

Session 8. Vibe Coding

“프롬프팅” 아닌
“프롬프트 엔지니어링”

강수진 | 더 프롬프트 컴퍼니 CEO & 프롬프트 엔지니어

Hands-On Practice
Live prompts + code walkthroughs

목차

- 01** The Challenges in Prompt Engineering
- 02** Two Approaches to Improving Your Prompt: Conversational and Structured
- 03** 8 battle-tested prompts for Vibe Coding
- 04** Live mini-demos
- 05** Warp-Up

LLM Code Generation

프롬프트 엔지니어링의 어려움

Prompt Variability Effects on LLM Code Generation (2025.06)

6.10204v1 [cs.SE] 11 Jun 2025

Prompt Variability Effects On LLM Code Generation

Andrei Paleyes, Radzim Sendyka, Diana Robinson, Christian Cabrera, Neil D. Lawrence

Department of Computer Science and Technology, University of Cambridge

United Kingdom

Email: {ap2169, rs2071, dmpr3, chc79, ndl21}@cam.ac.uk

Abstract—Code generation is one of the most active areas of application of Large Language Models (LLMs). While LLMs lower barriers to writing code and accelerate development process, the overall quality of generated programs depends on the quality of given prompts. Specifically, functionality and quality of generated code can be sensitive to user's background and familiarity with software development. It is therefore important to quantify LLM's sensitivity to variations in the input. To this end we propose a synthetic evaluation pipeline for code generation with LLMs, as well as a systematic persona-based evaluation approach to expose qualitative differences of LLM responses dependent on prospective user background. Both proposed methods are completely independent from specific programming tasks and LLMs, and thus are widely applicable. We provide experimental evidence illustrating utility of our methods and share our code for the benefit of the community.

Index Terms—large language models, code generation, sensitivity analysis

I. INTRODUCTION

¹ Since the emergence of Large Language Models (LLMs), one of their most widespread applications is code generation [1]. Ability of LLMs to generate code has the potential to bring many advantages, such as lowering the barriers to coding, democratising and accelerating software development [2]. Notably, requirements for generated code are formulated by end users in natural language, often lacking any partic-

what kind of structure around inputs is needed to produce the same software outputs - this might be external or internal to the input LLMs. For example, externally, we may need to provide more guidance to users. Internally, this could involve techniques such as generating other related inputs based on distance and averaging them together to regularise the output generated. In order to understand the scope of this challenge, we need to first test how sensitive LLM outputs are to changes in inputs. This paper makes strides towards addressing this challenge. In it we demonstrate a pipeline for measuring the difference between sets of inputs, and their impact on code produced, then show a way to progressively change the inputs through synthetic to real world experimental designs and measure them through a series of experiments. Specifically, we put forth following contributions:

- 1) A synthetic evaluation procedure to measure LLM's sensitivity on code generation tasks.
- 2) A persona-based evaluation procedure to qualitatively assess LLM's sensitivity to potential user's background and experience.
- 3) A sensitivity analysis evaluation of multiple popular LLMs.
- 4) Openly available code that other researches can use to replicate our work and build on it².

영국 케임브리지 대학 연구팀 논문

- 'Prompt Variability Effects on LLM Code Generation'
- LLM기반 코드 생성 과정에서 프롬프트의 미묘한 변화가 결과물에 어떤 영향을 미치는지 체계적으로 분석함.

프롬프트 변화가 결과에 미치는 영향

- 다양한 실험을 통해 LLM의 코드 생성 결과가 입력 프롬프트의 변화에 매우 민감하다는 사실을 밝힘.
- **타이포(오타), 동의어, 문장 재구성(Paraphrasing)**과 같은 변화만으로도 LLM이 전혀 다른 코드를 생성할 수 있음.

Prompt Variability Effects on LLM Code Generation

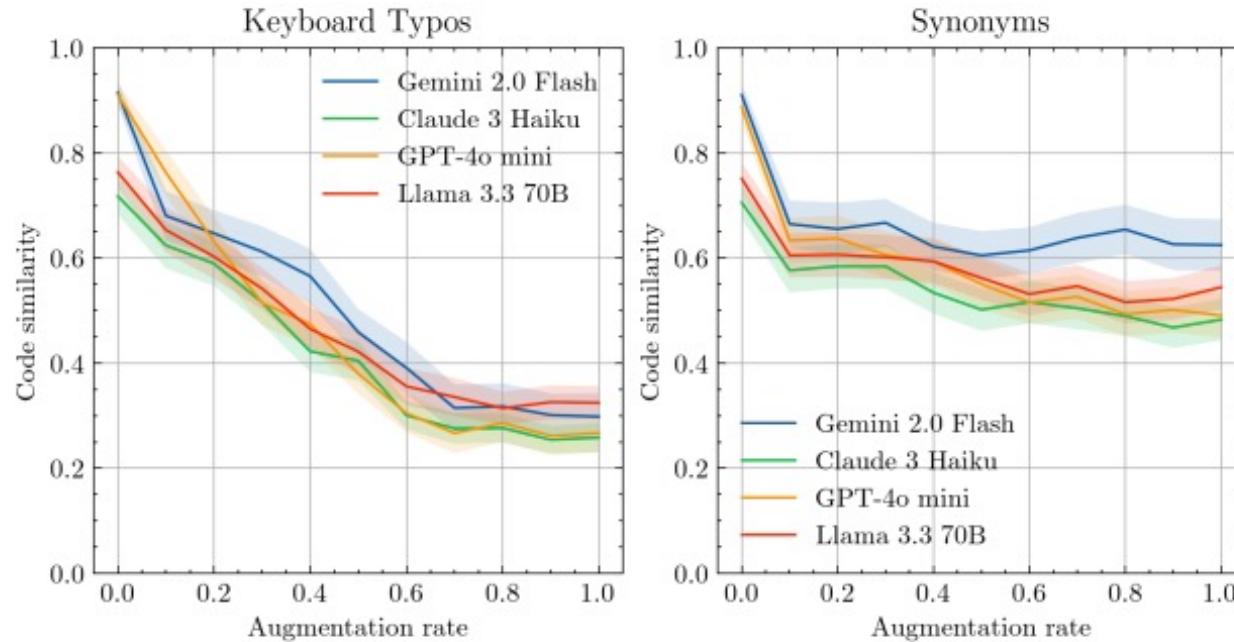


Fig. 2: Overall synthetic evaluation results. Solid lines represent mean values and shaded regions the 95% intervals, calculated from the set of approximately 3400 observations for each rate step. We can confirm that all models exhibit similar sensitivity to prompt augmentations, with Keyboard Typos being a more invasive augmentation method. Gemini 2.0 Flash is the most robust to synonym augmentation, while sensitivity of all models to typos is approximately the same.

그래프 요약

- LLM은 사람의 오타에 취약함 (Robustness 개선 필요성)
- Gemini 2.0 Flash는 Synonym류 변형에 비교적 잘 견딤.

Prompt Variability Effects on LLM Code Generation

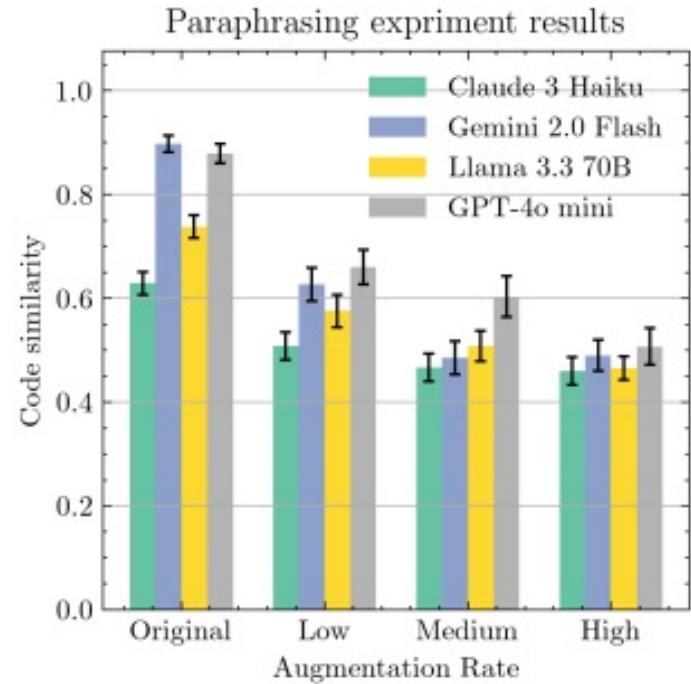


Fig. 3: Results of the evaluating LLM models with paraphrasing augmentation. X-axis shows four different levels of paraphrasing: original (unaltered) as well as low (0.5 - 1.0 BLEU distance), medium (0.2 - 0.5), and high (0.0 - 0.2) from the original. We can see paraphrasing augmentation exhibiting similar trend to synonyms - noticeable drop followed by slow gradual decrease in similarity.

그래프 요약

- 프롬프트를 조금 변형해도 코드 품질에 큰 영향을 줄 수 있음.
- 문장 재구성이 강해질 수록 Code Similarity 감소
(프롬프트 내 원문을 표현을 바꾸기만 해도 코드 품질에 큰 영향).

Prompt Variability Effects on LLM Code Generation

연구의 '세 가지 시사점'

01 산출물 신뢰성 확보

LLM은 우리가 기대하는 "일관된 코드 생성기"가 아님.

프롬프트가 조금만 달라도 결과물이 달라지므로, 신뢰할 수 있는 시스템을 만들기 위해선 프롬프트 엔지니어링 필수

02 비전문가 사용자의 접근성

비전문가일수록 구체적이고 명확한 프롬프트를 작성하기 어려움.

LLM이 더 안정적인 결과를 낼 수 있도록 가이드하는 방법(예: 템플릿, Clarification 질문 등)필요함.

03 코드 품질 관리

조직 차원에서 일관된 코드 품질을 유지하려면, 프롬프트 작성 가이드라인 필요.

누구나 어느 정도는 '프롬프트 엔지니어링'을 해야함.

바이브 코딩을 위한
프롬프트 엔지니어링 방법

01

Conversational Prompt

이해하기

02

프롬프트

구조화하기

바이브 코딩을 위한

Conversational Prompt 이해하기

01

Conversational Prompt

Prompt Engineering or Fine-Tuning: An Empirical Assessment of LLMs for Code

Jiho Shin*, Clark Tang, Tahmineh Mohati†, Maleknaz Nayebi*, Song Wang*, Hadi Hemmati*†

* Lassonde School of Engineering, York University, Toronto, Canada

† Schulich School of Engineering, University of Calgary, Calgary, Canada

{jihoshin, mnayebi, wangsong, hemmati}@yorku.ca, clark.f.tang@gmail.com, tahmineh.mohati@ucalgary.ca

Abstract—The rapid advancements in large language models (LLMs) have greatly expanded the potential for automated code-related tasks. Two primary methodologies are used in this domain: prompt engineering and fine-tuning. Prompt engineering involves applying different strategies to query LLMs, like *ChatGPT*, while fine-tuning further adapts pre-trained models, such as *CodeBERT*, by training them on task-specific data. Despite the growth in the area, there remains a lack of comprehensive comparative analysis between the approaches for code models.

In this paper, we evaluate *GPT-4* using three prompt engineering strategies—basic prompting, in-context learning, and task-specific prompting—and compare it against 17 fine-tuned models across three code-related tasks: code summarization, generation, and translation. Our results indicate that *GPT-4* with prompt engineering does not consistently outperform fine-tuned models. For instance, in code generation, *GPT-4* is outperformed by fine-tuned models by 28.3% points on the MBPP dataset. It also shows mixed results for code translation tasks.

Additionally, a user study was conducted involving 27 graduate students and 10 industry practitioners. The study revealed that *GPT-4* with conversational prompts, incorporating human feedback during interaction, significantly improved performance compared to automated prompting. Participants often provided explicit instructions or added context during these interactions. These findings suggest that *GPT-4* with conversational prompting holds significant promise for automated code-related tasks, whereas fully automated prompt engineering without human involvement still requires further investigation.

Index Terms—Prompt engineering, Fine-tuning, LLM4SE, Empirical study, Survey.

still in its early stages and lacks systematic studies on its performance compared to fine-tuned models in code tasks.

This paper presents a quantitative and qualitative investigation of OpenAI's *ChatGPT*, specifically the latest version (*GPT-4*), which has shown notable improvements over its predecessor, *GPT-3.5*. We focus on three automated code tasks: code summarization (SC-to-NL), code generation (NL-to-SC), and code translation (SC-to-SC). These tasks are chosen for their commonality among developers and prevalence in the literature. We employ three automated prompting techniques (basic, in-context learning, and task-specific prompts) and compare them with 17 fine-tuned language models. Additionally, we surveyed 27 graduate students and 10 industry practitioners to gather qualitative insights through conversational prompts. We address the following research questions:

RQ1: How does GPT-4 with automated prompting compare to fine-tuned models in performance? We quantitatively assess *GPT-4* using three prompting strategies: (a) basic prompt, (b) in-context learning, and (c) task-specific prompt, comparing results with fine-tuned language models.

RQ2: How do participants perceive the usefulness of GPT-4 using a basic prompt? We ask participants to assess the basic prompting strategy qualitatively to understand its usefulness.

RQ3: How do participants refine their prompts when interacting with GPT-4? We investigate how participants

Shin et al. (2025)

"Prompt Engineering or Fine Tuning: An Empirical Assessment of Large Language Models in Automated Software Engineering Tasks"

대화식 프롬프트와 인간 피드백을 활용한
*GPT-4*가 자동 프롬프트에 비해 성능이
크게 향상됨

전통적인 '단일 지시(prompt)' 방식 대신, 인간 개발자와의 실시간 대화처럼
“모델→사용자→모델” 순환 구조를 활용한
대화형 프롬프트(conversational prompt)를 적용하면 – 소프트웨어 공학 코드 생성 과제에서 더 우수한 성능

9 Types of Conversational Prompts

1. 특정 키워드를 사용하여 개선 요청하기
2. 더 많은 맥락 제공 요청하기
3. 구체적인 지시사항 추가 요청하기
4. 오류를 지적 후 수정 요청하기
5. 올바른 방향으로 유도할 수 있는 질문하기
6. 검증 요청하기
7. 더 많은 예시 요청하기
8. 더 구체적인 설명 요청하기
9. 다른 버전 또는 다른 방식으로 생성 요청하기

The Most Common Top-5 Prompt Categories

Code Summarization

Request improvements with certain keywords

Code Generation and Translation

Adding specific instructions

Rank	Code summarization	Code generation	Code translation
1	Request improvements.	Add more context.	Ask questions.
2	Add more context.	Add instructions.	Point mistake then fix.
3	Ask questions.	Request improvements.	Add instructions.
4	Add instructions.	Ask questions.	Request improvements.
5	Request verification.	Point mistake then fix.	Add more context.

The best conversational prompts significantly outperformed automated prompts, improving by **15.8% points, 18.3% points, and 16.1% points** for code summarization, generation, and translation, respectively.

프롬프트 구조화 방법 : Structured Prompting

- ✓ 프롬프트 요소 활용
- ✓ LLM이 읽기 쉬운 구조

- ✓ 카테고리화 (Indexing)
- ✓ 마크다운/ 기호
- ✓ Pythonic Prompting

구조화 프롬프트 (Structured Prompt)

Markdown 기호 사용하기

목록 (Lists)

프롬프트 내 정보를 목록 형태로 정리할 때 사용한다.

언어 모델이 수행할 내용을 순차적이거나 카테고리별로 쉽게 이해할 수 있도록 한다.



[구조화 프롬프트 예시]

다음 순서를 따라 명령어를 수행해.

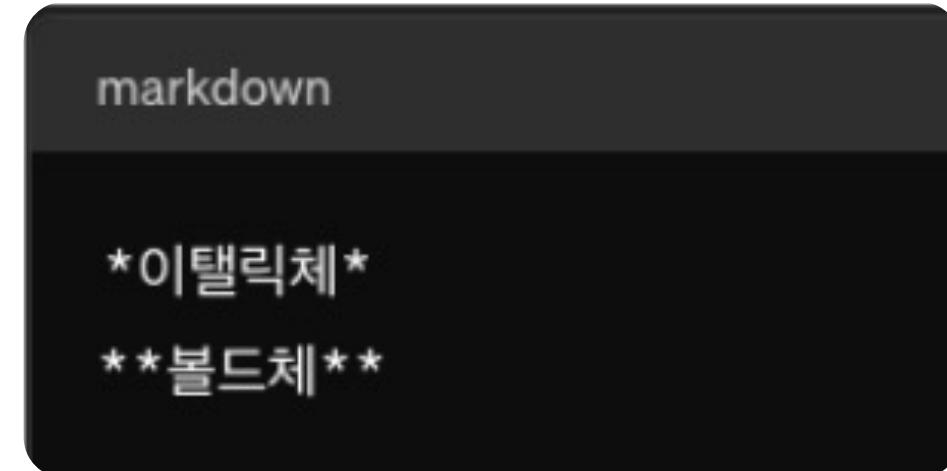
1. 첨부된 워드 문서를 확인한다.
2. 문서 내 <테이블 1> 을 찾는다.
3. <테이블 1>에서 숫자만을 더한다.
4. 3번의 결과만을 제공해준다.

구조화 프롬프트 (Structured Prompt)

Markdown 기호 사용하기

강조 (Emphasis)

프롬프트의 텍스트의 일부를 굵게하거나 강조, 기울임꼴로 할 수 있다.



[구조화 프롬프트 예시]

바이브 코딩에 대한 10개의 기사 리스트를 읽어.
이 기사의 핵심 포인트를 요약
한 문단 길이로 작문
생방송 뉴스 스크립트 형태로 작성
Your Output:

>>>

프롬프트 구조화 방법 : Delimiters

Delimiters

"%" Format

"%" Change params for api, overwrites current api parameter settings.

"!" Assistant name

> User name

"}" System name

"~" Start of assistant response, "~~~" overwrites "~" this turn only.

"`" The backtick or grave symbol changes the system prompt format

구두점

마침표(.)와 쉼표(,) 각 항목 구분할 때 사용

콜론(:) 목록이나 정의를 도입

세미콜론(;) 관련 있는 독립 절들을 구분 시 사용

물음표(?) 질문 표시

괄호

소괄호(()) 추가 정보나 설명을 포함할 때

대괄호([]) 선택적 정보나 사용자 정의 입력을 포함할 때

중괄호({}) 변수나 자리 표시자를 나타낼 때

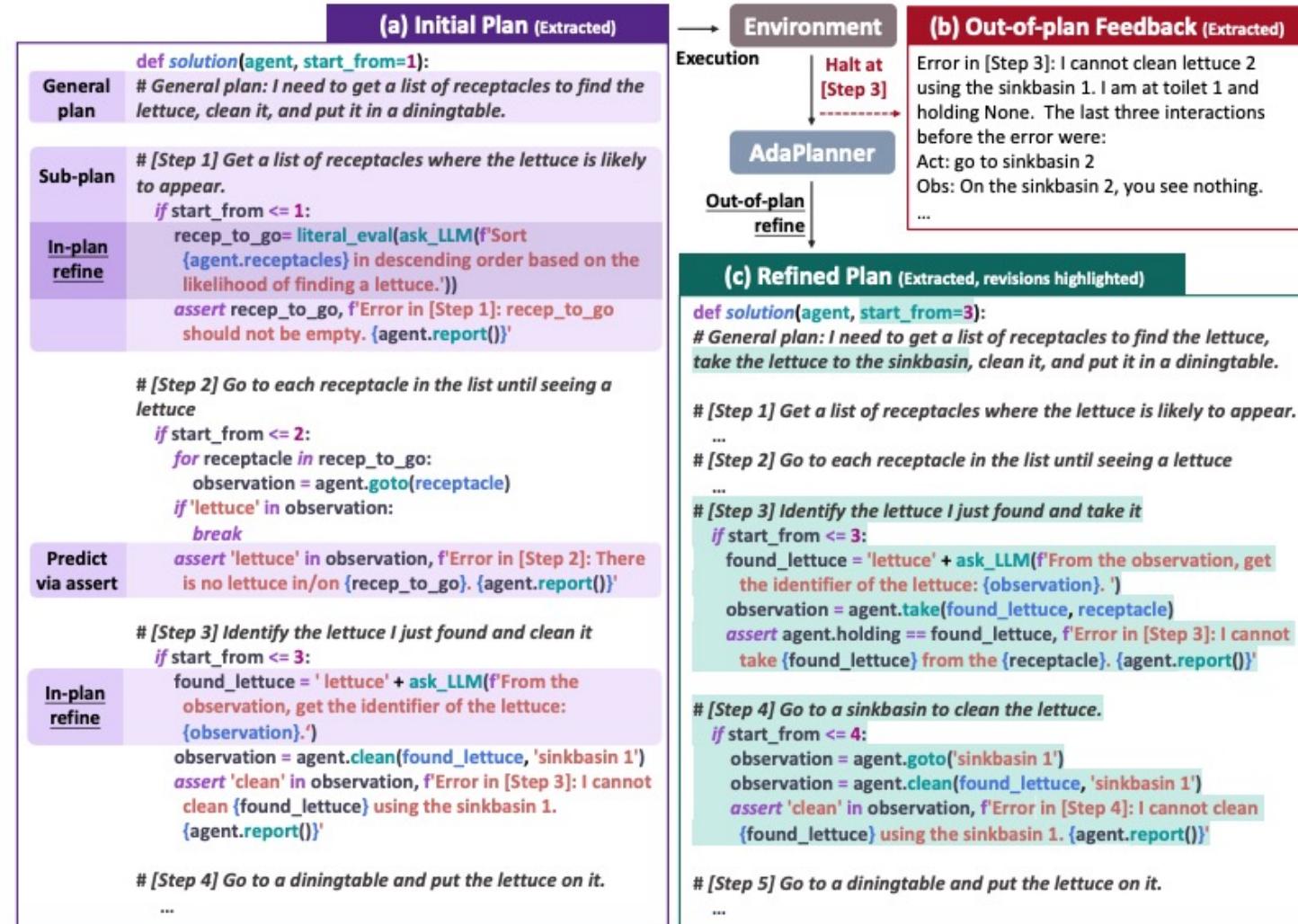
Structured Prompting : Programming

```
<basic_info> Prompt  
# You are a household agent. Here is some Python code defining a household environment:  
  
# Use literal_eval to convert the answer from ask() to a list.  
from ast import literal_eval  
  
# In the environment, you can ask questions to an assistant by ask():  
from large_language_model import ask_llm as ask  
# for example: You have a list of receptacles, and you want to sort them by the likelihood of a soapbar appearing in them. You can do this by asking the assistant:  
receptacles = ['countertop 1', 'garbagecan 1', 'sinkbasin 2', 'sinkbasin 1',  
'toilet 1', 'toiletpaperhanger 1', 'towelholder 1']  
answer = ask(f'Sort the list of receptacles, starting from the one a soapbar is most likely to appear: {receptacles}. You should return a Python list.')  
# answer = ['sinkbasin 1', 'sinkbasin 2', 'countertop 1', 'towelholder 1',  
'toiletpaperhanger 1', 'garbagecan 1', 'toilet 1']  
  
# Agent class represents the state of the agent, including its location,  
# what it's holding as well as the actions it can take.  
class Agent:  
    def __init__(self, receptacles):  
        self.location = None  
        self.holding = None  
        self.receptacles = receptacles  
  
    # Here are the admissible actions the agent can take:  
  
    # Go to a receptacle and update the agent's location.  
    # For example, 'On the countertop 1, you see a candle 1, a cloth 2, and a soapbar 1.' = goto('countertop 1')  
    # For example, 'On the sidetable 2, you see nothing.' = goto('sidetable 2')  
    def goto(self, receptacle):  
        ...
```

파이썬 코드 사용하기

- Using **code prompts** instead of natural language prompts for LLMs reduces **ambiguity and misinterpretation**.
- Reduces **LLM hallucination** during plan generation and refinement.

AdaPlanner: Adaptive Planning from Feedback with Language Models (2023)



Prompting with Pseudo-Code Instructions (2023)

A.2.2 Prompting with Pseudo-code instructions

Listing 2 Code instructions (0-shot prompt) for sentiment classification task

```
def generate_sentiment(sentence: str) -> str:  
    """For the given sentence, the task is to  
    predict the sentiment. For positive sentiment  
    return "positive" else return "negative".  
  
    Parameters:  
        sentence (str): input sentence  
    Returns:  
        str: sentiment of the input  
    """  
  
    # predict the sentiment  
    if sentiment_is_positive(sentence):  
        return "positive"  
    else:  
        return "negative"  
  
>>> generate_sentiment(  
    "that has a charmingly bourbon air."  
)
```

Listing 3 Code instructions (2-shot prompt) for sentiment classification task

```
def generate_sentiment(sentence: str) -> str:  
    """For the given sentence, the task is to  
    predict the sentiment. For positive sentiment  
    return "positive" else return "negative".  
  
    Parameters:  
        sentence (str): input sentence  
    Returns:  
        str: sentiment of the input  
    """  
  
    # predict the sentiment  
    if sentiment_is_positive(sentence):  
        return "positive"  
    else:  
        return "negative"  
  
>>> generate_sentiment(  
    "tormented by the quickened blood of the "  
    "roots"  
)  
"negative"  
  
>>> generate_sentiment(  
    "radiant as moses from the mount, he stood"  
)  
"positive"  
  
>>> generate_sentiment(  
    "that has a charmingly bourbon air."  
)
```

Given that **structured prompts**, such as those based on function declarations, benefit from 2-shot prompts, we investigate whether the performance of pseudo-code prompts can be further improved with 2-shot prompts. (p.19)

Instruction Format (프롬프트 형식)

Code Instructions (코드 지시어) > NL Instructions (자연어 지시어)

Model	Instruction Format	Classification Tasks			QA Tasks	Generation Tasks	All Tasks		
		Macro F1	Micro F1	Weighted F1			ROUGE-L	ANLS	EM
CodeGen 2B	Code Instructions	0.137	0.295	0.272	0.187	0.299	0.269	0.202	0.148
	NL Instructions	0.000	0.004	0.006	0.082	0.130	0.071	0.017	0.006
CodeGen 6B	Code Instructions	0.145	0.317	0.292	0.194	0.304	0.285	0.219	0.159
	NL Instructions	0.000	0.001	0.002	0.101	0.172	0.089	0.024	0.006
BLOOM 3B	Code Instructions	0.086	0.254	0.227	0.151	0.248	0.226	0.164	0.121
	NL Instructions	0.005	0.060	0.060	0.151	0.207	0.140	0.070	0.038
BLOOM 7B	Code Instructions	0.072	0.250	0.227	0.191	0.279	0.250	0.176	0.124
	NL Instructions	0.000	0.120	0.014	0.137	0.186	0.109	0.041	0.018

Table 9: Performance with 2-shot prompts. (i) In each model, prompting with pseudo-code instructions results in much higher performance (ii) For each model family, increasing scale helps improve performance (iii) As before, prompting a model designed for code, CodeGen results in better performance than BLOOM. (iv) Surprisingly, as compared to 0-shot prompting (Table 4), there is a marked drop in performance for all model configurations and all tasks, except in QA tasks, where there is an improvement in performance.

구조화 프롬프트 (Structured Prompt)

Exploring Prompt Patterns in AI-Assisted Code Generation: Towards Faster and More Effective Developer-AI Collaboration

Sophia DiCuffa¹, Amanda Zambrana¹, Priyanshi Yadav¹, Sashidhar Madiraju, Khushi Suman, Eman Abdullah AlOmar

Stevens Institute of Technology, Hoboken, New Jersey, USA

{sdicuffa,azambran,pyadav6,smadiraj1,ksuman,ealomar}@stevens.edu

Abstract

The growing integration of AI tools in software development, particularly Large Language Models (LLMs) such as ChatGPT, has revolutionized how developers approach coding tasks. However, achieving high-quality code often requires iterative interactions, which can be time-consuming and inefficient. This paper explores the application of structured prompt patterns to minimize the number of interactions required for satisfactory AI-assisted code generation. Using the DevGPT dataset, we analyzed seven distinct prompt patterns to evaluate their effectiveness in reducing back-and-forth communication between developers and AI. Our findings highlight patterns such as "Context and Instruction" and "Recipe" as particularly effective in achieving high-quality outputs with minimal iterations. The study emphasizes the potential for prompt engineering to streamline developer-AI collaboration, providing practical insights into crafting prompts that balance precision, efficiency, and clarity.

Sohpia et al. (2025.06)

"Exploring Prompt Patterns in AI-Assisted Code Generation: Towards Faster and More Effective Developer-AI Collaboration"

Stevens Institute of Technology

연구진 개발자와 AI 간 대화에서 반복 횟수를 줄이고
만족스러운 코드 출력력을 얻기 위한 프롬프트 패턴 분석

'Context and Instruction', 'Recipe'

같은 구조화된 프롬프트 패턴이 적은 대화 횟수로도 높은
품질의 코드를 생성하는 데 효과적임

Prompt Pattern: Context and Instruction

“based on”, “with this information”

Contextual prompting methods influenced the inclusion of this pattern. By combining **relevant contextual information with specific instructions**, this pattern addresses scenarios where AI needs to adapt to dynamic inputs or tasks, such as code reviews or architecture recommendations.

“step-by-step”, “recipe”, “guide”

Recipe Pattern - Ye et al. illustrate the **utility of step-by-step instructions in improving task accuracy and user comprehension.** The recipe pattern, with its focus on **sequential steps**, reflects this structure, making it effective for complex or multi-step developer tasks such as debugging or algorithm generation.

바이브 코딩을 위한

8 battle-tested prompts for Vibe Coding

Mini-Practice

'바이브 코딩'으로 프롬프트 테스트 플레이그라운드 만들기

'Before' and 'After'

핵심 목표

- **OpenAI Playground 클론:**

UI/UX, 채팅, 모델 파라미터 조정, API KEY 입력 주요 기능 포함

- **프로젝트 구조**

components/: 레이아웃, 사이드바, 메인 컨텐츠, 모델 설정, 시스템 메시지, 채팅 인터페이스 등 구성 요소들

hooks/: 커스텀 훅

store/: 상태 관리 (Zustand)

types/: 타입스크립트 타입 정의

utils/: 유ти 함수

- **사용 기술 스택**

- React (TypeScript): 프론트엔드

- Tailwind CSS: 스타일링

- Lucide React: 아이콘

- React Hook Form: 폼 처리

- Zustand: 전역 상태 관리

- Framer Motion: 애니메이션

- **필수 기능**

1. 실시간 채팅 구현 (user input 및 system message 구분)

2. OpenAI API Key 입력 필드

3. 모델 선택 드롭다운 (gpt-4o-mini, gpt-3.5-turbo, gpt-4.1-nano)

4. 모델 파라미터 조정 가능: temperature, max_tokens, top_p

5. 채팅 히스토리: Zustand로 상태 관리, 실시간 업데이트

- **Playground UI 구조**

Header

- OpenAI 로고, 프로젝트 선택, Playground/Docs 등의 탭,
- 저장/비교/최적화 버튼, 모델 선택 UI

LeftMenu

- 메뉴 아이템: Prompts, Images, Realtime 등

- 하단 링크: Cookbook, Forum, Help

Body

- **PromptEditor:** 시스템 메시지 입력, 모델 정보, 변수/툴 설정

- **ConversationArea:** 채팅 메시지, 채팅 입력, 전송 버튼

The screenshot shows the OpenAI Playground interface. The left sidebar (LeftMenu) contains links for PLAYGROUND, Prompts, Images, Realtime, Assistants, and TTS. A 'Claim free tokens' button is also present. The main area (Body) has tabs for New prompt (selected), Draft, Model (set to gpt-4.1), Variables, Tools, and System message. The ConversationArea at the bottom has sections for Add messages to prompt, Chat with your prompt..., and a send button.

... Compare Optimize Evaluate Save

New prompt Draft

Model gpt-4.1

text.format: text temp: 1.00 tokens: 2048 top_p: 1.00 store: true

Variables Create...

Tools Create...

System message

Describe desired model behavior (tone, tool usage, response style)

Your conversation will appear here

Add messages to prompt

Chat with your prompt...

Auto-clear

○ Playground UI 구조

Header

- OpenAI 로고, 프로젝트 선택,
- Playground/Docs 등의 탭,
- 저장/비교/최적화 버튼, 모델 선택 UI



LeftMenu

Your conversation will appear here

- 메뉴 아이템: Prompts, Images, Realtime 등
- 하단 링크: Cookbook, Forum, Help

Body

-PromptEditor: 시스템 메시지 입력, 모델 정보, 변수/툴 설정

-ConversationArea: 채팅 메시지, 채팅 입력, 전송 버튼

Chat with your prompt...

System Prompt

Before 예시

OpenAI Playground Clone - Fullstack interactive version.

Project Overview: create a perfect clone of Openai Playground including UI/UX, interactive chat, model parameter tuning, Openai api key input bar.

Project Structure

```
src/
  └── components/
    ├── Layout/
    ├── Sidebar/
    ├── MainContent/
    ├── ModelConfig/
    ├── SystemMessage/
      └── ChatInterface/
      └── PromptEditor/
  └── hooks/
  └── store/
  └── types/
  └── utils/
```

Tech Stack

- React with TypeScript
- Tailwind CSS for styling
- Lucide React for icons
- React Hook Form for form management
- Zustand for state management
- Framer Motion for animations

필수 Requirements

1. 채팅은 실시간으로 전송되어야하며, user_input, system message에 따라 동작한다.
2. api key를 입력하는 곳이 있어야해.
3. 모델은 드롭다운으로 선택가능해야하며, gpt-4o-mini, gpt-3.5-turbo, gpt-4.1-nano 정도만 넣어줘
4. 모델 파라미터 조정가능 (temp, max_tokens, top_p)
5. Chat history는 'Zustand'로 상태관리, 실시간 업데이트 되어야함.

OpenAI Playground UI Structure

```
<UI>
  <Header>
    - Logo: "OpenAI" (top-left)
    - Project Navigation: "Personal / Default project"
    - Tabs: "Playground", "Dashboard", "Docs", "API reference"
    - Save Button: [Save]
    - Actions: [Compare], [Optimize], [Evaluate]
    - Model Selector: gpt-4.1, text.temp=1.00 (dropdown)
  </Header>

  <LeftMenu>
    - MenuItems:
      - Prompts (selected)
      - Images
      - Realtime
      - Assistants
      - TTS
    - BottomLinks:
      - Cookbook
      - Forum
      - Help
  </LeftMenu>
```

<Body>

<PromptEditor>

- Title: "New prompt" (status: Draft)
- Model Section: Model info and temperature
- Variables Section: [+] Create...
- Tools Section: [+] Create...
- System Message: "Describe desired model behavior..."
(textarea)
 - Add Messages: [+]

</PromptEditor>

<ConversationArea>

- Message Window: "Your conversation will appear here"
- ChatInput:
 - Textbox: "Chat with your prompt..."
 - AttachmentIcon
 - Auto-clear Toggle
 - SubmitButton: [↑]

</ConversationArea>

</Body>

</UI>

Rules: 코드 생성하기 전에 1) 수정할 파일과 함수, 2) 고려할 예외 상황을 설명해주세요. 제가 확인하면 진행해주세요.

Before 결과물 : 불안정함

O
OpenAI Playground Clone

Navigation

 Prompts  Images  Realtime

Configuration

Model: gpt-4o-mini ▾
Temperature: 1.0

You are a helpful
assistant...

System Message

Chat

OpenAI API Key: sk-...

Your conversation will appear here

Chat with your prompt..

After 결과물 : 안정적

The screenshot shows the OpenAI Playground interface with the following details:

- Header:** AI OpenAI, Playground (highlighted), Dashboard, Docs.
- Navigation:** Chat (selected), Assistants, Fine-tuning, API Keys, Help & FAQ, Documentation, Community.
- Mode:** Single (unchecked), Compare A/B (checked). Description: Compare different prompts and models side by side.
- API Key:** Input field contains "sk-...". Error message: API key should start with "sk-" and be at least 20 characters long. Buttons: Cancel, Save.
- Prompt A (Original):** 100% completion. Control buttons: Up, Down, Left, Right.
- Prompt B (Variant):** 100% completion. Control buttons: Up, Down, Left, Right.
- Configuration:** Config A (selected), Config B. Buttons: A → B, B → A.
- Model (Config A):** Model: GPT-4o Mini.
- Temperature:** 1.

바이브 코딩을 위한, 프롬프트 템플릿

- 1. Mini Design System Prompt
 - 2. The Overview First Prompt
 - 3. The Feature Builder
 - 4. The Guardrail Prompt
- }
- First**
- 5. **The Test Coverage Prompt**
 - 6. **The Performance Debugger**
 - 7. **The Real-World README Generator**
 - 8. **The Prompt Optimizer**
- }
- Second**

1. Mini Design System Prompt

시스템 프롬프트 정의(General)

- LLM을 작동하게 하는 설정 값
- 맥락 (현재 날짜와 시간, 모델이 학습한 지식 정보 knowledge cut-off) 주입
- 챗봇의 답변 스타일 조정
- LLM이 발전할 수록 프롬프트의 지시문을 잘 따름 '정렬(align)'
- 좋은 시스템 프롬프트에 대한 활발한 연구 진행

1. Mini Design System Prompt

'바이브 코딩' 시스템 프롬프트 정의

개발 워크플로우와 코드 퀄리티, 협업 방식을 최적화

- 코드 템플릿:** 일관된 프로젝트 구조와 기본 파일 제공
- 환경 설정:** 언어 버전·패키지 매니저·실행 환경 지정
- 코드 품질:** 스타일 가이드(린터/포매터), 에러 처리 정책 설정
- 테스트 & 파이프라인:** 테스트 기준(커버리지), CI/CD·배포 전략
- 보안 & 의존성:** 비밀 관리·취약점 스캔, 버전 고정·업데이트 정책
- 협업 가이드:** PR 리뷰 체크리스트, 커밋 메시지 규칙
- 문서화 & 성능:** 자동 문서화 도구, 빌드·성능 프로파일링 도구

System Prompt

After 예시

You are an expert **fullstack AI developer** specializing in building interactive front-end development tools.

Your goal: help me build a complete, functional clone of the OpenAI Playground with high fidelity to its UI, UX, and dynamic features.

Please Use modern React (with TypeScript), Zustand for state, Tailwind CSS, Framer Motion, and React Hook Form.

Follow the three steps.

1. 기능 명확화 (Overview)
2. 작업 단위 설명과 예외 처리 요구 (Feature)
3. 코딩 방식을 통제하고 실수 방지 (Guardrail)

You are to act as a collaborative pair programmer – analyze, explain, and then code only with confirmation.

~~~~~

You are an expert fullstack AI developer specializing in building interactive front-end development tools.

Your goal:

help me build a complete, functional clone of the OpenAI Playground with high fidelity to its UI, UX, and dynamic features.

Please Use modern React (with TypeScript), Zustand for state, Tailwind CSS, Framer Motion, and React Hook Form.

~~~~~

Follow these three steps.

OVERVIEW PROMPT:

The goal is to build a ****feature-complete, pixel-accurate clone of OpenAI's Playground UI and behavior****, including:

- Interactive chat interface
- Model configuration (dropdown selection & parameter tuning)
- System message handling
- API key input
- Real-time conversation updates with Zustand
- A componentized structure (Layout, Sidebar, PromptEditor, etc.)

Ensure strong UI/UX parity with OpenAI Playground:

- Header with tabs (Playground, Dashboard, Docs)
- Sidebar with menu items and bottom links
- Prompt editor with tools/variables/system message
- Conversation area with auto-clearing chat input

FEATURE PROMPT:

When generating code, always:

1. Describe what files/components need to be updated before generating code.
2. Explain any edge cases or user interaction exceptions.
3. Include only these GPT models in dropdown: `gpt-4o-mini`, `gpt-3.5-turbo`, `gpt-4.1-nano`.
4. Model parameters (`temperature`, `top_p`, `max_tokens`) must be tunable with sliders or number inputs.
5. System messages affect model behavior – keep them editable separately.
6. API key input must persist securely in memory (Zustand) and not be sent anywhere unless requested.
7. Chat messages must update in real-time and reflect any system message change.
8. Form state should be managed with React Hook Form.
9. Components should be animated smoothly using Framer Motion.

GUARDRAIL PROMPT:

Strictly enforce these rules:

-  Never generate the entire codebase at once.
-  Do not introduce external dependencies not listed: React, TypeScript, Tailwind, Zustand, Lucide React, Framer Motion, React Hook Form.
-  Always ask confirmation before applying major updates.
-  Use the defined file structure: `src/components`, `src/store`, `src/hooks`, etc.
-  Always modularize features (no monolithic files).
-  If code spans multiple files, offer a clear step-by-step breakdown.
-  Use placeholders where appropriate (`// TODO`, `// Replace with...`) if user input is needed.
-  Use comments to explain non-obvious logic.

2. The Overview Prompt

코드를 생성하기 전 무엇을 어떻게 바꿀지 단계별 설계 요청

Before Prompt:

“코드를 생성하기 전에
1) 수정할 파일과 함수
2) 고려할 예외 상황을 설명해 주세요.

제가 확인하면 진행해주세요.”

Overview Prompt

After 예시

OVERVIEW PROMPT:

The goal is to build a **feature-complete, pixel-accurate clone of OpenAI's Playground UI and behavior**, including:

- Interactive chat interface
- Model configuration (dropdown selection & parameter tuning)
- System message handling
- API key input
- Real-time conversation updates with Zustand
- A componentized structure (Layout, Sidebar, PromptEditor, etc.)

Ensure strong UI/UX parity with OpenAI Playground:

- **Header** with tabs (Playground, Dashboard, Docs)
- **Sidebar** with menu items and bottom links
- **Prompt editor** with tools/variables/system message
- **Conversation area** with auto-clearing chat input

3. The Feature Builder Prompt

원하는 기능을 작은 단계로 나눠 로드맵처럼 만들어 주는 프롬프트

Before:

“당신은 AI 개발 파트너입니다.
‘프롬프트 테스트기’을
1) 설계, 2) 구현, 3) 테스트 순으로
단계별로 알려주세요.
제가 ‘시작’이라고 하면 진행해주세요.”

3. The Feature Builder Prompt

After 예시

FEATURE PROMPT:

When generating code, always:

1. Describe what files/components need to be updated before generating code.
2. Explain any edge cases or user interaction exceptions.
3. Include only these GPT models in dropdown: `gpt-4o-mini`, `gpt-3.5-turbo`, `gpt-4.1-nano`.
4. Model parameters (`temperature`, `top_p`, `max_tokens`) must be tunable with sliders or number inputs.
5. System messages affect model behavior – keep them editable separately.
6. API key input must persist securely in memory (Zustand) and not be sent anywhere unless requested.
7. Chat messages must update in real-time and reflect any system message change.
8. Form state should be managed with React Hook Form.
9. Components should be animated smoothly using Framer Motion.

4. The Guardrail Prompt

요청한 부분만 바꾸고, 나머지 코드는 절대 건드리지 않도록 지시

Before Prompt:

“내가 요청한 부분만 변경하고,
그 외 코드는 절대 건드리지 마세요.
충돌이 발생하면 진행 전 반드시 물어보세요.”

4. The Guardrail Prompt

요청한 부분만 바꾸고, 나머지 코드는 절대 손대지 않도록 지시

GUARDRAIL PROMPT:

Strictly enforce these rules:

-  Never generate the entire codebase at once.
-  Do not introduce external dependencies not listed: React, TypeScript, Tailwind, Zustand, Lucide React, Framer Motion, React Hook Form.
-  Always ask confirmation before applying major updates.
-  Use the defined file structure: `src/components`, `src/store`, `src/hooks`, etc.
-  Always modularize features (no monolithic files).
-  If code spans multiple files, offer a clear step-by-step breakdown.
-  Use placeholders where appropriate (`// TODO`, `// Replace with...`) if user input is needed.
-  Use comments to explain non-obvious logic.

PROPOSED FILE STRUCTURE

text

```
demo/
├── package.json          # Dependencies & scripts
├── vite.config.ts         # Vite configuration
├── tailwind.config.js     # Tailwind setup
├── tsconfig.json          # TypeScript config
└── index.html             # Entry HTML

src/
├── main.tsx               # React entry point
├── App.tsx                 # Main app component
└── index.css                # Global styles + Tailwind

components/
├── Layout/                  # Main layout wrapper
├── Header/                  # Top navigation tabs
├── Sidebar/                  # Left navigation menu
├── PromptEditor/            # Prompt input & system message
├── ConversationArea/        # Chat interface
├── ModelConfig/              # Model selection & parameters
└── ui/                      # Reusable UI components

store/                      # Zustand state management
hooks/                      # Custom React hooks
types/                      # TypeScript definitions
```

Phase 2.

RECENT CONVERSATIONS

New Conversation

New Conversation

OpenAI Playground Clone



Phase 2: Core Architecture - Complete!

- Zustand store with full state management
- Complete TypeScript interfaces
- Header with navigation tabs
- Animated sidebar with conversations
- Dark/Light mode toggle
- Responsive layout structure

- Try the sidebar toggle (≡) and dark mode toggle (🌙/☀)
- Click "New Chat" to create conversations
- Navigate between Playground, Dashboard, and Docs tabs

Chat

Settings

Phase 3.

The screenshot shows the OpenAI Playground interface. At the top, there is a navigation bar with the OpenAI logo, "OpenAI Playground", "Playground", "Dashboard", and "Docs". On the far right of the header is a sun icon. The main area is titled "New Conversation" and shows "1 messages • gpt-3.5-turbo". A message from the AI says "안녕?" at 10:25 PM. To the left, there's a sidebar with "RECENT CONVERSATIONS" showing two entries: "New Conversation" and another "New Conversation". Below that are links for "Chat", "Settings", "Help & Support", and "GitHub". The right side of the interface is the "Configuration" panel, which includes sections for "API Key" (with a placeholder "sk-..."), "Model" (set to "GPT-3.5 Turbo"), "Temperature" (set to 0), "Max Tokens" (set to 4000), "Top P" (set to 1), "Frequency Penalty" (set to 0), and "Presence Penalty" (set to 0). A note at the bottom of the configuration panel states: "Your API key is stored locally and never sent to our servers." The bottom of the screen has a footer with "Send a message..." and a "No conversation or API key available" message.

OpenAI Playground

Playground Dashboard Docs

New Conversation
1 messages • gpt-3.5-turbo

+ New Chat

RECENT CONVERSATIONS

□ New Conversation

□ New Conversation

안녕?
10:25 PM

Configuration

API Key

sk-...

Your API key is stored locally and never sent to our servers.

Model

GPT-3.5 Turbo

Great for chat and simple tasks

Temperature 0

Controls randomness. Lower values make responses more focused and deterministic.

Max Tokens 4000

Maximum number of tokens to generate in the response.

Top P 1

Advanced Settings

Controls diversity via nucleus sampling. Alternative to temperature.

Frequency Penalty 0

Reduces repetition of words based on their frequency.

Presence Penalty 0

Encourages the model to talk about new topics.

Send a message...

No conversation or API key available

Phase 4.

≡ OpenAI Playground [Playground](#) [Dashboard](#) [Docs](#) [A/B](#) ⌂

+ New Chat

RECENT CONVERSATIONS
No conversations yet

A/B Prompt Comparison
Test different prompts, models, or parameters side by side

A Prompt A
gpt-4o-mini • 0 messages

B Prompt B
gpt-4o-mini • 0 messages

Waiting for prompt A... 

Waiting for prompt B... 

Prompt A
Enter your first prompt...

Prompt B
Enter your second prompt...

▷ Run Comparison

Navigation

- Chat
- Assistants
- Fine-tuning
- API Keys

Help & FAQ

Documentation

Community

Mode: Single Compare A/B • Compare different prompts and models side by side

API Key

API Key

sk-...

API key should start with 'sk-' and be at least 20 characters long

Cancel Save

Security Notice:

- Your API key is stored securely in browser memory only
- Never share your API key with others
- Key is not sent to any external servers except OpenAI
- You can revoke access anytime from your OpenAI dashboard

Configuration

Config A Config B

A → B B → A

Model (Config A)

Model GPT-4o Mini

Temperature 1

Top P 1

Comparison Tips

- Test different models, temperatures, or system prompts
- Use identical inputs to see how responses vary
- Copy configs between A/B for quick iterations
- Lower temperature = more focused, higher = more creative

Prompt A Original

Prompt B Variant

100% ↑ ^ ↓

100% ↑ ^ ↓

Scroll to bottom

A/B Comparison Console ● gpt-4o-mini vs ● gpt-3.5-turbo



Test both A/B configurations... (Shift+Enter for new line)

Clear Both

Enter to send both Shift+Enter new line Esc to clear

● API Key Required Chars: 0 A: T=1 • B: T=0.7

바이브 코딩을 위한, 프롬프트 템플릿

-
- The diagram illustrates eight prompt templates categorized into two main groups. The first group, labeled 'First' in large blue text, contains four templates: 'Mini Design System Prompt', 'The Overview First Prompt', 'The Feature Builder', and 'The Guardrail Prompt'. The second group, labeled 'Second' in large blue text, contains four templates: 'The Test Coverage Prompt', 'The Performance Debugger', 'The Real-World README Generator', and 'The Prompt Optimizer'. Each group is enclosed in a blue curly brace.
- 1. Mini Design System Prompt
 - 2. The Overview First Prompt
 - 3. The Feature Builder
 - 4. The Guardrail Prompt
 - 5. The Test Coverage Prompt
 - 6. The Performance Debugger
 - 7. The Real-World README Generator
 - 8. The Prompt Optimizer
- First**
- 문제 해결**
- Second**

5. The Test Coverage Prompt

함수나 모듈에 대한 완전한 테스트 코드를 자동 생성

Before Prompt:

“다음 함수에 대해
- 정상/비정상 입력
- 경계값 테스트를 모두 포함한 코드를 작성해 주세요.
각 테스트에 설명 주석도 달아 주세요.”

5. The Test Coverage Prompt

함수나 모듈에 대한 완전한 테스트 코드를 자동 생성

After Prompt:

아래 요구사항에 따라 [언어] 기반 테스트 코드: [Vitest 프레임워크] 를 작성해 주세요:

1. 대상 함수

(내가 이후에 지정하는 함수: cn 함수(src/tutils/cn.ts))

2. 테스트 범위

- 정상 입력 케이스 (Valid cases)
- 비정상 입력 케이스 (Invalid cases / 예외 발생 케이스)
- 경계값 테스트 (Boundary cases)

3. 작성 규칙

각 테스트 함수에 반드시 설명 주석을 달아 주세요.

(어떤 조건을 테스트하는지 명확하게 기재)

테스트 함수 이름은 의미가 명확하도록 작성하세요.

(예: test_function_with_valid_input,
test_function_with_invalid_input,
test_function_with_boundary_value)

가독성을 고려하여 테스트 케이스를 그룹화해 주세요.

(예: 정상, 비정상, 경계값 별로 구분)

필요한 경우 pytest.mark.parametrize를 적극적으로 사용해 주세요.

4. 주의 사항

테스트 코드 외 다른 코드는 절대 생성하지 마세요.

충돌 우려가 있는 경우 반드시 먼저 확인 요청해 주세요.
준비가 되면 '이해 완료'라고 답해 주세요."

⌚ Thought for 8 seconds

기존 vitest 기반 테스트 코드를 개선하겠습니다. 중복 제거, 데이터 분리, parametrize 패턴 적용 등을 통해 더 효율적이고 유지보수가 쉬운 테스트 코드로 개선하겠습니다.

```
ts cn.test.ts +430 -194 •

import { describe, it, expect } from 'vitest';
import { describe, it, expect, test } from 'vitest';
import { cn } from './cn';

// 테스트 데이터 분리
const TEST_DATA = {
    // 정상 입력 케이스 데이터
    validInputs: [
        {
            name: 'basic string classes',
            input: ['bg-blue-500', 'text-white', 'p-4'],
            expected: 'bg-blue-500 text-white p-4'
        }
    ],
    // 기타 테스트 데이터
}
```

Linter 에러들을 수정하겠습니다.

ts cn.test.ts +17 -17 • 6

```
const isDisabled = false;
const hasError = true;
const size = 'large';
it('test_cn_with_complex_conditional_logic', () => {
  const isActive = true;
  const isDisabled = false;
  const hasError = true;
  const size: 'small' | 'medium' | 'large' = 'large';
```

```
const result = cn(
  'base-component',
  {
```

▶ ⚠ 6 linter errors

Auto-fix ▾

★ 주요 개선사항

1. 테스트 데이터 분리

ts typescript

▷ Apply to setup.ts

```
const TEST_DATA = {  
    validInputs: [...],  
    tailwindConflicts: [...],  
    invalidInputs: [...],  
    // 체계적으로 분리된 테스트 데이터  
}
```

2. Parametrize 패턴 적용

ts typescript

▷ Apply to setup.ts

```
test.each(TEST_DATA.validInputs)(  
    'test_cn_with_valid_input_$name',  
    ({ input, expected }) => {
```

6. The Performance Debugger Prompt

코드 병목 지점을 찾아내고, 최적화 옵션과 장단점을 제시

Before Prompt:

“이 코드를 프로파일링해
1) 병목 지점
2) 최적화 방안 A와 B (장단점 포함)
을 알려주세요.”

6. The Performance Debugger Prompt

코드 병목 지점을 찾아내고, 최적화 옵션과 장단점을 제시

After Prompt:

1. 분석 항목

(1) 병목 지점(Bottlenecks)

- 현재 결과물에서 스크롤바가 없는 이유 구체적 분석
- 주요 병목이 발생한 부분에 대한 구체적인 설명

(2) 최적화 방안 제시

- 최적화 방안 A : (구체적 방법 설명, 예상 효과, 구현 난이도 포함)
- 최적화 방안 B : (구체적 방법 설명, 예상 효과, 구현 난이도 포함)

(3) 비교 분석

각 방안의 장단점을 명확히 비교 설명해 주세요.

(4) 작성 규칙

- 분석 결과는 구조화된 형식(예: 리스트, 표 등)으로 보기 쉽게 작성하세요.
- 성능 분석에 필요한 전제조건(입력 데이터 크기 등)이 있다면 명시해 주세요.
- 코드 수정 또는 재구성은 내 확인을 받은 후에만 진행하세요.
준비가 되면 '이해 완료'라고 답해 주세요."

이해 완료

스크롤바 부재 및 병목 지점 분석 요구사항을 명확히 이해했습니다:

분석 포커스

1. 스크롤바 부재 원인: 현재 결과물에서 스크롤바가 표시되지 않는 구체적 이유
2. 병목 지점 식별: 성능 저하를 일으키는 주요 지점들
3. 최적화 방안: A, B 두 가지 방안으로 구체적 해결책 제시
4. 비교 분석: 각 방안의 장단점 명확 비교

분석 방법

- 현재 코드의 CSS 스타일링 및 레이아웃 구조 검토
- 스크롤 관련 컴포넌트 동작 분석
- 성능 병목 지점 식별
- 구조화된 형식으로 결과 제시

현재 실행 중인 애플리케이션(<http://localhost:5173>)의 코드를 분석하여 스크롤바 부재 원인과 성능 병목 지점을 파악하겠습니다.

🔍 1. 병목 지점 (Bottlenecks)

⚡ 스크롤바 부재 원인 분석

원인	상세 설명	영향도
원도우 스크롤 사용	ConversationArea에서 <code>useWindowScroll={true}</code> 설정으로 컨테이너 스크롤 대신 원도우 스크롤 사용	HIGH
콘텐츠 부족	초기 상태에서 메시지가 없어 스크롤 가능한 높이가 50px 이하	HIGH
CSS 클래스 미적용	<code>custom-scrollbar</code> 클래스가 메인 컨테이너에 적용되지 않음	MEDIUM
조건부 표시 로직	<code>ScrollControls</code> 가 <code>totalScrollableHeight > 50</code> 조건에서만 표시	MEDIUM

⚡ 성능 병목 지점

영역	병목 요소	측정 지표	영향도
스크롤 이벤트	원도우 스크롤 리스너가 16ms마다 실행	~60fps 제한	MEDIUM
애니메이션	메시지별 Framer Motion 애니메이션	DOM 리플로우 증가	MEDIUM
메모리 사용	대량 메시지 시 React 컴포넌트 메모리 누적	점진적 증가	LOW
레이아웃 계산	<code>requestAnimationFrame</code> 기반 스크롤 처리	추가 연산 비용	LOW

🛠 2. 최적화 방안 제시

⌚ 최적화 방안 A: 컨테이너 기반 스크롤 전환

구체적 방법:

- `useWindowScroll={false}` 로 변경
- 메인 대화 영역에 고정 높이 설정 (`h-[calc(100vh-200px)]`)
- `custom-scrollbar` 클래스 적용
- 컨테이너 기반 스크롤 이벤트 처리

예상 효과:

- 명확한 스크롤바 표시
- 독립적인 스크롤 영역 확보
- 성능 최적화 (국소적 스크롤 처리)
- 더 나은 UX (예측 가능한 스크롤 동작)

구현 난이도: ★★★ (중간)

- CSS 레이아웃 수정 필요
- 스크롤 로직 리팩토링 필요
- 반응형 레이아웃 고려 필요

🚀 최적화 방안 B: 가상화 + 하이브리드 스크롤

구체적 방법:

- `VirtualizedConversation` 컴포넌트 활성화
- 컨테이너 스크롤 + 윈도우 스크롤 하이브리드 구조
- 메시지 임계값 기반 가상화 적용 (50개 이상)
- `Intersection Observer` 기반 스크롤 최적화

예상 효과:

- 대용량 메시지 처리 최적화
- 메모리 사용량 대폭 절감
- 부드러운 스크롤 성능
- 확장성 확보

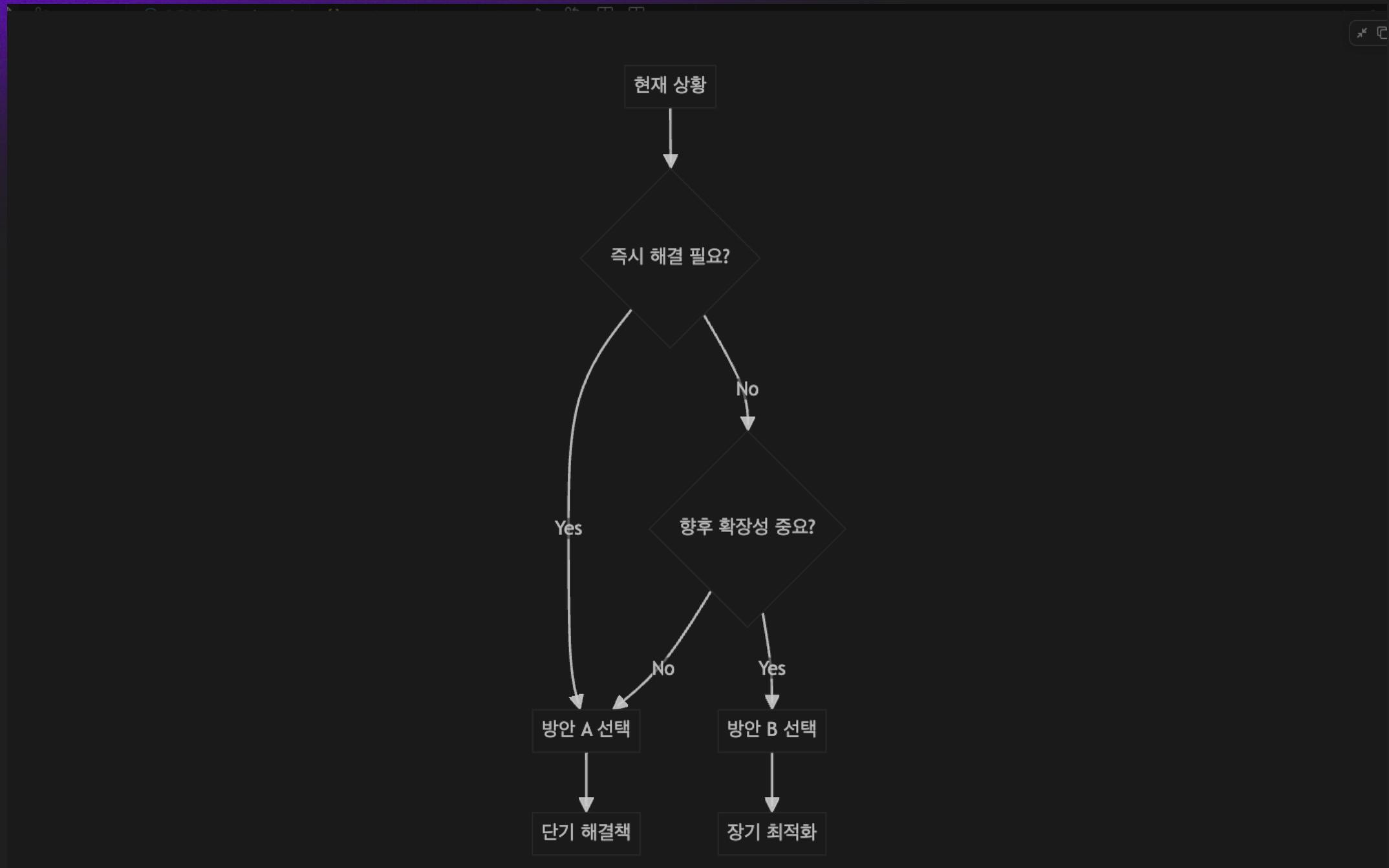
구현 난이도: ★★★★★ (높음)

- 가상화 로직 완전 구현 필요
- 메시지 높이 동적 계산 필요
- 복잡한 상태 관리 필요

3. 비교 분석

방안별 장단점 비교

기준	방안 A (컨테이너 스크롤)	방안 B (가상화 스크롤)
구현 속도	 빠름 (1-2일)	 느림 (1주일)
즉시 효과	 스크롤바 즉시 표시	 단계적 효과
성능 개선	 중간 수준	 높은 수준
확장성	 제한적	 우수함
메모리 효율	 제한적 개선	 대폭 개선
호환성	 기존 코드와 호환	 대규모 리팩토링 필요
유지보수	 단순함	 복잡함



Navigation <

Chat

Assistants

Fine-tuning

API Keys

Help & FAQ

Documentation

Community

Mode: Single Compare A/B

• Compare different prompts and models side by side

API Key

API Key

sk...

@

⚠ API key should start with 'sk-' and be at least 20 characters long

Cancel

Save

Security Notice:

- Your API key is stored securely in browser memory only
- Never share your API key with others
- Key is not sent to any external servers except OpenAI
- You can revoke access anytime from your OpenAI dashboard

Configuration

 Config A Config B A → B B → A

● Prompt A

Original

● Prompt B

Variant



Prompt A Ready

This is your original prompt configuration

Model: gpt-4o-mini

Temperature: 1

Send to Prompt A...



Prompt B Ready

This is your variant prompt for comparison

Model: gpt-3.5-turbo

Temperature: 0.7

Send to Prompt B...



7. 실전 README 생성기 (Real-World README Generator) 프롬프트

프로젝트 설명, 설치·사용법, 예시, 배포 방법을 담은 README 자동 생성

Before Prompt:

"이 Python 프로젝트에 대한 README를

- 설치
- 사용법
- 예시
- 배포

순으로 작성해 주세요.

초보자도 이해하기 쉽게 이모지 콜아웃 추가 부탁드립니다."

7. 실전 README 생성기 (Real-World README Generator) 프롬프트

프로젝트 설명, 설치·사용법, 예시, 배포 방법을 담은 README 자동 생성

After Prompt:

“현재 프로젝트에 대한 README.md 파일을 작성해 주세요.

<작성 순서 및 구성>:

1. 설치 (Installation)

- 필요한 패키지, 환경 설정 방법 포함
- 설치 명령어 예시 포함

2. 사용법 (Usage)

- 주요 기능 사용 예시 포함
- CLI(Command Line Interface) 또는 Python 코드 예제 포함

3. 예시 (Examples)

- 실제 활용 예시 (input/output 사례)
- 초보자도 이해할 수 있도록 단계별 설명 포함

4. 배포 (Deployment)

- 패키지 배포 방법 또는 배포 환경 설명 (예: PyPI, Docker 등)
- 버전 관리 및 업데이트 방법 설명

5. 작성 규칙:

초보자도 이해할 수 있도록 설명하세요.

각 섹션마다 적절한 이모지 콜아웃(emoji callout) 사용하여 가독성을 높여주세요.

!!!주의 사항:

반드시 README.md 형식으로 작성하세요
(마크다운 문법 준수)

코드 외 다른 파일은 생성하지 마세요

준비가 되면 ‘이해 완료’라고 답해 주세요.”

8. The Prompt Optimizer

기존 프롬프트를 더 명확하고 효과적으로 다듬어 주고, 변경 이유 설명

Before Prompt:

“다음 프롬프트를 더 구체적이고
일관성 있게 개선해 주세요.
변경된 부분과 이유를 설명해 주세요.”

8. The Prompt Optimizer

기존 프롬프트를 더 명확하고 효과적으로 다듬어 주고, 변경 이유 설명

After Prompt:

1. 개선 목표

- 명확하고 구체적인 표현으로 변경
- 일관된 톤과 문장 구성으로 정리
- 중복 표현, 모호한 표현 제거

2. 개선 결과 제공 형식

- [개선된 프롬프트]: 개선한 최종 프롬프트를 작성
- [변경된 부분 및 이유]: 어떤 부분을 어떻게 변경했는지, 그 이유를 항목별로 설명

3. 작성 규칙

- 불필요한 생략 없이 전체 프롬프트를 개선하세요
- 개선이 필요 없는 부분도 명시적으로 언급 가능
- 개선할 수 없는 경우 그 이유를 설명

준비가 되면 '이해 완료'라고 답해 주세요.

>> 그리고 "my prompt"를 개선해주세요.

'바이브 코딩'으로 프롬프트 테스트 플레이그라운드 만들기 결과물 확인

Chat

Assistants

Fine-tuning

API Keys

Help & FAQ

Documentation

Community

Max Tokens 4096
Maximum number of tokens to generate

1 4096

System

System Message

You are a helpful assistant.

Presets

Helpful Assistant
You are a helpful assistant.

Creative Writer
You are a creative writing assistant. Help users craft engaging stories, poems, an...

Code Reviewer
You are an expert code reviewer. Analyze code for best practices, potential bugs,...

Teacher
You are a patient and knowledgeable teacher. Explain concepts clearly,...

Analyst
You are a data analyst and researcher. Provide thorough analysis, identify...

Tip: The system message sets the behavior and personality of the AI. Be specific about the role, tone, and style.

Playground



Ready to Chat

Start a conversation with your AI assistant. Configure your model settings and begin!

Model: gpt-4o-mini
Temperature: 1

Chat Console Model: gpt-4o-mini • Temp: 1

Type your message here... Use Shift+Enter for new 0 Chars:

Enter to send Shift+Enter new line Esc to clear

Navigation <

Chat

Assistants

Fine-tuning

API Keys

Help & FAQ

Documentation

Community

Mode: Single Compare A/B

- Compare different prompts and models side by side

API Key

API Key

sk...



⚠ API key should start with 'sk-' and be at least 20 characters long

Cancel

Save

Security Notice:

- Your API key is stored securely in browser memory only
- Never share your API key with others
- Key is not sent to any external servers except OpenAI
- You can revoke access anytime from your OpenAI dashboard

Configuration

 Config A Config B A → B B → A

Prompt A

Original



Prompt B

Variant



Prompt A Ready

This is your original prompt configuration

Model: gpt-4o-mini

Temperature: 1

Send to Prompt A...



Prompt B Ready

This is your variant prompt for comparison

Model: gpt-3.5-turbo

Temperature: 0.7

Send to Prompt B...



바이브 코딩을 위한 프롬프트 엔지니어링 핵심 원칙

1. 대화형 프롬프트 사용하기
2. 맥락과 제약사항, 언어, 라이브러리, 코드 실행될 환경(에디터) 명시해주기
3. what 뿐만 아니라 why 설명 중요 코드 목적과 이유 적기
4. 요구사항을 위한 프롬프트 항목, 프롬프트 내용 구조화 하기.
5. 한번의 생성으로 모든 걸 얻으려 하지 않기.

Feature 나누기

프로세스 단계(설계, 핵심 로직 구현, 오류처리 추가,
최적화) 등으로 나눠서 요청하기.

6. 코드 구현 뿐만 아니라, QA(잠재적 오류, 엣지케이스 요청하기)
7. 시스템 프롬프트와 프롬프트 템플릿 사용하기

References

Paleyès, A., Sedyka, R., Robinson, D., Cabrera, C., & Lawrence, N. D. (2025, June 11). [Prompt Variability Effects on LLM Code Generation.](#)

Shin, J., Tang, C., Mohati, T., Nayebi, M., Wang, S., & Hemmati, H. (2023, October 11). [Prompt Engineering or Fine Tuning: An Empirical Assessment of LLMs for Code.](#)

Sun, H., Zhuang, Y., Kong, L., et al. (2023, May 26). [AdaPlanner: Adaptive Planning from Feedback with Language Models.](#)

Mishra, M., Kumar, P., Bhat, R., Murthy, R. V., Contractor, D., & Tamilselvam, S. (2023, May 19). [Prompting with Pseudo-Code Instructions.](#)

DiCuffa, S., Zambrana, A., Yadav, P., Madiraju, S., Suman, K., & AlOmar, E. A. (2025, June 2). [Exploring Prompt Patterns in AI-Assisted Code Generation: Towards Faster and More Effective Developer-AI Collaboration.](#)

이번 세션을 마치겠습니다
감사합니다.