what you want**: Claude's prompt handling is improved when you give detailed instructions. If you need an answer in a certain format or with certain content, say so explicitly. For instance, "Provide a step-by-step solution" or "Please give the answer in bullet points" (if you want bullet points) will steer Claude accordingly. Clarity helps Claude decide if it needs to search or not, and how to structure the output [147] .
- **Indicate the scope or depth**: If you want a comprehensive analysis, use words that trigger Claude's research mode. For example, asking "Can you do an in-depth comparison of X and Y, with recent data?" signals that multiple sources might be needed. Claude will likely then perform several searches to compile a thorough answer [54] [53] . Conversely, if you want a quick fact and not a long explanation, you can say "Briefly, what is..." and Claude will try to be succinct.
- **Leverage the knowledge cutoff**: Remember Claude's knowledge is solid up to end of Jan 2025, but for anything after that, it relies on search [15] . If you need current information (like a 2026 development), phrase your question to make that clear (e.g., "as of 2026, what is..."). Claude is programmed to then immediately use web search for "current/latest" queries [19] . If you don't hint at recency and the topic isn't obviously new, Claude might give an older answer from memory and only optionally offer to update. So, include a timeframe if needed.
- **Avoid ambiguous phrasing around disallowed content**: If your question could *innocently* be interpreted as asking for something disallowed, try to clarify it. For example, "How to deal with termination?" might be misread (is it employment termination or something harmful?). A safer phrasing like "How to handle terminating an employee's contract fairly" ensures Claude doesn't misinterpret and apply an irrelevant safety filter. Knowing Claude's caution, a little specificity can prevent an unnecessary refusal.
- **Use the first person for internal data**: If you have internal tools enabled (like Google Drive, Gmail, etc.), you might need to mention "my" or "our" to clue Claude to search those instead of the web for company-specific info [148] . For example, "find **our** Q3 sales presentation" prompted Claude in the example to use the Google Drive tool [149] . So, when appropriate, use possessive language to tap internal knowledge (assuming you have those integrations available).
- **Expect and utilize artifacts**: If you ask for something that sounds like a deliverable (code, a long report, a story, a chart), be prepared for Claude to return it as an artifact file. You can even encourage this: e.g. "Give the answer as a markdown document" or "provide the code in a

Python file". Claude will then oblige, creating a nicely formatted artifact. This is helpful because artifacts can preserve formatting (like tables, headings, or runnable code) better than plain chat.

- **Prompt for reasoning if needed**: Claude generally hides its internal chain-of-thought, but if you want a step-by-step explanation, you can ask, "Please show your reasoning" or "Explain how you got this answer." Claude can then produce a more explicit reasoning or even a step-by-step solution (it will still follow the rules: no multiple questions to the user at once, etc., but it can break down its thought process for you).
- **Use examples in your prompt**: The system prompt itself notes that providing **positive and negative examples** in the user query can help Claude understand the desired style or content [147]. For instance, you can say: "Explain X. For example, I liked how you explained Y in simple terms. Do it like that." Claude will adapt to mimic the style of the example, since the prompt encourages using user-provided examples as guidance [138]. This is a form of few-shot prompting that Claude supports.
- **Style and tone requests**: Instead of relying on the `<userStyle>` feature (which may be set via UI or API), you can directly instruct Claude in the prompt if needed. For example: "Answer in a humorous tone" or "Use formal academic language." Claude will accommodate as long as it doesn't conflict with other rules. It already is built to be flexible in tone, and an explicit ask will override any default style (the last user instruction wins in such cases) [137].
- **Cite or source requests**: If you want Claude to provide sources even for things it knows, you can ask "Can you give references?" Usually, it cites only when using search results. But if you specifically request sources for a factual answer, Claude can sometimes provide a citation from its knowledge base or a general source (though it might need to search anyway to get a citable reference). Given how strongly citations are emphasized for external info, asking for them will make Claude careful to back up claims.
- **Don't ask for obviously disallowed content**: This might go without saying, but attempts to get Claude to produce disallowed content are futile due to the extensive safety guardrails. For example, asking for violent wrongdoing plans, explicit sexual content, personal data on someone, or large copyrighted text will lead to polite refusal [114] [109]. It's a waste of time and could cause the session to be flagged. If you need information that tangentially touches sensitive areas, frame it academically or for a legitimate purpose and ensure it doesn't cross into instructions to do harm or privacy violations.
- **Use iterative refinement**: If Claude's answer isn't exactly what you need, you can ask follow-ups to clarify or adjust. Claude is quite capable of revising its output. For instance, "Can you make that explanation simpler?" or "Please provide more detail on the second point," will prompt Claude to modify its style or content accordingly, given it has the conversation context. It will not be offended by corrections; in fact, the prompt encourages it to accept and analyze user corrections thoughtfully [12].

In essence, to get the best from Claude, **guide it in the direction you want using clear instructions and context cues**. The model is already inclined to behave helpfully thanks to its system prompt – it will try to answer thoroughly, check facts, and stay safe. But if you, as a developer, know these internal rules, you can craft your requests to exploit them. For example, if you want a full research report, use language that triggers the research workflow (Claude will then not stop at the first answer but go the extra mile with multiple sources). If you want a more conversational answer with empathy, you can either rely on its detection of a personal context or directly say, "I'm feeling down, can you give some advice?" – it will then know to be more gentle and supportive [120]. If something factual seems outdated, prompt Claude to "check the latest info on…" and it will remember to search due to the cue of recency.

Furthermore, the system prompt itself suggests that Claude can educate users on effective prompting if asked [147]. As developers, we can take that meta-advice: it suggests being **detailed, providing examples of desired output, encouraging step-by-step reasoning, and specifying format or length** as ways to get the most helpful answers [147]. Those are exactly the techniques one should use with Claude (or any LLM) to reduce ambiguity and increase quality.

By aligning your interaction with Claude's built-in policies and decision-making, you essentially "speak its language," making it easier for Claude to deliver what you need without misunderstanding. The result is a more efficient collaboration between you and the AI – with Claude leveraging its powerful capabilities (like retrieval, coding, and structured output) in the way it was intended, guided by you.

Conclusion

The leaked Claude system prompt offers a rare glimpse into the **governor under the hood** of a large language model. We see an AI assistant engineered with layered rules: from high-level principles of helpfulness and harmlessness, down to specific procedures for web search, code generation, and style adaptation. Key takeaways include Claude's commitment to answer using internal knowledge when possible (to save time), its graduated approach to research (ensuring thoroughness for complex tasks), and rigorous citation and content safety practices that keep responses transparent and trustworthy [2] [6].

For developers, understanding these mechanisms means they can better predict how Claude will handle different requests and can craft prompts to utilize its strengths. Claude is essentially a **research analyst, coder, and conversational partner combined**, with guardrails to keep it factual and safe. By providing clear instructions and relevant context, users can unlock high-quality outputs – whether it's generating a piece of code with correct syntax and modern best practices, summarizing a document with proper citations, or engaging in a nuanced dialogue that respects personal preferences.

This white paper has distilled the prompt's content to highlight the design philosophy behind Claude: an AI that is **helpful, correct, and considerate by design**, due to explicit rules that govern its every move. As generative AI developers, incorporating such principles into our own prompting and system designs can lead to more effective and responsible AI interactions. Claude's example demonstrates that with the right prompt architecture, an AI assistant can be both **a creative problem-solver and a cautious guardian** of information, providing value while mitigating risks.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] [104] [105] [106] [107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119] [120] [121] [122] [123] [124] [125] [126] [127] [128] [129] [130] [131] [132] [133] [134] [135] [136] [137] [138] [139] [140] [141] [142] [143] [144] [145] [146] [147] [148] [149] claude-system prompt.txt
file://file_00000000fa6c71f49dd75904aa48ae2f