ChatGPT

Comprehensive Codex Prompt Template and Best Practices

Prompt Quality Criteria (Industry Standards)

Industry best practices emphasize several key criteria for a high-quality Codex prompt. A well-crafted prompt is **clear**, provides sufficient **context**, follows a logical **structure**, and specifies the **intent** or desired outcome explicitly [1]. These factors directly influence the relevance and correctness of the AI's output. Below we detail each quality parameter and why it matters:

- **Clarity of Instructions:** The prompt should be unambiguous and easy to understand. Clearly state the task or question so the model isn't guessing at your meaning. Vague queries like *"Help me with this code"* can lead to irrelevant or generic answers, whereas a specific request like *"Explain what the following C# method does, line by line"* is far better. As one guide notes, *"Clarity is king. A specific prompt yields a more accurate response"* [2]. Think of instructing a junior developer – you would spell out the requirements plainly. Similarly, *"imagine you were giving the task to a human intern – what details would you provide?... Models cannot ask clarifying questions mid-prompt, so your prompt must be self-contained and clear"* [3]. Ensuring the prompt is self-contained and precise will minimize confusion.

- **Sufficient Context:** Provide all relevant background information or code context the model needs. If the task relates to existing code, include the snippet or describe the environment (framework, database, etc.). LLMs don't have implicit knowledge of your specific project, so you must supply it. For example, if asking to debug a function, present the function code and maybe how it's used. *"Relevant background information or an explanation of the purpose of your request can help the model generate the desired response... Don't be vague or exclude context"* [4]. If code is provided, **delimit it clearly** (e.g. with triple backticks `` ``` for code blocks, or quotes) so the model can distinguish between instructions and reference code [5]. OpenAI's guidelines suggest using delimiters like `"""` or markdown fences for clarity when giving examples or code snippets [6]. The goal is to avoid any guesswork about the scenario – the model should have the necessary information upfront.

- **Structured Format:** Organize the prompt in a logical, easy-to-follow manner. Structured prompts (using bullet points, steps, or sections with headings) help the model parse complex instructions and ensure no part is overlooked [7] [8]. If you have multiple requirements, list them out rather than merging into one long paragraph. For instance, instead of a single block of text describing several tasks, break them into an ordered list: 1) do X, 2) then do Y, etc. Models like GPT-4 and Claude tend to mirror a well-structured prompt in their output, addressing each point in order [9]. A structured prompt also makes it easier to request a structured answer. As an example, you might write: *"Summarize the pros and cons, using separate bullet lists for each."* Because the input is clearly structured, the model is more likely to respond in a structured way. In summary, **formatting** the prompt with headings, lists, or step-by-step sections improves clarity and completeness of the response [8].

- **Specific Intent and Goal:** A good prompt makes the intended outcome explicit. Specify **what** you want the model to produce and **how**. Are you asking for a code snippet, an explanation, a translation of code from one language to another? Stating the intent guides the model's approach. For example, *"Translate this T-SQL query into a MongoDB query"* or *"Refactor the code for readability without changing functionality."* Include details about the expected output format or style: if you want a function in C#, say so; if you need a step-by-step explanation, request that. **Intent specificity** also means conveying the criteria for success – e.g. "the code should handle edge cases X and Y" or "the explanation should be in simple terms." According to OpenAI's best practices, you should *"clearly state how you want the response. If you want just code, say so. If you want an explanation with the code, request it... If you need a specific format (JSON, HTML, etc.), explicitly instruct that format"* [10] . By being explicit about the desired outcome and format, you prevent the model from making incorrect assumptions. This includes naming the programming language or framework version if it's not obvious (for instance, *"Provide a PowerShell script..."* or *"Using C# 10, do X"*). In short, leave no ambiguity about what the AI's output should look like.

Together, these parameters – clarity, context, structure, and intent specificity – serve as a **rubric for prompt quality**. A prompt excelling in all four is far more likely to yield a useful and accurate result [1] .

Quality Parameters and Template Evaluation

Using the above standards, we can evaluate the proposed Codex prompt template. Below is a breakdown of how the template addresses each quality parameter, along with a qualitative "grade":

- **Clarity: Excellent.** The template forces clarity by design – it has a dedicated **Task/Goal** section where the user must clearly state the objective. Ambiguous instructions are avoided because the template explicitly asks for what needs to be done (e.g. *"Write a function to...", "Explain what...", "Identify and fix..."*). By filling out this section in concrete terms, the user produces a clear directive for the model. This focus on a single, well-defined task significantly reduces the chance of confusion.

- **Context: Excellent.** The template includes a **Background/Context** section and encourages providing code snippets or relevant details (with proper markdown formatting). This ensures that all necessary information – such as existing code, data schema, or environment description – is supplied to the model. Because the template explicitly prompts the user for context, it helps prevent the common mistake of assuming the AI knows things it does not. In terms of our criteria, the template scores high by allocating space for context and advising how to present it (e.g. using triple backticks for code). The resulting prompt will be self-contained with the needed background, which is ideal for Codex.

- **Structure: Excellent.** The template is highly structured with clear sections: Background, Task/Goal, Specific Requirements, Examples, Output Format, and Additional Instructions. This organized layout aligns perfectly with best practices for complex prompts [8] . Each aspect of the request is separated, which not only makes it easier for a human to fill out, but also easier for the model to follow. The use of **bullet points** (for requirements) and **sections with bold headers** provides a logical flow. Models tend to respect this structure and address each part in order [7] , so we expect more complete and focused answers. Overall, the template's structure ensures no component of a good prompt is missing or buried in text. It demonstrates top-tier organization.

- **Intent Specificity: Excellent.** The template directly asks for the **Output Format** and any **Specific Requirements**, which covers intent specificity. By specifying the desired output (e.g. "only provide code" or "include an explanation"), the template makes the user articulate exactly what they want from the model. It also has an **Additional Instructions** section where the user can mention any constraints or preferences (for example, "use .NET 6 only" or "the explanation should be in bullet points"). This level of detail ensures the model's purpose is crystal clear. There is little room for the model to misinterpret the goal or produce an undesired format. The explicit nature of the template on this front warrants an excellent rating – it aligns with the guideline to *"include as many relevant details as possible"* for the task at hand [11].

In summary, the proposed prompt template **meets or exceeds expectations** across all key quality parameters. It earned "excellent" marks for clarity, context completeness, structured formatting, and intent specificity. By guiding the user to include all these elements, the template maximizes the likelihood of a high-quality result from Codex. (Of course, the ultimate outcome also depends on the correctness of the content the user provides, but the template ensures the form of the prompt is optimal.)

Known Anti-Patterns in Prompt Design to Avoid

Even with a good template, it's important to steer clear of common prompt design mistakes. Here are several **anti-patterns** and pitfalls that industry experts warn against:

- **Ambiguous or Open-Ended Instructions:** Avoid vague prompts like *"Make this better"* or *"Do it differently."* Such instructions lack clear criteria. The model might guess what "better" means and get it wrong. Instead, **be explicit** about improvements or changes you want. For example, *"Improve the code's readability by using meaningful variable names and adding comments"* is much more specific and helpful [12]. Always replace fuzzy directives with concrete goals.

- **Overloading the Prompt with Irrelevant Context:** Providing *too much* context (especially irrelevant details or entire codebases) can confuse the model or exceed token limits. Don't dump your whole project code if the question is only about one function. **Focus** on the relevant pieces. For instance, rather than pasting an entire repository, you might say, *"Focus on reviewing the* `CalculateTotal()` *function (shown below) and ignore other modules"*. This gives necessary context without overloading [13]. In short, include context, but **don't include extraneous information** that distracts from the task.

- **Insufficient Constraints or Details:** The opposite of overloading is providing too little guidance. Prompts that are underspecified can lead to undesired results. For example, *"Generate test cases for this function"* is open-ended – the model might produce a few trivial tests. It's better to add constraints: *"Generate unit tests using NUnit that cover edge cases (null input, max values) and expected error conditions"*. Without such specifics, the model may not know what quality or scope of output you need [14]. Always mention important constraints (performance needs, libraries to use/avoid, style preferences, etc.) so the solution fits your requirements.

- **Multiple Requests in a Single Prompt:** Asking the AI to perform several unrelated tasks at once (without structure) is a recipe for confusion. For example: *"Explain the code and optimize its performance and also suggest security improvements."* The model might do one part and neglect the

rest. If you have multiple objectives, split them into separate prompts or clearly enumerate them as sub-tasks in one prompt. A prompt should have a **focused goal** or a well-structured list of goals. As a rule, *"avoid adding multiple requests to a single, unstructured prompt"*, which can result in incomplete or mixed-up responses [15].

- **Unclear Output Format:** Vague instructions on the format can yield outputs that are hard to use. If you don't specify, the model might return a verbose explanation when you wanted just code, or vice versa. Always clarify the expected format: e.g., *"Respond with only the SQL query,"* or *"Provide a JSON object as shown in the example."* An anti-pattern is to assume the model will know the format – often it won't unless told. The template's **Output Format** section is designed to avoid this pitfall. Similarly, if you need the answer in bullet points, say so. If you require markdown formatting (like an answer with proper code blocks), make that clear. Essentially, **don't make the format an afterthought**.

- **Ignoring the Model's Limitations:** While not a prompt formatting issue per se, it's a bad practice to ask for things beyond the model's capability or that violate usage policies (e.g., asking Codex to produce very large code beyond the context window, or to output disallowed content). Pushing the model with such prompts can lead to errors or refusals. Design prompts that are feasible in scope. If a task is very large, break it into smaller prompts iteratively rather than one huge prompt. Avoid instructions that conflict (e.g., "Be concise" and "also explain everything in detail" at the same time). Such contradictions confuse the model.

By being aware of these anti-patterns, you can **avoid prompt designs that yield poor results**. The provided template inherently helps avoid many of them (for instance, it asks for one clear task, it separates context from instructions, and it asks for format), but it's still possible to misuse it. Always double-check that your filled-in prompt is specific, focused, and complete, and that you're not including anything that could lead the model astray.

Final Codex Prompt Template (Downloadable)

Below is the final **Prompt Template** incorporating all the best practices discussed. It is presented in Markdown format and includes placeholder sections to guide you in crafting prompts for tasks like code generation, explanation, debugging, refactoring, and translation. The template also comes with **practical examples** for C#, .NET, PowerShell, and T-SQL, demonstrating how to fill it out for common use cases. You can download the template as a Markdown file for reuse.

Download the Template: **CodexPromptTemplate.md** – *(Right-click and Save Link As to download the file)*

```
# Codex Prompt Template

## Instructions:
Use this template to craft prompts for code-related tasks. Fill in each section
and remove the placeholders and examples.

## Template Outline:
**Background/Context:** *(Describe any necessary context or code needed for the
task. Include code snippets if relevant, enclosed in triple backticks with
```

````
language tags for clarity.)*

**Task/Goal:** *(Clearly state the primary objective, e.g., "Generate X",
"Explain Y code", "Debug the error in Z", etc.)*

**Specific Requirements:** *(List any specific requirements or constraints,
e.g., performance goals, libraries to use/avoid, output formatting, coding style
guidelines.)*

**Examples (if any):**
*(Provide input-output examples or references if helpful. This guides the model.
Use triple backticks for code or input/output examples.)*

**Output Format:**
*(Describe the desired format of the response: e.g., "Provide only the code",
"Code followed by explanation", "Output as JSON", etc.)*

**Additional Instructions:** *(Any final instructions such as "Include comments
in code" or "Avoid using recursion", etc.)*

## Example Scenarios:

### Example 1: Code Generation (C#)
**Background/Context:** We have a list of user objects and need to filter active
users and sort them by registration date.
**Task/Goal:** Write a C# function `GetActiveUsersSorted(List<User> users)` that
returns the active users sorted by their `RegistrationDate`.
**Specific Requirements:** Use LINQ for filtering and sorting. Assume `User` has
properties `IsActive` (bool) and `RegistrationDate` (DateTime).
**Examples:** *(No specific examples provided, straightforward logic.)*
**Output Format:** Provide only the C# code for the function, enclosed in a
Markdown code block with language identifier.
**Additional Instructions:** Include necessary `using` statements if any.

### Example 2: Code Explanation (PowerShell)
**Background/Context:** A PowerShell script is provided that automates user
account creation.
**Task/Goal:** Explain what the following PowerShell script does, step by
step.
**Specific Requirements:** The explanation should be in clear, non-technical
language for a junior IT audience.
**Examples:** *(The script is provided in the context below.)*
```
<PowerShell script snippet here>
```
**Output Format:** The answer should be a few paragraphs explaining the script's
logic (not just a list of commands).
**Additional Instructions:** If certain commands or parameters are unusual,
````

```
briefly describe their purpose.

### Example 3: Debugging (T-SQL)
**Background/Context:** A T-SQL query intended to calculate total sales is
returning incorrect results.
**Task/Goal:** Identify and fix the bug in the T-SQL query below.
**Specific Requirements:** The query uses an INNER JOIN which might be causing
row duplication. Suggest a fix (e.g., using `SUM(DISTINCT ...)` or adjusting the
JOIN).
**Examples:** *(Problematic query provided below.)*
```
<SQL query snippet here>
```
**Output Format:** Provide the corrected T-SQL query and a short explanation of
the change.
**Additional Instructions:** Only modify the necessary part of the query; do not
rewrite the entire query.

### Example 4: Refactoring (C# .NET)
**Background/Context:** A C# method is too long and has repeated code.
**Task/Goal:** Refactor the `ProcessRecords(List<Record> records)` method for
better readability and maintainability.
**Specific Requirements:** - Break the method into smaller, focused functions if
needed.
- Eliminate duplicate code by reusing logic.
- Ensure no change in external behavior.
**Examples:** *(Original method is given below.)*
```
<C# method snippet here>
```
**Output Format:** Provide the refactored C# code in a code block, followed by a
brief explanation of the improvements.
**Additional Instructions:** Preserve any original comments and ensure the
refactored code passes all original tests.
```

In the template above, *italicized text* indicates instructions for the prompt writer (to be removed in the final prompt). By following this template, a user can fill in each section with the specifics of their request. The result will be a prompt that is clear about the task, contains all necessary context, is well-structured, and explicitly states the desired outcome — in short, a prompt that adheres to the highest quality standards for Codex.

Sources: The recommendations and template design are informed by OpenAI's and others' prompt engineering best practices, including clarity and context guidelines [3] [4], structured prompting techniques [8], and expert advice on specifying intent and format [10]. The anti-patterns are adapted from common pitfalls noted in industry literature [12] [14] and official guides. By integrating these insights, the provided template aims to maximize Codex's performance across code generation, explanation, debugging, refactoring, and translation tasks. Each example scenario demonstrates how to apply the template for a

different use-case, ensuring the template is both **comprehensive and practical** for real-world programming assistance.

---

[1] What is Prompt Management? Tools, Tips and Best Practices | JFrog ML
https://www.qwak.com/post/prompt-management

[2] [3] [5] [6] [7] [9] [10] [11] Prompting AI for Code Generation: Best Practices and Model Insights (2025)
https://www.switchlabs.dev/post/prompting-ai-for-code-generation-best-practices-and-model-insights-2025

[4] [8] [15] LLM Prompting: How to Prompt LLMs for Best Results
https://www.multimodal.dev/post/llm-prompting

[12] [13] [14] All the Wrong (and Right) Ways to Prompt: A Tiny Guide | by Aalap Davjekar | Medium
https://aalapdavjekar.medium.com/all-the-wrong-and-right-ways-to-prompt-a-tiny-guide-5bd119d312b3