

프롬프트 엔지니어링

- **프롬프트엔지니어링 기본원리**
- 구체적인 프롬프트 구성을 위한 6가지 구성요소
- 프로그래밍을위한 프롬프트

Song, Young Ohk

프롬프트 엔지니어링 기초

1. 프롬프트엔지니어링 기본원리

(1.1) 프롬프트 엔지니어링의 정의

❖ 정의:

- 대규모 언어 모델(LLM)과 같은 생성형 AI가 사용자의 의도에 맞는 최적의 결과물을 생성하도록,
- 입력(프롬프트)을 설계, 테스트, 개선하는 모든 과정.

❖ 핵심: AI에게 "무엇을, 어떻게 해달라고" 명확하고 효과적으로 요청하는 기술

❖ 비유:

- AI를 위한 "사용 설명서" 작성법
- AI와의 효과적인 "대화법"

Song, Young Ohk

프롬프트 엔지니어링

Very subtle

1. 프롬프트엔지니어링 기본원리

(1.2) 프롬프트 엔지니어링의 필요성

❖ AI는 "만능" 이 아니다

❖ AI의 한계: AI는 입력된 프롬프트에 절대적으로 의존하여 응답을 생성합니다.

- "척하면 척" 알아서 해주지 않습니다. 사용자가 명령하고 안내해야 합니다.

❖ 프롬프트 = AI의 나침반:

- 좋은 프롬프트: 원하는 목적지로 정확히 안내 -> 유용한 결과물
- 나쁜/모호한 프롬프트: 길을 잃게 만들 -> 엉뚱하거나 쓸모없는 결과물

❖ 예시:

- (X) "여행 계획 짜줘." -> 어디로? 언제? 누구와? 예산은?
- (O) "다음 주 주말, 2명이 부산으로 1박 2일 기차여행 갈 건데, 맛집 포함된 관광 코스 추천해줘. 예산은 1인당 20만원이야."

❖ 결론: AI의 능력을 제대로 활용하려면, 명확한 방향 제시가 필수입니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.2) 프롬프트 엔지니어링의 필요성

❖ "내가 원하는 결과 " 를 얻기 위해

- ❖ 의도 명확화: 머릿속의 추상적인 아이디어나 복잡한 요구사항을 AI가 이해할 수 있는 언어로 번역하는 과정입니다.
- ❖ 결과물 제어: 단순히 정보를 얻는 것을 넘어, 원하는 형식, 어조, 관점, 길이 등을 갖춘 결과물을 얻으려면 섬세한 프롬프트 조정이 필요합니다.
 - ❖ 예: 보고서 요약, 이메일 초안 작성, 특정 인물 어투 흉내 내기 등
- ❖ 오해 방지: 프롬프트가 불분명하면 AI가 사용자의 의도를 잘못 해석하여 원치 않는 결과를 생성할 수 있습니다.
- ❖ 결론: 구체적이고 명확한 프롬프트는 사용자가 원하는 '바로 그 결과'를 얻을 확률을 비약적으로 높입니다.

Song, Young Ohk

프롬프트 엔지니어링

Very subtle

1. 프롬프트엔지니어링 기본원리

(1.2) 프롬프트 엔지니어링의 필요성

❖ “효과성”과 “생산성” 극대화

❖ 시간 절약:

- 잘 만든 프롬프트 하나로 원하는 결과를 한 번에!
- 여러 번 시도하고 수정하는 불필요한 반복 작업을 줄여줍니다.

❖ 비용 절감:

- (API 사용 시) 프롬프트 입력 및 결과 생성에는 비용(토큰 사용)이 발생합니다.
- 효과적인 프롬프트는 최소한의 시도로 결과를 얻어 비용을 절약합니다.

❖ 새로운 가능성 발견:

- 프롬프트 엔지니어링은 단순히 질문하는 것을 넘어, AI의 숨겨진 능력을 탐색하고 창의적인 활용을 가능하게 합니다. (예: 코드 생성, 소셜 쓰기, 복잡한 문제 분석)

❖ 결론: 효과적인 프롬프트는 AI 활용의 효율과 생산성을 극대화하는 핵심 동력입니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.3) 기본 원리 및 전략

❖ AI를 '도구'로 인식하고, 명확한 지시를 내리려는 자세가 중요합니다.

❖ 핵심 요소:

- 명확성 (Clarity): 모호함 없이 쉽게 이해되도록
- 구체성 (Specificity): 필요한 정보, 원하는 결과의 특징을 상세히
- 맥락 (Context): 배경 정보 제공으로 이해도 높이기
- 간결성 (Conciseness) vs 상세함 (Detail): 상황에 맞는 조절

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

- ❖ 경제 분야 사례(gpt 4o 기준)
- ❖ 프롬프트1 : **경제에 대해 알려줘.**

좋아요! 경제에 대해 알고 싶다고 하셨는데, 경제는 워낙 범위가 넓어서요. 먼저 간단히 개요를 설명드릴게요. 그다음에 궁금한 주제나 분야가 있으면 말씀해주세요. 😊

🔍 경제란?

경제(Economy)는 한정된 자원을 가지고 무엇을, 어떻게, 누구에게 생산하고 분배할 것인지를 다루는 학문이에요. 즉, 사람들이 어떻게 돈을 벌고, 쓰고, 저축하고, 투자하는지를 연구하죠.

📊 경제의 주요 분야

1. 미시경제학 (Microeconomics)
 - 개인, 가계, 기업의 경제 활동 분석
 - 수요와 공급, 시장 가격 결정, 소비자 행동, 기업의 생산과 비용 구조 등
2. 거시경제학 (Macroeconomics)
 - 국가 전체 경제 흐름 분석
 - GDP, 실업률, 물가 상승률(인플레이션), 통화 정책, 재정 정책 등

[문제점]

너무 광범위하고 모호합니다. 어떤 경제(세계 경제, 한국 경제?), 어떤 측면(현황, 역사, 이론?), 누구 대상인지 알 수 없어 AI가 어떤 정보를 제공해야 할지 판단하기 어렵습니다.

💡 중요한 경제 개념들

- GDP (국내총생산): 한 나라에서 일정 기간 동안 생산된 모든 재화와 서비스의 총합.
- 인플레이션: 물가가 전반적으로 오르는 현상.
- 실업률: 일할 능력이 있는데 일자리를 찾지 못한 사람들의 비율.
- 금리: 돈을 빌릴 때 내야 하는 이자율. 중앙은행의 기준금리가 중요한 역할을 해요.
- 수요와 공급: 상품의 가격과 양을 결정하는 핵심 원리.

🌍 경제는 왜 중요할까?

- 개인의 재정 관리(저축, 투자, 소비 등)
- 기업의 경영 전략
- 정부의 정책 결정(세금, 복지, 금리 등)
- 세계 경제와의 상호작용(무역, 환율, 글로벌 경기 등)

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

- ❖ 경제 분야 사례(gpt 4o 기준)
- ❖ 프롬프트2 : 현재 (2025년 4월 기준) 한국의 기준 금리가 물가 상승률에 미치는 영향에 대해 설명해줘. 특히, 금리 변동이 가계 부채와 부동산 시장에 어떤 연관성을 가지는지 경제 뉴스 기사 형식으로 600자 내외로 작성해줘. 대상 독자는 경제에 관심 있는 대학생이야.

[개선 포인트]

구체적 주제: '한국 기준 금리와 물가 상승률의 관계', '가계 부채 및 부동산 시장 연관성'으로 주제 명확화

시점 명시: '현재 (2025년 4월 기준)'

출력 형식: '경제 뉴스 기사 형식'

분량: '600자 내외'

맥락/대상: '경제에 관심 있는 대학생'

기준금리 인하, 물가 상승률에 미치는 영향과 가계부채·부동산 시장의 연관성

한국은행은 2024년 10월부터 2025년 2월까지 세 차례에 걸쳐 기준금리를 총 0.75%포인트 인하하여 현재 2.75%로 조정하였습니다. 이러한 금리 인하는 경기 부양을 목표로 한 조치로, 물가 상승률과 가계부채, 부동산 시장에 다양한 영향을 미치고 있습니다. 뉴데일리 +2 동아일보 +3

한국은행이 발표한 자료에 따르면, 이번 기준금리 인하로 인해 올해 물가상승률은 0.09%포인트, 내년에는 0.20%포인트 상승할 것으로 예상됩니다. 이는 금리 인하로 인한 소비와 투자 증가가 물가에 일정 부분 상승 압력을 가할 수 있음을 시사합니다. 뉴데일리 +1

가계부채 측면에서는, 금리 인하가 가계대출 이자 부담을 낮춰 대출 수요를 촉진할 수 있습니다. 실제로, 지난해 10월 이후의 금리 인하가 올해 가계부채 증가율을 0.6%포인트, 내년에는 1.53%포인트 높일 것으로 추정됩니다. 이는 가계부채 관리에 있어 주의가 필요함을 나타냅니다. 뉴데일리 +4

부동산 시장에 대해서는, 금리 인하로 인한 대출 이자 부담 감소가 주택 구매를 촉진하여 부동산 가격 상승을 유발할 수 있습니다. 특히, 서울 일부 지역의 토지거래허가구역 해제와 맞물려 부동산 관련 가계부채 증가 우려가 제기되고 있습니다. 동아일보 +1

전문가들은 금리 인하가 경기 부양에 기여하지만, 가계부채와 부동산 시장의 과열을 방지하기 위한 정책적 대응이 필요하다고 지적합니다. 따라서, 정부와 금융당국은 거시건전성 정책을 강화하여 가계부채 증가를 억제하고 부동산 시장의 안정을 도모해야 할 것입니다. Chosunbiz +2 뉴데일리 +1

이러한 상황을 종합해보면, 기준금리 인하가 경제 전반에 미치는 영향은 복합적이며, 각 부문에 대한 면밀한 모니터링과 균형 잡힌 정책 대응이 요구됩니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

❖ 경제 분야 사례(gpt 4o 기준)

❖ 프롬프트3 :

제목: 2025년 4월 기준 한국의 기준금리 변동이 물가상승률에 미치는 영향 분석

요청사항:

내용: 2025년 4월 현재 한국의 기준금리 변동이 물가상승률에 미치는 영향을 분석하고, 특히 금리 변동이 가계부채와 부동산 시장에 어떤 연관성을 가지는지 설명해 주십시오.

형식: 경제 뉴스 기사 형식으로 작성해 주십시오.

분량: 600자 내외로 작성해 주십시오.

대상 독자: 경제에 관심 있는 대학생을 대상으로 합니다.

한국은행, 기준금리 인하의 경제적 파급효과 분석

2025년 4월 현재, 한국은행은 경기 부양을 목적으로 기준금리를 2.75%로 인하하였습니다. 이러한 금리 변동은 물가상승률, 가계부채, 부동산 시장에 다양한 영향을 미치고 있습니다.

한국은행의 통화신용정책 보고서에 따르면, 기준금리 인하로 인해 올해와 내년의 물가상승률이 각각 0.09%포인트, 0.20%포인트 상승할 것으로 예상됩니다. 이는 금리 인하가 소비와 투자를 촉진하여 수요 측면에서 물가에 상승 압력을 가할 수 있음을 시사합니다. Nate News

가계부채 측면에서는, 금리 인하로 인한 대출 이자 부담 감소가 가계대출 증가를 유발할 수 있습니다. 실제로, 지난해 4월 이후 가계부채는 증가세로 전환되었으며, 수도권 중심의 부동산 가격 상승과 금리 경쟁 심화로 인해 여름철에 그 증가세가 더욱 확대되었습니다. 이에 따라 금융당국은 올해 가계부채 증가율을 경상성장률(3.8%) 이내로 관리하겠다는 방침을 밝혔습니다. 조세금융신문 +2

부동산 시장에 대해서는, 금리 인하가 주택담보대출 이자 부담을 낮춰 주택 구매 수요를 증가시킬 수 있습니다. 특히, 서울 일부 지역의 토지거래허가구역 해제 등으로 인해 아파트 매매가격 상승 폭이 확대되고 있습니다. 이러한 상황은 부동산 시장의 과열을 초래할 수 있어 주의가 필요합니다. 한국은행 +1

전문가들은 금리 인하가 경기 부양에 기여하지만, 가계부채 증가와 부동산 시장 과열 등의 부작용을 최소화하기 위한 정책적 대응이 필요하다고 지적합니다. 따라서, 금융당국은 가계부채 관리를 강화하고 부동산 시장의 안정을 도모하기 위한 다양한 조치를 시행하고 있습니다.

이러한 복합적인 경제 상황 속에서, 경제에 관심 있는 대학생들은 금리 변동이 실물 경제에 미치는 영향을 면밀히 관찰하고, 개인 재무 관리에 있어 신중한 접근이 필요합니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

- ❖ 여행 일정 사례(gpt 4o 기준)
- ❖ 프롬프트1 : 여행 계획을 좀 짜줘.

좋죠! 여행 계획 도와드릴게요 😊

그런데 몇 가지 먼저 여쭙볼게요:

1. 여행지는 어디인가요? (예: 국내/해외, 특정 도시나 나라 등)
2. 여행 기간은 얼마나 되나요? (예: 3박 4일, 주말만, 일주일 등)
3. 출발 예정일은 언제인가요?
4. 여행 스타일은 어떤 걸 좋아하세요?
 - 휴양 vs 액티비티
 - 자연 vs 도시
 - 맛집 투어 vs 관광지 위주 등
5. 예산이 어느 정도인가요? (대략적인 금액대 알려주셔도 좋아요)

이 정보 알려주시면 딱 맞는 여행 일정 짜드릴게요! 🗺️📍

[문제점]

어디로, 언제, 누구와, 얼마의 예산으로, 무엇을 하고 싶은지에 대한 정보가 전혀 없습니다. AI는 사용자의 선호도를 전혀 알 수 없습니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

❖ 여행 일정 사례(gpt 4o 기준)

❖ 프롬프트2 :

다음 달 마지막 주 주말(5월 24일~25일), 20대 여성 친구 4명이 서울에서 출발하는 1박 2일 국내 기차 여행 계획을 세워줘. 예산은 1인당 25만원이고, 자연 풍경 감상과 현지 맛집 탐방(카페 포함)을 좋아해. 숙소는 깔끔한 호텔이나 펜션이었으면 좋겠어. 추천 여행지 2곳과 함께 각 여행지별 상세 일정(교통, 숙소, 식사, 활동 포함) 및 예상 경비를 정리해서 표 형식으로 보여줘.

❖ 결과

안녕하세요! 20대 여성 친구 4분이 함께하는 1박 2일 국내 기차 여행을 계획해드리겠습니다. 자연 풍경 감상과 현지 맛집 탐방(카페 포함)을 선호하시며, 깔끔한 호텔이나 펜션을 원하신다는 점을 고려하여 두 곳을 추천드립니다:

1. 전주
2. 광주

각 여행지별 상세 일정을 교통, 숙소, 식사, 활동, 예상 경비와 함께 표로 정리해드리겠습니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

❖ 여행 일정 사례(gpt 4o 기준)

❖ 프롬프트2 : 결과

1. 전주 여행 일정

시간	일정 및 내용	비고
1일 차		
08:00	서울역 출발 - KTX 탑승 (약 1시간 40분 소요)	KTX 왕복 약 60,000원
09:40	전주역 도착 후 택시로 한옥마을 이동 (약 15분 소요)	택시비 약 10,000원
10:00	전주한옥마을 도착 - 한복 대여 후 마을 산책 및 사진 촬영	한복 대여 15,000원
12:00	현지 맛집 '베테랑 칼국수'에서 점심 식사	식사 8,000원
13:30	경기전 방문 - 태조 이성계 어진 봉안소 탐방	입장료 3,000원
15:00	전통성당 및 풍남문 방문	무료
16:00	한옥마을 내 카페 '차경' 방문 - 전통 한옥 분위기에서 음료와 디저트 즐기	음료 및 디저트 10,000원
18:00	숙소 체크인 - 한옥스테이 (예: '사랑가득 펜션')	1박 약 120,000원
19:30	남부시장 야시장 방문 - 다양한 현지 음식 탐방 및 저녁 식사	식사 및 간식 15,000원

2일 차

08:00	숙소 조식 후 체크아웃	조식 포함
09:30	덕진공원 방문 - 연꽃과 자연 풍경 감상	무료
11:00	카페 '호노니' 방문 - 유명한 크로플 맛보기	음료 및 디저트 12,000원
12:30	현지 맛집 '한국집'에서 전주비빔밥으로 점심 식사	식사 12,000원
14:00	택시로 전주역 이동	택시비 약 10,000원
14:30	전주역 출발 - KTX 탑승	
16:10	서울역 도착	

예상 경비 (1인당):

- 교통 (KTX 왕복): 약 60,000원
- 택시비: 약 5,000원 (4인 기준 분담)
- 숙박: 약 30,000원 (4인 기준 분담)
- 식사 및 간식: 약 57,000원
- 입장료 및 한복 대여: 약 18,000원

총합: 약 170,000원

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

❖ 여행 일정 사례(gpt 4o 기준)

❖ 프롬프트2 : 결과

2. 광주 여행 일정

시간	일정 및 내용	비고
1일자		
08:00	용산역 출발 - KTX 탑승 (약 1시간 50분 소요)	KTX 왕복 약 60,000원
09:50	광주송정역 도착 후 택시로 1913송정역시장 이동 (약 5분 소요)	택시비 약 5,000원
10:00	1913송정역시장 탐방 - 다양한 먹거리와 기념품 구경	간식 및 기념품 10,000원
12:00	현지 맛집 '영명국밥'에서 점심 식사	식사 9,000원
13:30	무등산 국립공원 방문 - 자연 풍경 감상 및 산책	무료
16:00	카페 '가배당' 방문 - 한옥카페에서 전통과 현대의 조화 즐기기	음료 및 디저트 12,000원
18:00	숙소 체크인 - 깔끔한 호텔 (예: '마드리드 호텔 광주')	1박 약 160,000원
19:30	동명동 카페거리 방문 - 저녁 식사 및 카페 탐방	식사 및 음료 20,000원

2일자

08:00	호텔 조식 후 체크아웃	조식 포함
09:30	국립아시아문화전당 방문 - 문화 예술 전시 관람	무료
11:30	카페 '사월의 파랑' 방문 - 북카페에서 독서와 커피 타임	음료 및 디저트 10,000원
13:00	현지 맛집 '빛고을떡갈비'에서 떡갈비 정식으로 점심 식사	식사 15,000원
14:30	택시로 광주송정역 이동	택시비 약 5,000원
15:00	광주송정역 출발 - KTX 탑승	
16:50	용산역 도착	

예상 경비 (1인당):

- 교통 (KTX 왕복): 약 60,000원
- 택시비: 약 2,500원 (4인 기준 분담)
- 숙박: 약 40,000원 (4인 기준 분담)
- 식사 및 간식: 약 66,000원
- 입장료 등 기타: 무료

총합: 약 168,500원

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

❖ 여행 일정 사례(gpt 4o 기준)

❖ 프롬프트3 :

1박 2일 국내 기차 여행 계획을 세워줘. 아래 조건에 맞춰 추천 여행지 2곳을 선정하고, 각 여행지별 상세 일정(교통, 추천 숙소, 식사, 활동 포함)과 1인당 예상 경비를 정리해서 표 형식으로 보여줘.

[여행 기본 정보]

- * 여행객: 20대 여성 친구 4명
- * 여행 기간: 5월 24일 (토) ~ 5월 25일 (일) (1박 2일)
- * 출발지: 서울
- * 주요 교통수단: 기차 (KTX, ITX 등 활용)
- * 총 예산: 1인당 25만원 이내

[여행 선호도]

- * 활동: 아름다운 자연 풍경(바다, 산, 호수 등) 감상, 지역 특색 맛집 및 분위기 좋은 카페 방문
- * 숙소: 청결하고 만족도 높은 호텔 또는 펜션 (구체적인 숙소명과 예상 가격 포함 추천)

[요청 결과물]

1. 위 조건과 선호도에 가장 적합한 "추천 여행지 2곳" 선정 및 간단한 추천 이유 명시.
2. 선정된 각 여행지별 "1박 2일 상세 일정표" 작성.
 - * 시간대별 활동 및 이동 경로 포함.
 - * 이용 교통편(기차 시간 포함) 및 예상 소요 시간 명시.
3. 일정표 내에 "추천 숙소, 식당, 카페" 정보 포함 (이름, 특징, 예상 비용).
4. 1인당 "총 예상 경비" 산출 (교통비, 숙박비, 식비, 활동비 등 구분).
5. 위 모든 내용을 "하나의 표(Table)" 형식으로 깔끔하게 정리하여 제시.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.4) 사례 분석

❖ 여행 일정 사례(gpt 4o 기준)

❖ 프롬프트3 : 결과

✅ 추천 여행지 2곳

여행지	추천 이유
강릉	동해 바다와 안목해변, 순두부 맛집, 감성 카페, 청결한 해변 숙소가 조화를 이룸
여수	바다 전망, 낭만포차 거리, 오동도 산책로, 전통 시장 먹거리, 전망 좋은 숙소 등 다양

📅 1박 2일 상세 일정 & 예상 경비 정리표

항목	강릉	여수
교통	🚆 KTX 서울역 → 강릉역 (왕복 약 60,000원, 2시간 소요)	🚆 KTX 용산역 → 여수엑스포역 (왕복 약 70,000원, 3시간 소요)
숙소	씨마크 레지던스 or 강릉 씨스루호텔 (4인 1실, 240,000원 → 1인 약 60,000원)	여수 라마다 플라자 오션뷰 (4인 1실, 280,000원 → 1인 약 70,000원)
1일차 일정	<ul style="list-style-type: none">- 오전: 서울 출발, 강릉 도착- 점심: 초당순두부 마을 (15,000원)- 오후: 안목해변 카페거리 산책 + 사진 촬영 (카페 12,000원)- 저녁: 강릉 중앙시장 포장마차 탐방 (20,000원)- 숙소 체크인 및 휴식	<ul style="list-style-type: none">- 오전: 용산 출발, 여수 도착- 점심: 여수 아바이김밥/해산물 비빔밥 (13,000원)- 오후: 오동도 산책로, 바다 전망 카페 (카페 10,000원)- 저녁: 낭만포차 거리 식사 (해산물 플레이트 25,000원)- 숙소 체크인 및 야경 감상
2일차 일정	<ul style="list-style-type: none">- 조식 후 경포호 산책- 점심: 동화가든 짬순 (15,000원)- 기차 탑승 후 서울 복귀	<ul style="list-style-type: none">- 조식 후 여수 돌산공원 + 케이블카 (입장/이용료 10,000원)- 점심: 여수 풍성한 백반 (12,000원)- 여수엑스포역에서 서울 복귀
1인 예상 경비	교통 60,000원 + 숙박 60,000원 + 식비 62,000원 + 카페 및 기타 15,000원 = 197,000원	교통 70,000원 + 숙박 70,000원 + 식비 65,000원 + 카페 및 기타 20,000원 = 225,000원

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Zero-shot Prompting
- Few-shot Prompting
- Chain-of-Thought Prompting
- Self-Consistency
- Generate Knowledge Prompting
- Prompt Chaining
- Tree of Thoughts
- ...

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Zero-Shot Prompting의 개념과 특징

• 정의

- 모델에게 추가 예제 없이, 단 하나의 명령이나 질문을 통해 직접 답을 생성하도록 요청하는 방식입니다.

• 특징

- 추가적인 학습이나 미세조정(fine-tuning)이 필요하지 않음
- 빠르고 간편하게 사용 가능
- 간단한 작업이나 모델이 이미 잘 알고 있는 내용에 효과적

• 한계

- 복잡하거나 추상적인 문제는 성능이 떨어질 수 있음
- 예측 불가능한 결과가 나올 가능성 존재

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- **Zero-Shot Prompting이 적절한 경우**

- Zero-Shot Prompting은 다음과 같은 상황에 적합합니다.

- **모델이 이미 잘 학습한 지식을 활용할 때**

- 사전 학습된 일반적인 지식 기반 질문 (상식, 정의, 개념 등)

- **간단한 작업이나 직관적인 질문**

- 간단한 정보 요청 (날씨, 간단한 요약, 정의 등)

- **신속한 응답이 필요한 경우**

- 즉각적인 답변이 필요한 질의응답 상황

- **데이터가 부족하거나 추가적인 예시 제공이 어려울 때**

- 예제를 마련하기 어려운 신규 주제나 즉흥적인 질문

- **Few-Shot 예시 구성이 어려울 때**

- 작업에 대한 적절한 예시를 만들거나 찾기 어렵거나, 예시를 포함하면 프롬프트가 너무 길어지는 경우 제로샷이 대안이 됨

- **빠른 테스트 및 탐색**

- 특정 작업에 대해 LLM의 성능을 빠르게 확인하거나 다양한 가능성을 탐색하려는 초기 단계에서 간단하게 시도 가능

- **매우 큰 규모의 모델 사용 시**

- GPT-4, Gemini 등 최신 대규모 언어 모델은 지시 이해 능력이 뛰어나 제로샷 프롬프팅만으로도 복잡한 작업을 꽤 잘 수행

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Zero-Shot Prompting의 예시

[일반 지식 질문]

태양계에서 가장 큰 행성은?

목성입니다.

[간단한 정의 설명 요청]

Zero-shot learning의 정의를 간단하게 설명해줘.

Zero-shot learning이란 모델이 학습 시 보지 못했던 새로운 카테고리에 대해, ...

[번역 작업 요청]

다음 문장을 영어로 번역해줘: "오늘 날씨가 정말 좋네요."

The weather is really nice today.

Very subtle

AI

AI

AI

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Zero-Shot Prompting의 예시

[단순 분류 요청]

다음 문장은 긍정인가요 부정인가요?
문장: "음식 맛이 너무 별로였어요."
부정입니다.

[간단한 수학 계산]

125 × 4는 얼마인가요?.

500입니다.

[텍스트 요약]

다음 뉴스 기사를 세 문장으로 요약해줘

기사의 핵심 내용을 담은 세 문장 요약

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Zero-Shot Prompting의 예시

[코드 설명]

프롬프트: 다음 파이썬 코드의 기능을 설명해줘:

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)
```

(예상)주어진 숫자의 팩토리얼을 계산하는 재귀 함수라는 설명

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Few-shot Prompting의 개념과 특징

• 정의

- 모델에게 한 가지 이상의 예시(일반적으로 2~5개)를 제공하여 수행해야 할 작업의 형태와 규칙을 모델이 더 잘 이해하도록 돕는 기법
- 모델은 제시된 예시를 참고하여 새로운 질문이나 작업에 대해 더욱 정확하고 일관된 답변을 생성

• 특징

• 소량의 예시 제공

- 몇 가지 구체적인 예시를 통해 작업을 명확히 이해할 수 있게 도와줌

• 작업의 맥락을 이해

- 모델이 답변의 형식과 내용을 더 정확하게 유추할 수 있도록 도와줌

• 장점

- 정확도 향상
 - 명확한 예시를 통해 모델의 성능과 응답의 일관성이 크게 향상됨
- 복잡한 작업 가능
 - 모델이 처음 접하거나 복잡한 문제도 비교적 쉽게 이해할 수 있도록 함
- 유연한 활용
 - 예시만 변경하여 다양한 작업을 쉽게 수행할 수 있음

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- **Few-shot Prompting이 적절한 경우**

Few-Shot Prompting은 다음과 같은 상황에서 특히 효과적

- **복잡하거나 세부적인 형식이 필요한 작업**

- 분류, 요약, 번역, 스타일 변환 등 예시를 통해 작업을 명확히 전달해야 할 때

- **모델이 충분히 잘 학습하지 못한 특정 분야**

- 새롭거나 도메인 특화된 작업에 대한 예시를 통해 성능 향상

- **명확한 규칙이나 패턴을 학습시키고 싶을 때**

- 모델이 작업의 논리를 명확히 파악할 수 있도록 예시를 제공

- **특정 출력 형식이나 스타일 요구 될 때**

- JSON 형식 출력, 특정 어조의 글쓰기, 정해진 구조의 답변 등 결과물이 일정한 형식이나 스타일을 따라야 할 때,

- **새롭거나 특수한 작업이 필요할 때**

- 모델이 사전 학습 데이터에서 많이 접해보지 못했을 가능성이 있는 새로운 유형의 작업이나 특정 도메인에 특화된 작업 할 때

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Few-shot Prompting 예시

[특정 레이블로 감성 분류]

프롬프트:

다음은 문장의 감성을 '행복', '슬픔', '분노'로 분류하는 예시입니다.

문장: 오늘 시험을 너무 잘 봐서 기분이 날아갈 것 같아!

감성: 행복

문장: 지갑을 잃어버려서 너무 속상하다.

감성: 슬픔

문장: 약속 시간에 늦는 친구 때문에 화가 난다.

감성: 분노

이제 다음 문장의 감성을 분류해주세요.

문장: 기대했던 영화가 너무 실망스러웠어.

감성:

(예상 결과): 슬픔 (또는 문맥에 따라 분노)

- 예시에서 보여준 '행복', '슬픔', '분노' 중 하나로 답변 유도

[약어 풀네임 변환]

프롬프트:

다음은 기술 용어 약어를 풀네임으로 변환하는 예시입니다.

약어: AI

풀네임: Artificial Intelligence

약어: ML

풀네임: Machine Learning

이제 다음 약어의 풀네임을 알려주세요.

약어: NLP

풀네임:

(예상 결과): Natural Language Processing

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Few-shot Prompting 예시

[간단한 코드 생성 패턴 학습]

프롬프트:

다음은 주어진 설명에 맞는 파이썬 함수를 생성하는 예시입니다.

설명: 두 숫자를 더하는 함수

코드:

```
def add(x, y):  
    return x + y
```

설명: 리스트의 모든 요소를 곱하는 함수

코드:

```
def multiply_list(numbers):  
    result = 1  
    for num in numbers:  
        result *= num  
    return result
```

이제 다음 설명에 맞는 함수를 생성해주세요.

설명: 주어진 문자열을 거꾸로 뒤집는 함수

코드:

(예상 결과): 문제에 맞는 로직

```
def reverse_string(s):  
    return s[::-1]
```

[특정 형식으로 정보 추출]

프롬프트:

다음은 텍스트에서 이름과 이메일 주소를 추출하여 JSON 형식으로 만드는 예시입니다.

텍스트: "담당자는 이순신이며, 이메일 주소는 lss@abc.kr 입니다."

JSON: {"이름": "이순신", "이메일": "lss@abc.kr"}

이제 다음 텍스트에서 이름과 이메일을 추출하여 JSON으로 만들어주세요.

텍스트: "회의록 작성자: 홍길동 (hgd@gmail.com)"

JSON:

(예상 결과):

{"이름": "홍길동", "이메일": "hgd@example.com"}

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Few-shot Prompting 예시

[창의적 문장 생성 (특정 스타일)]

프롬프트:

다음은 동물을 사용하여 비유적인 문장을 만드는 예시입니다.

주제: 빠른 속도

비유: 치타처럼 빠르게 달린다.

주제: 조용한 움직임

비유: 고양이처럼 소리 없이 다가온다.

이제 다음 주제에 대해 동물을 사용한 비유적인 문장을 만들어주세요.

주제: 뛰어난 시력

비유:

(예상 결과):

매처럼 날카로운 눈으로 본다. (또는 다른 적절한 동물을 사용한 비유)

[문장 스타일 변환]

다음 문장을 존댓말로 바꾸세요.

반말: "밥 먹었어?"

존댓말: "식사하셨어요?"

반말: "잘 지내지?"

존댓말: "잘 지내시죠?"

반말: "뭐하고 있어?"

존댓말: "무엇을 하고 계세요?"

반말: "이거 먹어봐."

존댓말:

(예상 결과): 이거 드셔보세요.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- **Few-shot Prompting 예시**

[간단한 요약 작업]

주어진 텍스트를 한 문장으로 요약하세요.

텍스트: "챗GPT는 OpenAI에서 개발한 인공지능 모델로, 다양한 질문에 정확하게 답변하고 복잡한 작업도 처리할 수 있습니다."

요약: "챗GPT는 질문에 정확히 답변하고 복잡한 작업 처리가 가능한 OpenAI의 인공지능입니다."

텍스트: "파이썬은 배우기 쉬운 프로그래밍 언어로, 데이터 분석, 인공지능, 웹 개발 등 다양한 분야에서 널리 쓰이고 있습니다."

요약: "파이썬은 배우기 쉽고 다양한 분야에서 널리 쓰이는 프로그래밍 언어입니다."

텍스트: "커피는 전 세계적으로 인기 있는 음료이며, 특히 아침에 잠을 깨는 데 자주 마십니다."

요약:

(예상 결과)

커피는 전 세계에서 인기 있는 음료로 특히 아침에 잠을 깨는 데 사용됩니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Chain-of-Thought Prompting(CoT)의 개념과 특징

• 정의

- 인공지능 모델이 문제를 해결할 때 결론에 도달하는 중간 과정을 단계별로 명시하도록 유도하는 프롬프팅 방법입니다. 즉, 모델이 논리적 추론 과정을 단계별로 서술하도록 하는 것입니다.
- 이를 통해 모델이 더 복잡하고 추론이 필요한 문제에 대해 정확한 답변을 제공할 가능성을 높일 수 있습니다.

• 특징

- 문제 해결을 위한 단계적 사고 과정(추론)을 포함
- 복잡한 문제나 논리적 추론이 필요한 작업에 유용
- 답의 신뢰성과 투명성 증가 (어떻게 답을 얻었는지 이해 가능)

• 장점

- 복잡한 문제에 대한 정확도 향상
- 모델이 어떤 논리로 결론을 내렸는지 파악 가능
- 모델의 답변 과정 중 오류를 쉽게 파악하고 교정 가능

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Chain-of-Thought Prompting(CoT)이 적절한 경우

- 수학 및 산술 문제
 - 여러 계산 단계가 필요한 응용 문제나 수학 문제 풀이에 매우 효과적입니다. (예: "사과 5개에서 2개를 먹고 3개를 더 사면 몇 개?")
- 상식 추론 및 논리 퍼즐
 - 세상의 지식이나 주어진 정보를 바탕으로 여러 단계를 거쳐 결론을 도출해야 하는 문제에 적합합니다. (예: 알리바이 추리, 인과관계 파악)
- 다단계 지시 사항 처리
 - 여러 지시가 순서대로 또는 복합적으로 주어졌을 때, 각 단계를 순차적으로 처리하고 결과를 종합해야 하는 작업에 유용합니다.
- 계획 수립
 - 특정 목표를 달성하기 위한 단계적인 계획을 세우는 작업에 활용될 수 있습니다.
- 답변의 신뢰성 및 설명 가능성이 중요할 때
 - 단순히 답만 아는 것보다 '왜' 그런 답이 나왔는지 과정이 중요할 때, CoT는 그 과정을 보여줌으로써 신뢰도를 높입니다.
- 제로샷/퓨샷으로 성능이 부족한 추론 문제
 - 일반적인 프롬프팅 방식으로는 모델이 자주 틀리는 복잡한 추론 문제에서 CoT를 적용하면 성능이 개선될 가능성이 높습니다.

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Chain-of-Thought Prompting 예시

[산술 응용 문제]

프롬프트:

질문: 탁구공 한 상자에는 24개의 공이 들어있습니다. 5개의 상자를 사고, 그중에서 15개의 공을 친구에게 주었습니다. 지금 몇 개의 탁구공을 가지고 있나요? 단계별로 생각해서 답해주세요.

(예상 결과 - 모델의 사고 과정 포함):

1. 한 상자에는 24개의 탁구공이 있습니다.
2. 5개의 상자를 샀으므로, 총 공의 개수는 $24\text{개}/\text{상자} \times 5\text{상자} = 120\text{개}$ 입니다.
3. 친구에게 15개의 공을 주었으므로, 남은 공의 개수는 $120\text{개} - 15\text{개} = 105\text{개}$ 입니다.

최종 답: 105개의 탁구공을 가지고 있습니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Chain-of-Thought Prompting 예시

[논리 추론 문제]

프롬프트:

질문: 방에는 세 명의 사람 A, B, C가 있습니다. A는 B보다 키가 큼니다. C는 A보다 키가 큼니다. 세 사람 중 누가 가장 키가 큰가요?
단계별로 생각해서 답해주세요.

(예상 결과 - 모델의 사고 과정 포함):

1. 주어진 정보는 다음과 같습니다:

- $A > B$ (A는 B보다 크다)

- $C > A$ (C는 A보다 크다)

2. 두 정보를 종합하면 $C > A$ 이고 $A > B$ 입니다.

3. 따라서 키 순서는 $C > A > B$ 가 됩니다.

4. 이 순서에서 가장 키가 큰 사람은 C입니다.

최종 답: C가 가장 키가 큼니다.

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Chain-of-Thought Prompting 예시

[상식 추론 및 계획]

프롬프트:

질문: 친구 생일 선물을 사러 쇼핑몰에 가려고 합니다. 가기 전에 무엇을 준비해야 할까요? 단계별로 생각해서 목록을 만들어주세요.

(예상 결과 - 모델의 사고 과정 포함):

1. 목표는 친구 생일 선물을 사러 쇼핑몰에 가는 것입니다.
2. 쇼핑몰에 가려면 무엇이 필요할지 생각합니다.
 - 선물을 살 돈이 필요합니다. (현금 또는 카드)
 - 어떤 선물을 살지 대략적인 아이디어가 있으면 좋습니다. (선물 목록 또는 예산)
 - 쇼핑몰까지 이동할 교통 수단과 비용을 고려해야 합니다. (대중교통 카드, 자동차 키 등)
 - 개인 소지품이 필요합니다. (지갑, 휴대폰, 신분증 등)
 - 쇼핑백이 필요할 수도 있습니다. (선물을 담을 가방)
3. 이 생각들을 정리하여 준비물 목록을 만듭니다.

최종 답: 쇼핑몰에 친구 선물을 사러 가기 전에 준비해야 할 것들은 다음과 같습니다:

- 지갑 (현금/카드 포함)
- 휴대폰
- 선물 아이디어 또는 예산
- 교통 카드 또는 자동차 키
- (선택 사항) 장바구니/쇼핑백

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Chain-of-Thought Prompting 예시

[정보 조합 및 계산]

프롬프트:

질문: 1시간에 5km를 걷는 사람이 있습니다. 이 사람이 30분 동안 걸었다면 몇 km를 걸었을까요? 소수점 첫째 자리까지 계산하고, 단계별로 설명해주세요.

(예상 결과 - 모델의 사고 과정 포함):

1. 이 사람은 1시간(60분) 동안 5km를 걷습니다.
 2. 걷는 속도는 $5 \text{ km} / 60 \text{ 분}$ 입니다.
 3. 30분 동안 걸은 거리를 계산해야 합니다.
 4. 걸은 거리 = 속도 * 시간 = $(5 \text{ km} / 60 \text{ 분}) * 30 \text{ 분}$
 5. 계산하면 $(5 * 30) / 60 = 150 / 60 = 15 / 6 = 2.5 \text{ km}$ 입니다.
- 최종 답: 2.5 km를 걸었습니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Chain-of-Thought Prompting 예시

[간단한 물리 추론]

프롬프트:

질문: 깃털과 볼링공을 진공 상태에서 같은 높이에서 동시에 떨어뜨리면 어떻게 될까요? 왜 그런지 단계별로 설명해주세요.

(예상 결과 - 모델의 사고 과정 포함):

1. 질문은 진공 상태에서 깃털과 볼링공을 동시에 떨어뜨리는 상황에 대한 것입니다.
2. 진공 상태는 공기 저항이 전혀 없는 상태를 의미합니다.
3. 물체가 자유 낙하할 때, 공기 저항이 없다면 물체의 낙하 가속도는 오직 중력 가속도(g)에만 의존하며, 물체의 질량과는 무관합니다. (갈릴레오의 낙하 실험 원리)
4. 깃털과 볼링공은 질량이 매우 다르지만, 진공 상태에서는 둘 다 동일한 중력 가속도를 받습니다.
5. 따라서 두 물체는 같은 속도로 가속되며 같은 시간 동안 같은 거리를 낙하합니다.
6. 결과적으로 깃털과 볼링공은 동시에 땅에 도달합니다.

최종 답: 깃털과 볼링공은 동시에 땅에 떨어집니다. 왜냐하면 진공 상태에서는 공기 저항이 없고 모든 물체는 질량에 관계없이 동일한 중력 가속도로 낙하하기 때문입니다.

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Chain-of-Thought Prompting 예시

[언어추론]

질문: 다음 중 날 수 있는 동물은 무엇인가요? (코끼리, 펭귄, 박쥐)

답은 박쥐입니다.

단계별 풀이:

1. 코끼리: 크고 무거운 포유류 → 날 수 없음
2. 펭귄: 날개가 있지만 날 수 없는 조류
3. 박쥐: 날개가 있고 실제로 날 수 있는 유일한 포유류
따라서, 날 수 있는 동물은 박쥐입니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Chain-of-Thought Prompting 예시

[단계적 계산(단순 수학)]

질문: $(5 + 3) \times (8 \div 2)$ 의 값을 계산하세요.

단계별 풀이:

1. 괄호 안의 덧셈: $5 + 3 = 8$
2. 괄호 안의 나눗셈: $8 \div 2 = 4$
3. 최종 계산: $8 \times 4 = 32$

답: 32

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Self-Consistency Prompting의 개념과 특징

• 정의

- 여러 개의 서로 다른 추론 경로(Chain-of-Thought)를 샘플링한 뒤, 그중 가장 빈도가 높은 답을 선택하도록 하는 기법입니다.
- 단일 샘플링에 의존하지 않고 "다수결"로 답을 확정함으로써, 노이즈나 편향에 덜 민감한 더 안정적이고 정확한 결과를 얻을 수 있습니다.
- 자기 일관성(Self-Consistency)은 대규모 언어 모델(LLM)이 특히 복잡한 추론 문제에 대해 더 정확하고 일관된 답변을 생성하도록 돕는 디코딩 전략(decoding strategy)입니다.
- 이는 단일 답변만 생성하는 대신, 동일한 질문에 대해 여러 개의 다양한 추론 경로(reasoning path)를 생성한 후, 그 결과들을 종합하여 가장 자주 등장하는 최종 답변을 선택하는 방식입니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Self-Consistency Prompting의 핵심

- 다양한 추론 경로 생성
 - 모델에게 동일한 질문을 주고, 약간의 무작위성(예: 높은 temperature 설정)을 주어 여러 번 답변을 생성하게 합니다. 이때 각 답변은 서로 다른 "사고 과정"이나 "추론 단계" (예: Chain-of-Thought 사용 시)를 포함할 수 있습니다.
- 최종 답변 집계
 - 생성된 여러 답변들에서 최종 결론 부분만을 추출합니다.
- 다수결 원칙 적용
 - 추출된 최종 결론들 중 가장 빈번하게 나타난 것을 최종 답변으로 채택합니다.

마치 여러 명의 전문가에게 같은 질문을 하고, 그들이 각기 다른 방식으로 문제를 풀더라도 가장 많은 전문가가 동의하는 답을 최종 답으로 선택하는 것과 유사합니다. 하나의 추론 과정에서 발생할 수 있는 오류나 잘못된 가정이, 다른 여러 추론 과정에서는 나타나지 않을 수 있다는 점을 활용하여 전체적인 답변의 신뢰도를 높입니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- **Self-Consistency 개념과 특징**

- **특징:**

- 다수결 원칙: 여러 독립적인 추론 과정에서 공통적으로 도출되는 결론이 더 정확할 것이라는 직관에 기반합니다. 마치 여러 명의 전문가에게 자문하여 가장 많은 전문가가 동의하는 의견을 따르는 것과 유사합니다.
- 견고성 향상: 단일 추론 경로는 중간 단계의 작은 오류로 인해 완전히 잘못된 결론에 도달할 수 있습니다. Self-Consistency는 여러 경로를 생성하므로, 일부 경로에 오류가 있더라도 다수의 올바른 경로가 있다면 최종적으로는 정답을 선택할 가능성이 높아집니다.
- Chain of Thought (CoT) 확장: 주로 CoT와 함께 사용되어, CoT가 생성하는 단계별 추론 과정의 다양성을 활용합니다.
- 계산 비용 증가: 동일한 문제에 대해 여러 번 추론을 수행해야 하므로, 단일 추론 방식보다 더 많은 계산 자원이 필요합니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Self-Consistency 사용 적합한 경우

- 수학 및 산술 추론: 여러 단계를 거쳐 계산해야 하는 수학 문제 (예: 응용 문제, 복잡한 계산). 다양한 계산 방법이나 중간 과정이 있을 수 있지만 최종 답은 하나로 정해지는 경우가 많습니다.
- 상식 추론 및 논리 퍼즐: 명확한 정답이 존재하지만 그에 도달하는 논리적 경로가 다양할 수 있는 문제.
- 단계적 추론이 필요한 문제: 정답을 도출하기 위해 여러 논리적 단계를 거쳐야 하며, 각 단계에서 오류가 발생할 가능성이 있는 문제.
- 즉, 정답이 비교적 명확하게 정의될 수 있고, 그 답에 도달하는 다양한 사고 과정이 존재할 수 있는 문제들에 효과적입니다.

• 다중 추론 경로 활용

- 하나의 답변 대신 여러 개의 추론 과정을 생성 및 비교합니다.

• 다수결 기반 선택

- 가장 빈번하게 도출된 최종 결론을 최종 답변으로 선택합니다.

• 정확성 및 강건성 향상

- 특히 복잡한 추론 문제에서 오류 가능성을 줄이고 결과의 신뢰도를 높입니다.

• 디코딩 전략

- 프롬프팅 기법(예: CoT)과 함께 사용되어 그 효과를 증대 시킵니다.

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Self-Consistency Prompting 예시

[산술 응용 문제]

프롬프트:

질문: 탁구공 한 상자에는 24개의 공이 들어있습니다. 5개의 상자를 사고, 그중에서 15개의 공을 친구에게 주었습니다. 지금 몇 개의 탁구공을 가지고 있나요? 단계별로 생각해서 답해주세요.

(예상 결과 - 모델의 사고 과정 포함):

1. 한 상자에는 24개의 탁구공이 있습니다.
2. 5개의 상자를 샀으므로, 총 공의 개수는 $24\text{개}/\text{상자} \times 5\text{상자} = 120\text{개}$ 입니다.
3. 친구에게 15개의 공을 주었으므로, 남은 공의 개수는 $120\text{개} - 15\text{개} = 105\text{개}$ 입니다.

최종 답: 105개의 탁구공을 가지고 있습니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- **Self-Consistency** 사용

Self-Consistency는 단일 프롬프트 기법이라기보다는, 하나의 프롬프트(주로 CoT)를 여러 번 실행하고 그 결과를 집계하는 '메타 기법'에 가깝습니다.

따라서 아래는 Self-Consistency를 적용하기 위해 반복적으로 사용될 수 있는 기본 CoT 프롬프트의 예시입니다.

문제 예시: 간단한 산술 응용 문제

"사과 5개를 사고 각 1,000원을 지불했습니다. 그리고 오렌지 3개를 샀는데, 오렌지 가격은 사과 가격의 절반입니다. 총 얼마를 지불해야 할까요?"

질문: 사과 5개를 사고 각 \$1,000\$원을 지불했습니다. 그리고 오렌지 3개를 샀는데, 오렌지 가격은 사과 가격의 절반입니다. 총 얼마를 지불해야 할까요? 단계별로 생각해서 답을 구해주세요.

응답:

Song, Young Ohk

프롬프트 엔지니어링

- Self-Consistency 전략을 적용하기 위한 기반이 되며, 모델이 일관되게 정답을 낼 수 있는지 확인하는 데 사용

질문 1

사과 5개를 사고 각 \$1,000\$원을 지불했습니다.
그리고 오렌지 3개를 샀는데, 오렌지 가격은 사과 가격의 절반입니다.
총 얼마를 지불해야 할까요? 단계별로 생각해서 답을 구해주세요.

질문 2

사과 5개를 사고 개당 \$1,000\$원을 냈습니다.
오렌지 3개의 가격은 사과 가격의 절반입니다.
총 지불한 금액은 얼마인가요? 단계별로 설명해주세요.

질문 3

각 사과가 \$1,000\$원일 때 사과를 5개 구입했습니다.
오렌지 3개도 샀고, 오렌지 하나는 사과 하나의 절반 가격입니다.
총 얼마를 냈는지 계산 과정을 포함하여 알려주세요.

질문 4

사과 가격이 1,000원이고 5개를 구매했습니다.
오렌지는 3개 샀고, 가격은 사과의 절반입니다.
총 구매 금액을 계산해 주세요. 단계별로 답변해주세요.

질문 5

사과 5개를 각 1,000원에 샀고,
오렌지 3개의 가격은 사과 가격의 반입니다.
총 얼마를 지불했는지 순서대로 계산해주세요.

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Prompt Chaining 개념과 특징

• 정의

- 프롬프트 체이닝(Prompt Chaining) 또는 연쇄 프롬프팅은 하나의 거대하고 복잡한 작업을 여러 개의 더 작고 관리하기 쉬운 하위 작업(sub-task)으로 나눈 뒤, 각 하위 작업을 별도의 프롬프트로 처리하고 그 결과를 다음 프롬프트의 입력으로 연결하는 기법입니다.
- 마치 공장의 조립 라인처럼, 첫 번째 프롬프트(작업 스테이션)의 결과물(처리된 부품)이 다음 프롬프트(다음 작업 스테이션)의 입력으로 전달되어 순차적으로 작업을 완성해 나가는 방식입니다.
- 즉, LLM 호출을 여러 번에 걸쳐 사슬처럼 연결하여 최종 목표를 달성합니다.

• 특징

- 작업 분해: 복잡한 작업을 여러 개의 단순한 단계(프롬프트)로 나눕니다.
- 결과 연결: 한 프롬프트의 출력이 다음 프롬프트의 입력으로 사용됩니다.
- 순차적 처리: 작업이 단계별로 순차적으로 진행됩니다.
- 프롬프트 체이닝이 더 적절한 경우
- 프롬프트 체이닝은 다음과 같은 경우에 특히 유용하고 적절합니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Prompt Chaining 사용

- 매우 복잡하거나 다단계적인 작업
 - 단일 프롬프트로는 안정적으로 처리하기 어려운 매우 복잡한 작업이나 여러 뚜렷한 단계를 거쳐야 하는 작업
- 명확한 작업 흐름(Workflow)이 있는 경우
 - 처리 과정이 자연스럽게 여러 단계로 나뉘는 경우, 각 단계를 별도의 프롬프트로 구현하면 전체 흐름을 명확하게 관리할 수 있습니다.
- 각 단계별 제어 및 신뢰도 향상이 필요할 때
 - 전체 작업을 한 번에 처리할 때보다 각 단계를 나누어 처리하면, 특정 단계에서 오류가 발생했을 때 원인을 파악하고 수정하기 용이합니다. 각 단계의 성공 여부를 확인하며 진행할 수 있어 전체적인 신뢰도를 높일 수 있습니다.
- 다양한 LLM 기능 조합
 - 어떤 단계는 정보 추출에 특화된 프롬프트가 필요하고, 다른 단계는 창의적 글쓰기나 논리적 추론이 필요할 때 각 단계에 최적화된 프롬프트를 사용할 수 있습니다.
- 컨텍스트 길이(Context Length) 제약 관리
 - 처리해야 할 정보의 양이 단일 프롬프트의 컨텍스트 길이를 초과할 때, 작업을 나누어 단계별로 필요한 정보만 처리하도록 구성할 수 있습니다.
- 외부 도구 또는 데이터 연동
 - LLM의 답변을 바탕으로 외부 API를 호출하거나 데이터베이스를 조회한 후, 그 결과를 다시 LLM 프롬프트의 입력으로 사용하여 다음 작업을 진행해야 할 때 유용합니다. (에이전트(Agent)나 ReAct 패턴과 유사)

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[영문 기사 요약 및 번역]

목표: 긴 영어 뉴스 기사의 핵심 내용을 추출하여 한국어로 요약하고, 그 요약본을 다시 스페인어로 번역한다.

프롬프트 1 (핵심 추출): "다음 영어 뉴스 기사에서 가장 중요한 핵심 문장 5개를 정확히 추출해줘: [긴 영어 기사 텍스트]"

[출력 1]: 영어로 된 핵심 문장 5개 목록

프롬프트 2 (한국어 요약): "다음 영어 문장들을 바탕으로, 전체 내용을 이해할 수 있도록 한국어로 2문장 요약을 작성해줘: [출력 1: 영어 핵심 문장 목록]"

[출력 2]: 한국어로 된 2문장 요약

프롬프트 3 (스페인어 번역): "다음 한국어 문장을 스페인어로 번역해줘: [출력 2: 한국어 요약문]"

[출력 3]: 스페인어로 번역된 요약문

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[사용자 리뷰 기반 제품 개선 아이디어 도출]

목표: 여러 사용자 리뷰에서 장단점을 분석하고, 이를 바탕으로 제품 개선 아이디어를 제안한다.

프롬프트 1 (장단점 분류): "다음 사용자 리뷰 목록에서 제품의 장점과 단점을 각각 구분하여 불렛포인트로 정리해줘: [사용자 리뷰 목록]"

[출력 1]: 장점 목록과 단점 목록

프롬프트 2 (개선 아이디어 제안): "[출력 1: 장점 및 단점 목록] 이 장단점 분석 결과를 바탕으로, 이 제품을 개선하기 위한 구체적인 아이디어 3가지를 제안해줘. 특히 단점을 보완하는 데 초점을 맞춰줘."

[출력 2]: 제품 개선 아이디어 3가지

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[복잡한 질문 분해 및 답변 생성]

목표: "기후 변화의 주요 원인은 무엇이며, 이로 인해 한국의 농업은 어떤 영향을 받을 것으로 예상되고, 이에 대한 정부의 주요 대응 정책은 무엇인가?" 라는 복잡한 질문에 답변한다.

프롬프트 1 (원인 파악): "기후 변화의 주요 과학적 원인들을 설명해줘."

[출력 1]: 기후 변화 원인 설명 (예: 온실가스 배출 등)

프롬프트 2 (한국 농업 영향 분석): "[출력 1: 기후 변화 원인] 이러한 기후 변화가 한국의 농업(예: 쌀농사, 과수 재배 등)에 미칠 것으로 예상되는 구체적인 영향들을 알려줘."

[출력 2]: 한국 농업에 미치는 영향 분석 (예: 재배 적지 변화, 병충해 증가 등)

프롬프트 3 (정부 정책 요약): "[출력 2: 농업 영향 분석] 이러한 기후 변화와 농업 영향에 대응하기 위한 한국 정부의 주요 정책이나 대책들을 요약해줘."

[출력 3]: 정부 대응 정책 요약

(최종 통합): 출력 1, 2, 3을 조합하여 최종 답변 구성

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[이메일 초안 작성 및 수정]

목표: 고객 클레임에 대한 정중한 사과와 해결 방안을 제시하는 이메일을 작성하고, 너무 딱딱하지 않게 어조를 수정한다.

프롬프트 1 (초안 작성): "고객 ID [고객 ID]가 제기한 [클레임 내용 요약]에 대해 사과하고, 문제 해결을 위해 [해결 방안]을 제시하는 공식적인 이메일 초안을 작성해줘."

[출력 1]: 공식적인 어조의 이메일 초안

프롬프트 2 (어조 수정): "다음 이메일 초안을 좀 더 공감적이고 부드러운 어조로 수정해줘. 너무 사무적이지 않게 고객의 입장을 이해하고 있음을 표현해줘: [출력 1: 이메일 초안]"

[출력 2]: 수정된 어조의 최종 이메일

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[소설 아이디어 구체화]

목표: '미래 도시의 탐정'이라는 기본 아이디어를 구체적인 캐릭터 설정과 플롯 개요로 발전시킨다.

프롬프트 1 (캐릭터 설정): "'2077년 서울을 배경으로 활동하는 냉소적이지만 유능한 사립 탐정' 캐릭터의 이름, 외모, 성격, 특별한 능력(있다면)을 설정해 줘."

[출력 1]: 탐정 캐릭터 설정 상세 내용

프롬프트 2 (플롯 개요 생성): "[출력 1: 탐정 캐릭터 설정] 이 탐정이 미래 도시의 부패한 기업과 관련된 미스터리한 사건을 해결하는 내용의 소설 플롯 개요(발단-전개-위기-절정-결말)를 작성해줘."

[출력 2]: 소설 플롯 개요

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[SQL 작성 예시 (Prompt Chaining 활용)]

목표: "2023년 1월 1일 이후에 입사한 직원이 3명 이상인 각 부서의 이름과 해당 직원 수를 조회하는 SQL"을 작성한다.

가정된 테이블 스키마:

departments (dept_id INT PRIMARY KEY, dept_name VARCHAR(255))

employees (emp_id INT PRIMARY KEY, emp_name VARCHAR(255), hire_date DATE, dept_id INT)

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[SQL 작성 예시 (Prompt Chaining 활용)]

프롬프트 1: 필요한 정보 식별 및 계획 수립

[사용자 요청]

"2023년 1월 1일 이후에 입사한 직원이 3명 이상인 각 부서의 이름과 해당 직원 수를 조회하는 SQL을 만들어줘."

[테이블 스키마]

- departments (dept_id INT PK, dept_name VARCHAR(255))
- employees (emp_id INT PK, emp_name VARCHAR(255), hire_date DATE, dept_id INT)

[지시]

위 사용자 요청과 스키마를 바탕으로, 필요한 SQL 쿼리를 작성하기 위한 계획을 세워줘. 다음 항목들을 명확히 식별해줘:

1. 필요한 테이블
2. 최종 결과에 표시될 컬럼
3. 테이블 간의 조인(JOIN) 조건
4. 데이터를 필터링할 조건 (WHERE)
5. 데이터를 그룹화할 기준 (GROUP BY)
6. 그룹화된 결과에 적용할 조건 (HAVING)
7. 계산이 필요한 집계 함수 (예: COUNT, SUM)

Very subtle

AI

AI

, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Prompt Chaining 사용 방법

[SQL 작성 예시 (Prompt Chaining 활용)]

프롬프트 1: 필요한 정보 식별 및 계획 수립

[사용자 요청]

"2023년 1월 1일 이후에 입사한 직원이 3명 이상인 각 부서의 이름과 해당 직원 수를 조회하는 SQL을 만들어줘."

[테이블 스키마]

- departments (dept_id INT PK, dept_name VARCHAR(255))
- employees (emp_id INT PK, emp_name VARCHAR(255), hire_date DATE, dept_id INT)

[지시]

위 사용자 요청과 스키마를 바탕으로, 필요한 SQL 쿼리를 작성하기 위한 계획을 세워줘. 다음 항목들을 명확히 식별해줘:

1. 필요한 테이블
2. 최종 결과에 표시될 컬럼
3. 테이블 간의 조인(JOIN) 조건
4. 데이터를 필터링할 조건 (WHERE)
5. 데이터를 그룹화할 기준 (GROUP BY)
6. 그룹화된 결과에 적용할 조건 (HAVING)
7. 계산이 필요한 집계 함수 (예: COUNT, SUM)

[출력 1]: SQL 작성 계획

1. 필요한 테이블: departments, employees
2. 표시될 컬럼: departments.dept_name, 직원 수 (COUNT)
3. 조인 조건: departments.dept_id = employees.dept_id
4. 필터링 조건 (WHERE): employees.hire_date > '2023-01-01'
5. 그룹화 기준 (GROUP BY): departments.dept_id, departments.dept_name (부서별로 그룹화)
6. 그룹 조건 (HAVING): 직원 수 > 3
7. 집계 함수: COUNT(employees.emp_id) 또는 COUNT(*) 로 직원 수 계산

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Prompt Chaining 사용 방법

[SQL 작성 예시 (Prompt Chaining 활용)]

프롬프트 1: 필요한 정보 식별 및 계획 수립

[SQL 작성 계획]

1. 필요한 테이블: departments, employees
2. 표시될 컬럼: departments.dept_name, 직원 수 (COUNT)
3. 조인 조건: departments.dept_id = employees.dept_id
4. 필터링 조건 (WHERE): employees.hire_date > '2023-01-01'
5. 그룹화 기준 (GROUP BY): departments.dept_id, departments.dept_name
6. 그룹 조건 (HAVING): 직원 수 > 3
7. 집계 함수: COUNT(employees.emp_id) 로 직원 수 계산

[지시]

위 계획에 따라, "2023년 1월 1일 이후에 입사한 직원이 3명 이상인 각 부서의 이름과 해당 직원 수를 조회하는 SQL 쿼리"를 작성해줘. 테이블과 컬럼 이름에는 명확성을 위해 테이블 별칭(alias)을 사용해줘 (예: d, e).

[출력 2]: 최종 SQL 쿼리

```
SELECT
    d.dept_name,
    COUNT(e.emp_id) AS employee_count
FROM
    departments d
JOIN
    employees e ON d.dept_id = e.dept_id
WHERE
    e.hire_date > '2023-01-01'
GROUP BY
    d.dept_id, d.dept_name
HAVING
    COUNT(e.emp_id) > 3;
```

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- **Tree of Thoughts(ToT) 개념과 특징**

- **개념:**

- ToT는 문제 해결 과정을 '생각의 나무(Tree of Thoughts)' 구조로 모델링합니다.
- 루트 노드는 초기 문제를 나타내고, 각 노드는 문제 해결 과정의 특정 상태 또는 중간 생각(thought)을 나타냅니다.
- 각 노드에서는 여러 개의 다음 생각(자식 노드)을 생성하고, 이를 평가하여 가장 유망한 경로를 탐색합니다.
- 이 과정은 최종 해결책에 도달하거나 더 이상 탐색할 가치가 없는 경로가 없을 때까지 계속됩니다.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Tree of Thoughts(ToT) 개념과 특징

• 특징:

- 다중 경로 탐색: 기존의 Chain of Thought (CoT)가 단일한 생각의 연쇄(chain)를 따르는 것과 달리, ToT는 여러 가능한 생각의 경로를 동시에 탐색합니다. 이는 마치 나무의 가지가 여러 방향으로 뻗어 나가는 것과 같습니다.
- 생각 생성 (Thought Generation): 현재 상태(노드)에서 LLM을 사용하여 다음 단계로 이어질 수 있는 다양한 중간 생각(자식 노드)들을 생성합니다.
- 상태 평가 (State Evaluation): 생성된 생각들이 최종 목표에 얼마나 가까운지, 또는 얼마나 유망 한지를 평가합니다. 이 평가는 LLM 자체를 사용하거나, 휴리스틱, 외부 규칙 등을 활용할 수 있습니다. 평가는 각 생각의 '가치'를 판단하여 탐색 우선순위를 정하는 데 사용됩니다.
- 탐색 알고리즘 활용: 생각의 나무를 탐색하기 위해 너비 우선 탐색(BFS), 깊이 우선 탐색(DFS)과 같은 체계적인 탐색 알고리즘을 사용합니다. 이를 통해 유망한 경로는 깊게 탐색하고, 가능성이 낮은 경로는 가지치기(pruning)하여 효율성을 높입니다.
- 의도적 추론 및 계획: 단순히 다음 단어를 예측하는 것을 넘어, 문제 해결을 위한 의도적인 계획, 탐색, 필요한 경우 되돌아가기(backtracking)를 가능하게 합니다.

프롬프트 엔지니어링

Very subtle

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

• Tree of Thoughts(ToT) 사용 적합한 경우

- 복잡한 추론 및 계획
 - 정답에 도달하기 위해 여러 단계를 거쳐야 하고, 각 단계에서 다양한 선택지가 존재하는 문제 (예: 수학 문제 증명, 복잡한 논리 퍼즐, 전략 게임 계획).
- 탐색 공간이 넓은 문제
 - 가능한 해결 경로가 매우 많아 최적의 해를 찾기 위해 체계적인 탐색이 필요한 경우 (예: 창의적인 글쓰기, 새로운 아이디어 발상).
- 초기 결정의 영향이 큰 문제
 - 초반의 잘못된 선택이 나중에 큰 실패로 이어질 수 있어 신중한 탐색과 평가가 중요한 문제.
- CoT의 한계가 드러나는 문제
 - 단일 경로 추론만으로는 최적의 해를 찾기 어렵거나 오류에 빠지기 쉬운 문제.

Song, Young Ohk

프롬프트 엔지니어링

1. 프롬프트엔지니어링 기본원리

(1.5) 프롬프트 엔지니어링 기법

- Tree of Thoughts(ToT) 사용 방법

ToT는 단일 프롬프트로 완벽하게 구현하기보다는, 여러 번의 LLM 호출과 이를 제어하는 외부 로직(코드)을 통해 구현되는 경우가 많습니다.

기본적인 아이디어를 보여주는 프롬프트 예시

문제 예시: "24 게임"

주어진 숫자 4개 (예: 4, 6, 8, 8)와 사칙연산(+, -, *, /) 및 괄호를 사용하여 결과가 24가 되도록 만드는 문제입니다.

ToT 방식의 프롬프트 (개념적 예시 - 실제 구현은 더 복잡함)

ToT는 여러 단계의 프롬프트와 평가 로직이 필요하므로, 아래는 그 과정을 단순화하여 보여주는 개념적 프롬프트입니다. 실제로는 각 단계별로 프롬프트를 나누고, 생성된 결과들을 평가하여 다음 단계를 진행합니다.

[참고] 예시는 ToT의 아이디어를 보여주기 위한 것입니다. 실제 ToT 구현은 LLM 호출을 통해 생각을 생성하고, 생성된 생각들을 평가하는 로직(LLM 자체 평가 또는 외부 평가 기준 사용), 그리고 BFS/DFS와 같은 탐색 알고리즘을 코드로 구현하여 전체 과정을 자동화하는 방식으로 이루어집니다. 단일 프롬프트만으로는 이 복잡한 탐색 및 평가 과정을 완벽히 수행하기 어려움이 있습니다.

프롬프트 엔지니어링

문제 정의

주어진 숫자 4, 6, 8, 8과 사칙연산 (+, -, *, /), 괄호를 사용하여 정확히 24를 만드세요. 각 숫자는 한 번씩만 사용해야 합니다.

1단계: 초기 가능한 연산 생성 (생각 생성)

"주어진 숫자 4, 6, 8, 8 중 두 개를 선택하여 가능한 초기 연산 3가지 이상 생성하고, 각 결과와 남은 숫자를 알려주세요."

(LLM 응답 예시)

1. $8 / 8 = 1$. 남은 숫자: 4, 6
 2. $6 * 4 = 24$. 남은 숫자: 8, 8 (-> 잠재적 해답 경로이지만, 남은 숫자를 사용하지 않음)
 3. $8 - 6 = 2$. 남은 숫자: 4, 8
 4. $8 * 8 = 64$. 남은 숫자: 4, 6
- ... 등등

2단계: 각 경로 평가 및 다음 연산 생성 (평가 및 탐색)

(외부 로직 또는 다음 프롬프트에서 각 경로를 평가합니다. 예를 들어, 결과값이 너무 크거나 작으면 낮은 점수를 줍니다. 유망한 경로를 선택합니다.)

"경로 1 (결과 1, 남은 숫자 4, 6) 에서 다음 연산을 시도하여 24에 가까워지는 방법 2가지 이상 생성해주세요."

"경로 3 (결과 2, 남은 숫자 4, 8) 에서 다음 연산을 시도하여 24에 가까워지는 방법 2가지 이상 생성해주세요."

(LLM 응답 예시 - 경로 1 기반)

- 1-1. $6 * 4 = 24$. 이제 1과 24를 사용. ($1 + 24 = 25$, $24 / 1 = 24$) -> $6 * 4 / (8/8) = 24$ 발견!
 - 1-2. $6 + 1 = 7$. 남은 숫자: 4. ($7 * 4 = 28$)
 - 1-3. $4 + 1 = 5$. 남은 숫자: 6. ($5 * 6 = 30$)
- ... 등등

3단계 이상: 반복 및 최종 해답 도출

(이 과정을 반복하며, 각 단계에서 생성된 생각(연산 결과)을 평가하고 유망한 경로를 계속 탐색합니다. 목표값 24에 도달하면 탐색을 종료합니다.)

최종 프롬프트 (해답 확인)

"찾아낸 해답 경로 ' $6 * 4 / (8/8)$ '가 주어진 숫자 4, 6, 8, 8을 모두 한 번씩 사용하고 사칙연산과 괄호를 올바르게 사용하여 24를 만드는지 확인해주세요."

Very subtle

CHAT

AI

AI

프롬프트 엔지니어링

- 프롬프트엔지니어링 기본원리
- **구체적인 프롬프트 구성을 위한 6가지 구성요소**
- 프로그래밍을위한 프롬프트

Song, Young Ohk

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.1) 작업/지시.명령(Task/Instruction) : 명확한 목표 설정

- 정의

- AI가 어떤 역할이나 정체성을 가지고 응답해야 하는지 지정하는 것입니다. 특정 전문가, 캐릭터, 시스템 등의 역할을 부여할 수 있습니다.

- 역할

- 역할 설정은 AI의 응답 톤, 스타일, 관점, 그리고 사용하는 어휘까지 결정하는 데 큰 영향을 줍니다. 예를 들어, '친절한 초등학교 선생님' 역할과 '냉철한 데이터 분석가' 역할의 답변은 확연히 다르겠죠?

- 중요성

- 가장 기본적이고 필수적인 요소, 명령이 없거나 모호하면 AI는 무엇을 해야할지 모른다.

- 예시:

- "요약해줘: [기사 내용]"
- "번역해줘: 다음 문장을 영어로."
- "아이디어 5개 제안해줘: 30대 여성을 위한 생일 선물"
- "비교 분석해줘: 스마트폰 A와 B 모델의 장단점"
- "코드 작성해줘: 파이썬으로 웹 스크래핑하는 코드"

Song, Young Ohk

프롬프트 엔지니어링

2. 프롬프트 구성 요소

(2.2) 맥락 (Context)

- 정의

- 명령을 수행하는 데 필요한 배경 정보, 상황, 제약 조건 등을 제공.

- 역할

- AI가 사용자의 의도를 더 깊이 이해하고, 상황에 맞는 적절한 결과물을 생성하도록 도움.

- 중요성

- 맥락이 부족하면 AI가 엉뚱하거나 일반적인 답변만 내놓을 수 있음.

- 예시

- 명령: "이메일 초안 작성해줘."
- + 맥락: "상황: 고객에게 신제품 출시 안내 메일을 보내려고 해. 대상: 기존 구매 고객 대상이야."
- 명령: "설명해줘."
- + 맥락: "배경: 나는 마케팅 초보자야. 주제: 소셜 미디어 마케팅 전략에 대해 쉽게 설명해줘."

- Tip: 누가, 언제, 어디서, 왜 등 육하원칙(5W1H)을 떠올리며 필요한 배경 정보를 추가하세요.

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.3) 입력 데이터(Input Data)

- 정의

- AI가 처리하거나 참조해야 할 구체적인 정보나 텍스트.

- 역할

- AI 작업의 대상이 되는 재료를 제공.

- 중요성

- 입력 데이터가 없으면 AI는 무엇에 대해 작업해야 할지 알 수 없음. (예: 무엇을 요약할지, 무엇을 번역할지)

- 예시:

- 명령: "요약해줘."
- + 입력 데이터: "다음 기사 내용을: [긴 기사 본문 붙여넣기]"
- 명령: "이 문장의 오류를 수정해줘."
- + 입력 데이터: "문장: '철수가 학교를 갔지만 영희는 집에 있었다.'"
- 명령: "이 정보를 바탕으로 보고서를 써줘."
- + 입력 데이터: "참고 자료: [회의록 내용 또는 데이터 요약]"

- Tip: 텍스트, 코드, 데이터 등 AI가 작업할 내용을 명확하게 제공하세요.

Song, Young Ohk

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.4) 출력 형식(Output Format)

- 정의
 - AI가 생성할 결과물의 구조, 형식, 스타일을 지정.
- 역할
 - 결과물을 원하는 형태로 받아 즉시 활용할 수 있도록 함.
- 중요성
 - 형식을 지정하지 않으면 AI가 임의의 형식으로 결과를 제공하여 추가 작업이 필요할 수 있음.
- 예시:
 - 명령: "장단점을 알려줘."
 - + 출력 형식: "결과는 표 형식으로 정리해줘."
 - 명령: "아이디어를 내줘."
 - + 출력 형식: "각 아이디어를 불릿 포인트 목록으로 작성해줘."
 - 명령: "정보를 추출해줘."
 - + 출력 형식: "결과는 JSON 형식으로 만들어줘." (키:값 형태)
 - 기타: "이메일 형식으로", "마크다운 형식으로", "간결한 문장 3개로" 등
- Tip: 표, 목록, JSON, 이메일, 보고서, 코드 블록 등 원하는 형식을 명확히 요구하세요.

Song, Young Ohk

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.5) 페르소나(Persona) :

- 정의

- AI에게 특정 역할, 직업, 캐릭터를 부여하여 그 입장에서 응답하도록 설정.

- 역할

- 결과물의 관점, 전문성 수준, 스타일 등에 큰 영향을 미침.

- 중요성

- 특정 대상이나 목적에 맞는 맞춤형 응답을 유도할 수 있음.

- 예시:

- "당신은 마케팅 전문가입니다. 소셜 미디어 광고 문구를 3개 작성해줘."
- "친절한 여행 가이드처럼 다음 여행지에 대해 설명해줘."
- "경험 많은 개발자의 입장에서 이 코드의 문제점을 지적해줘."
- "셰익스피어 스타일로 사랑에 대한 짧은 시를 써줘."

- **Tip: AI에게 원하는 역할이나 직업을 명확하게 부여하면 결과물의 깊이와 방향성이 달라집니다.**

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.6) 어조(Tone)

- 정의
 - 결과물의 분위기, 감성적 스타일, 말투를 지정.
- 역할
 - 메시지가 전달되는 느낌을 조절하여 상황과 대상에 맞게 소통.
- 중요성
 - 같은 내용이라도 어조에 따라 받아들이는 느낌이 크게 달라짐.
- 예시:
 - 명령: "사과문을 작성해줘."
 - + 어조: "매우 정중하고 진심으로 사과하는 톤으로."
 - 명령: "제품 소개글을 써줘."
 - + 어조: "밝고 친근하며 유머러스한 톤으로."
 - 명령: "경고 메시지를 작성해줘."
 - + 어조: "단호하고 직접적인 어조로."
 - 기타: "전문적인 톤", "공감하는 어조", "객관적인 톤", "긍정적인 톤" 등
- Tip: 페르소나와 함께 사용하면 더욱 효과적입니다. (예: 전문가 페르소나 + 객관적인 톤)

Song, Young Ohk

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.7) 각 요소의 중요성 및 상호작용

- 명령: 무엇을 할지 (가장 중요!)
- 맥락 & 입력 데이터: 무엇을 가지고 할지 (AI의 이해도 높임)
- 페르소나 & 어조: 어떤 관점과 느낌으로 할지 (결과의 스타일 결정)
- 출력 형식: 어떻게 보여줄지 (결과의 활용도 높임)
- 상호작용:
 - 각 요소는 독립적이면서도 서로 영향을 줍니다.
 - 예: '전문가 페르소나'는 '전문적인 톤'과 '보고서 형식'과 잘 어울립니다.
 - 상황에 따라 필요한 요소를 선택하고 조합하는 것이 중요합니다.

Song, Young Ohk

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.8) 프롬프트는 한 번에 완성되지 않는다! (반복 개선 필요)

- 처음부터 완벽한 프롬프트를 작성하기는 어렵습니다.
- AI의 결과물을 보고 6가지 요소를 기준으로 프롬프트를 수정하고 개선하는 과정이 필수적입니다.
- 명령이 모호했나? -> 더 구체적인 동사 사용
- 맥락이 부족했나? -> 배경 정보 추가
- 출력 형식이 다른가? -> 원하는 형식 명확히 지정
- 톤이 마음에 안 드나? -> 다른 어조 요구
- 실험하고 반복하는 과정을 통해 프롬프트 엔지니어링 실력이 향상됩니다.

Song, Young Ohk

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.8) Markdown 특수 문자

(1) 제목 구분: #, ##,

Markdown 스타일에서 #은 **제목(Heading)**을 나타냅니다.
갯수가 많아질수록 제목의 수준이 낮아집니다.

기호	의미	예시
#	가장 큰 제목 (H1)	# 프로젝트 개요
##	중간 제목 (H2)	## 목적 및 목표
###	소제목 (H3)	### 구현 기능 목록

(2) 인용문: >

인용문 또는 설명문을 표시할 때 사용한다.

(3) 구분선: ---, ***

내용을 구분하고 싶을 때 사용한다.

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.8) Markdown 특수 문자

(4). 목록 표시: -, *, 1., 2.
리스트를 표현할 때 사용합니다.

- **순서 없는 리스트 (Bullet List)**

- , * , + 를 사용합니다.

- 사용자 등록 기능
- 이메일 인증

- **순서 있는 리스트 (Numbered List)**

1. , 2. 등 숫자를 사용합니다.

1. 데이터 수집
2. 모델 학습
3. 결과 평가

리모델링 계획안 작성

기본 정보

- 예산: 7천만 원
- 공간: 주방, 욕실, 거실
- 기간: 2개월

요청 사항

1. 미니멀 모던 스타일 적용
2. 수납 공간 확장
3. 동선 개선

출력 형식 예시:

Song, Young Ohk

프롬프트 엔지니어링

Very subtle

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.8) Markdown 특수 문자

(5). 강조: ******, *_*, ~~~~~~

기호	의미	예시	결과
텍스트	굵게 (bold)	**중요**	중요
<i>_텍스트_</i> 또는 *텍스트*	이탤릭 (italic)	<i>_강조_</i>	강조
~~텍스트~~	취소선	~~삭제됨~~	삭제됨

(6). 코드/명령어: ``코드``

인라인 코드: 명령어나 파일명을 표현할 때
예) python main.py

```
python
def hello():
    print("Hello")
```

Song, Young Ohk

프롬프트 엔지니어링

2. 구체적인 프롬프트 구성을 위한 6가지 구성요소

(2.10) 실습 – 6가지 구성요소를 활용한 프롬프트 작성

다양한 주제를 이용하여 가능한 한 6가지 구성 요소를 활용하여 프롬프트 작성 연습

- 어느 부분을 강조할 지 결정

Song, Young Ohk

예시 1: 기본 순서 조합

• 문제 제시

- 6가지 구성 요소를 순서대로 명확하게 나열하여 인스타그램 광고 문구 생성을 요청하는 가장 기본적인 프롬프트입니다.

• 예시 프롬프트 작성:

- (역할) 당신은 소셜 미디어 마케팅 전문가입니다.
- (작업) 우리 회사의 새로운 친환경 텀블러에 대한 인스타그램 게시물 광고 문구 3가지 버전을 작성해주세요.
- (맥락) 타겟은 20대 대학생이며, 이 텀블러는 보온/보냉 기능이 뛰어나고 디자인이 세련된 친환경 제품입니다. [첨부: 텀블러 이미지, 상세 스펙]
- (형식) 각 버전은 최대 20단어의 짧은 문장과 관련 해시태그 3개로 구성되어야 하며, 번호 매겨진 목록으로 결과를 보여주세요.
- (제약 조건) 너무 딱딱한 말투나 '필수템', '인싸템' 같은 유행어는 사용하지 말고, 가격 정보는 제외해주세요.
- (목표) 이 광고 문구의 목표는 사용자들이 제품 상세 페이지를 클릭하도록 유도하는 것입니다.

Song, Young Ohk

예시 2: 맥락 강조 조합

- 문제 제시

- 제품 및 타겟 정보를 먼저 제시하여 AI가 맥락을 충분히 이해한 상태에서 작업을 시작하도록 유도하는 프롬프트입니다.

- 예시 프롬프트 작성:

- (맥락) 저희는 20대 대학생들을 위한 새로운 친환경 텀블러를 출시했습니다. 뛰어난 보온/보냉과 세련된 디자인이 특징입니다.

[첨부: 텀블러 이미지, 상세 스펙]

- (역할) 소셜 미디어 마케팅 전문가로서,
- (작업) 이 제품을 홍보할 인스타그램 광고 문구 3가지 버전을 제안해주세요.
- (목표) 목표는 제품 상세 페이지 트래픽 증대입니다.
- (형식) 각 버전은 20단어 이내 문장과 해시태그 3개로 구성하고, 번호 목록으로 제시해주세요.
- (제약 조건) 가격 언급 없이, 친근하지만 유행어는 배제한 톤으로 작성해주세요. ('필수템', '인싸템' 금지)

Song, Young Ohk

예시 3: 작업 및 형식 중심 조합

- 문제 제시

- 수행해야 할 작업과 결과물의 형식을 먼저 명확히 지정하여 AI가 무엇을 만들어야 하는지에 집중하도록 하는 프롬프트입니다.

- 예시 프롬프트 작성:

- (작업) 인스타그램 광고 문구 3가지 버전을 작성하는 것이 당신의 임무입니다.
- (형식) 각 버전은 최대 20단어 문장과 해시태그 3개로 구성된 번호 매겨진 목록이어야 합니다.
- (역할) 당신은 이 작업을 수행할 소셜 미디어 마케팅 전문가입니다.
- (맥락) 대상 제품은 20대 대학생 타겟의 친환경 텀블러로, 보온/보냉이 우수하고 디자인이 좋습니다.

[첨부: 텀블러 이미지, 상세 스펙]

- (제약 조건) 딱딱하거나 특정 유행어('필수템', '인싸템')를 사용하지 말고, 가격 정보는 빼주세요.
- (목표) 최종 목표는 제품 상세 페이지 클릭 유도입니다.

Song, Young Ohk

예시 4: 목표 및 역할 우선 조합

- 문제 제시

- 작업의 최종 목표와 AI의 역할을 먼저 강조하여 AI가 전체적인 방향성을 인지하고 작업을 수행하도록 유도하는 프롬프트입니다.

- 예시 프롬프트 작성:

- (목표) 우리의 목표는 새로운 친환경 텀블러의 제품 상세 페이지로 더 많은 트래픽을 유도하는 것입니다.
- (역할) 이 목표 달성을 돕기 위해, 당신은 소셜 미디어 마케팅 전문가로서 활동해야 합니다.
- (작업) 20대 대학생들 사로잡을 인스타그램 광고 문구 3가지 버전을 만들어주세요.
- (맥락) 이 텀블러는 친환경적이며, 보온/보냉 기능과 세련된 디자인을 갖추고 있습니다. [첨부: 텀블러 이미지, 상세 스펙]
- (형식) 결과는 각 버전당 20단어 이내 문장과 해시태그 3개를 포함한 번호 목록 형식입니다.
- (제약 조건) 가격 정보는 제외하고, '필수템', '인싸템' 같은 유행어 없이 친근한 어조를 유지해주세요.

Song, Young Ohk

예시 5: 제약 조건 강조 및 자연스러운 통합

- 문제 제시

- 중요한 제약 조건을 먼저 언급하고, 나머지 요소들을 좀 더 자연스러운 대화체 형식으로 통합하여 전달하는 프롬프트입니다.

- 예시 프롬프트 작성:

- (역할) 소셜 미디어 마케팅 전문가님, 도와주세요!
- (제약 조건) 먼저, 광고 문구에 가격 정보를 넣거나 '필수템', '인싸템' 같은 유행어를 쓰는 것은 피해야 합니다. 너무 딱딱한 톤도 안됩니다.
- (작업) 20대 대학생들이 관심을 가질 만한 새로운 친환경 텀블러의 인스타그램 광고 문구 아이디어 3가지를 제안해주세요.
- (맥락) 이 텀블러는 디자인도 예쁘고 보온/보냉도 잘 되는 친환경 제품입니다. [첨부: 텀블러 이미지, 상세 스펙]
- (형식) 각 아이디어는 20단어 내외의 짧은 문장과 해시태그 3개로 구성해서, 번호 목록으로 보여주시면 좋겠습니다.
- (목표) 궁극적으로는 이 광고를 보고 제품 상세 페이지를 방문하도록 만드는 것이 중요합니다.

Song, Young Ohk

프롬프트 엔지니어링

- 프롬프트엔지니어링 기본원리
- 구체적인 프롬프트 구성을 위한 6가지 구성요소
- **프로그래밍을위한 프롬프트**

Song, Young Ohk

Prompting

- 최적화 포인트
 - 메모리 최적화
 - const & 사용
 - 불필요한 복사 제거 (std::move 활용)
 - unique_ptr/shared_ptr 적절 사용
 - 알고리즘 최적화
 - 중첩 루프 단순화
 - std::unordered_map vs std::map 선택
 - 컴파일러 친화적 코드
 - inline, constexpr 적용
 - 템플릿 함수 최적화
 - 반복문 개선
 - range-based for 적용
 - reserve()로 벡터 재할당 방지

Song, Young Ohk

Prompting

- 단순히 가독성 개선뿐만 아니라 성능 최적화까지 동시에 시키는 프롬프트 활용

[프롬프트 예시]

이 코드를 리팩토링하면서, 불필요한 연산을 줄이고 메모리 사용을 최소화하는 방향으로 최적화해줘.

- 가독성을 유지하면서 함수/클래스 구조를 개선
- 반복문이나 조건문 최적화
- 불필요한 동적 할당 제거
- const & 참조 사용 가능 시 적용

- 리팩토링과 최적화를 분리하지 말고 한 프롬프트에 묶어서 요청

Song, Young Ohk

Prompting

- 리팩토링 전용 프롬프트

[프롬프트 예시]

이 코드를 리팩토링해줘.

- SOLID 원칙을 참고하여 클래스와 함수 역할을 분리
- 중복된 코드를 제거하고 함수 추출
- 네이밍 컨벤션을 Google C++ Style Guide에 맞게 정리
- 불필요한 전역 변수 제거
- 가독성을 위해 헤더(.h)와 소스(.cpp) 분리

Song, Young Ohk

Prompting

- 최적화 전용 프롬프트

[프롬프트 예시]

이 코드를 성능 최적화해줘.

- 불필요한 복사를 제거하고 가능한 곳에는 const reference(&) 사용
- move semantics(std::move) 적용할 수 있는 부분 최적화
- vector, map 등 STL 컨테이너에 reserve(), emplace_back() 사용
- 반복문을 range-based for로 변경
- 알고리즘을 $O(N^2) \rightarrow O(N \log N)$ 으로 개선할 수 있는지 확인
- 불필요한 동적 할당(new/delete) 제거, RAII 적용
- inline 함수, constexpr을 적용할 수 있는 부분 최적화

Song, Young Ohk

Prompting

- 리팩토링 + 최적화 전용 프롬프트

[프롬프트 예시]

이 코드를 리팩토링하면서 성능 최적화도 함께 적용해줘.

[리팩토링 기준]

- 함수/클래스를 SRP(Single Responsibility Principle)에 맞게 분리
- 중복된 로직을 제거하고 재사용 가능하게 구성
- 코드 네이밍을 일관성 있게 변경

[최적화 기준]

- const&, move semantics, emplace_back, reserve 적용
- $O(N^2)$ 이상의 알고리즘이 있으면 효율적인 자료구조로 교체
- 불필요한 동적 메모리 제거하고 스마트 포인터 적용
- inline/constexpr 적용
- 멀티스레딩 가능하면 std::async 또는 std::thread 활용
- 컴파일러 경고가 나지 않도록 수정

Prompting

- 리팩토링 + 최적화 전용 프롬프트

[프롬프트 예시]

C++17 이상을 기준으로 이 코드를 리팩토링 및 최적화해줘.

- 가독성을 위한 리팩토링(SOLID 원칙, 중복 제거, 네이밍 개선)
- 성능 최적화(메모리 복사 최소화, move semantics 적극 활용)
- STL 알고리즘(std::sort, std::transform, std::accumulate 등) 적용
- 범위 기반 for, auto, structured bindings, lambda 활용
- 스마트 포인터(unique_ptr, shared_ptr) 및 RAII 적용
- 예외 안전성 보장(try-catch, noexcept, strong exception safety)
- 프로파일링 관점에서 병목이 될 수 있는 부분 최적화 제안
- 멀티스레딩(std::thread, std::future, std::async) 적용 가능성 확인
- 병렬 알고리즘(std::execution::par) 활용 여부 검토

Prompting

Very subtle

```
// before.cpp
#include <bits/stdc++.h>
using namespace std;

struct IntHolder {
    vector<int>* data;
    IntHolder() { data = new vector<int>(); }
    ~IntHolder() { delete data; } // 예외 안전성 취약
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // 입력
    ifstream fin("data.txt"); // 에러 체크 없음
    IntHolder holder;
    int x;
    while (fin >> x) {
        // 음수 제거: O(N^2) 방식(매번 지우거나 복사)
        if (x >= 0) {
            holder.data->push_back(x); // reserve 미사용
        }
    }
}
```

```
// 평균
double sum = 0.0;
for (size_t i = 0; i < holder.data->size(); ++i) {
    sum += (*holder.data)[i]; // 불필요한 반복 인덱싱
}
double mean = holder.data->empty() ? 0.0 : sum / holder.data->size();

// 중앙값: 버블 정렬 (O(N^2))
vector<int> tmp = *holder.data; // 대용량 복사
for (size_t i = 0; i < tmp.size(); ++i) {
    for (size_t j = 1; j < tmp.size(); ++j) {
        if (tmp[j-1] > tmp[j]) {
            int t = tmp[j-1]; tmp[j-1] = tmp[j]; tmp[j] = t;
        }
    }
}
double median = 0.0;
if (!tmp.empty()) {
    size_t n = tmp.size();
    if (n % 2 == 0) median = (tmp[n/2-1] + tmp[n/2]) / 2.0;
    else median = tmp[n/2];
}

// 빈도: map(정렬 트리) 사용 + emplace 미사용
map<int, int> freq;
for (size_t i = 0; i < holder.data->size(); ++i) {
    int v = (*holder.data)[i];
    if (freq.find(v) == freq.end()) freq[v] = 1;
    else freq[v] = freq[v] + 1;
}
```

Prompting

Very subtle

CHAT

```
// Top-K: 전체 정렬 (불필요하게 O(M log M))
int K = 5;
vector<pair<int,int>> items;
for (auto &p : freq) items.push_back(p);
sort(items.begin(), items.end(), [](auto &a, auto &b){
    if (a.second != b.second) return a.second > b.second;
    return a.first < b.first;
});
if ((int)items.size() > K) items.resize(K);

// 출력
cout << "Count: " << holder.data->size() << "\n";
cout << "Mean : " << mean << "\n";
cout << "Median: " << median << "\n";
cout << "Top-" << K << " frequencies:\n";
for (auto &kv : items) {
    cout << kv.first << " -> " << kv.second << "\n";
}
}
```

• 문제점

- RAI(Resource Acquisition Is Initialization) 위반:
new/delete 직접 관리 → 예외 안전성 취약.
- $O(N^2)$ 버블정렬, 음수 필터링 방식도 비효율.
- map 사용으로 불필요한 정렬 비용(로그 팩터).
- reserve()/emplace_back() 미사용, 불필요한 복사 다수.
- 에러 처리 부재, const 정확성 부족, 네이밍/역할 분리 미흡.

Song, Young Ohk

당신은 C++ 퍼포먼스 엔지니어 겸 리팩토링 전문가입니다.

지금 열려 있는 이 C++ “적용 전(before)” 코드를 다음 기준으로 변환한 “적용 후(after)” 버전을 생성하세요.

[리팩토링(SRP)]

1) 기능을 다음 함수로 분리하고 analysis 네임스페이스로 감싸세요.

- [[nodiscard]] optional<vector<int>> load_nonnegatives(const std::string& path) noexcept;
- [[nodiscard]] inline double mean(const vector<int>& v) noexcept;
- [[nodiscard]] double median(vector<int> v);
- [[nodiscard]] unordered_map<int,int> frequencies(const vector<int>& v);
- [[nodiscard]] vector<pair<int,int>> top_k(const unordered_map<int,int>& freq, int K);

2) main()은 위 함수들을 조합만 하도록 단순화합니다.

3) 헤더/소스 단일 after.cpp로 구성하되, 상단에 변경 요약을 주석으로 기록합니다.

[알고리즘 최적화]

- median: std::nth_element 사용 (정렬 $O(N \log N)$ 회피, 평균은 좌/우 중 최대+중앙 원소 평균)
- Top-K: 전체 정렬 금지 → nth_element로 K경계 설정 후 필요한 구간만 정렬
- 빈도: unordered_map + reserve(v.size()) + 전위 증가(++cnt)
- 합계: std::accumulate로 64-bit 누적(long long) → 오버플로우 완화

[메모리 최적화]

- vector는 reserve(), push_back 대신 emplace_back()
- 불필요한 복사 제거(필요 시에만 값 복사, 그 외는 const& 혹은 이동)
- new/delete 금지, RAII 원칙 준수

[현대 C++ 스타일]

- C++17 이상: [[nodiscard]], noexcept, constexpr(가능한 상수), range-based for, auto, 람다/구조적 바인딩
- 불필요한 <bits/stdc++.h> 제거, 필요한 헤더만 포함
- using 별칭(예: using Vec = std::vector<int>)로 가독성 향상

[코드 개선을 위한 프롬프트 작성]

[건고성/품질]

- 파일 오픈 실패 시 std::nullopt 반환 및 main에서 오류 출력 후 종료
- 빈 컨테이너 입력 시 mean/median 0 반환, top_k 빈 컨테이너 처리
- 경고 0을 목표로 컴파일 가능(-std=c++17 -O3)

[출력 형식]

- after.cpp 전체 코드를 제시하세요(별도 설명 없이 코드만).
- 코드 상단 주석에 “변경 요약(알고리즘/메모리/스타일)” 8줄 내외로 포함하세요.
- 각 핵심 최적화 지점에는 한 줄 근거 주석을 남기세요.

[검증 가이드(주석으로 포함)]

- 예시 data.txt로 빠르게 검증하는 빌드 커맨드:
g++ -O3 -std=c++20 after.cpp -o after && ./after
- 동일 입력에서 before와 결과가 일치함을 전제로 하며, 대용량 데이터에서 시간/메모리 이점이 발생해야 합니다.

Prompting

Very subtle

[1차 시도 예]

```
// after.cpp (C++17 이상 권장)
#include <algorithm>
#include <execution>    // 병렬 알고리즘 고려 시
#include <fstream>
#include <iostream>
#include <numeric>
#include <optional>
#include <string>
#include <unordered_map>
#include <utility>
#include <vector>

namespace analysis {

using Vec = std::vector<int>;
constexpr int DEFAULT_TOP_K = 5;

// 파일에서 정수 로드 (음수는 스킵). 예외 안전, 에러 체크.
[[nodiscard]] std::optional<Vec> load_nonnegatives(const std::string& path) noexcept
{
    std::ifstream fin(path);
    if (!fin) return std::nullopt;

    Vec v;
    v.reserve(1'000); // 대략치라도 먼저 확보(재할당 감소)
    int x;
    while (fin >> x) {
        if (x >= 0) v.emplace_back(x);
    }
    return v;
}
```

```
// 평균 (const&, 범용 numeric 사용)
[[nodiscard]] inline double mean(const Vec& v) noexcept {
    if (v.empty()) return 0.0;
    const auto s = std::accumulate(v.begin(), v.end(), 0LL); // overflow 완화
    return static_cast<double>(s) / static_cast<double>(v.size());
}

// 중앙값: O(N log N) 또는 복사없이 nth_element로 O(N) 근사
[[nodiscard]] double median(Vec v) { // 값 복사: 원본 보존
    if (v.empty()) return 0.0;
    const auto n = v.size();
    auto mid = v.begin() + n / 2;
    std::nth_element(v.begin(), mid, v.end());
    if (n % 2) return static_cast<double>(*mid);
    const auto left_max = *std::max_element(v.begin(), mid);
    return (static_cast<double>(left_max) + static_cast<double>(*mid)) / 2.0;
}

// 빈도 계산: unordered_map + reserve + 전위 증가
[[nodiscard]] inline std::unordered_map<int,int> frequencies(const Vec& v) {
    std::unordered_map<int,int> freq;
    freq.reserve(v.size());
    for (const auto& val : v) ++freq[val];
    return freq;
}
```

Song, Young OhK

Prompting

Very subtle

```
// Top-K 빈도: 전체 정렬 대신 partial_sort/nth_element로 최적화
[[nodiscard]] std::vector<std::pair<int,int>>
top_k(const std::unordered_map<int,int>& freq, int K) {
    std::vector<std::pair<int,int>> items(freq.begin(), freq.end());
    if ((int)items.size() <= K) {
        std::sort(items.begin(), items.end(),
            [](const auto& a, const auto& b){
                if (a.second != b.second) return a.second > b.second;
                return a.first < b.first;
            });
        return items;
    }
    auto nth = items.begin() + K;
    std::nth_element(items.begin(), nth, items.end(),
        [](const auto& a, const auto& b){
            if (a.second != b.second) return a.second > b.second;
            return a.first < b.first;
        });
    items.resize(K);
    std::sort(items.begin(), items.end(),
        [](const auto& a, const auto& b){
            if (a.second != b.second) return a.second > b.second;
            return a.first < b.first;
        });
    return items;
}

} // namespace analysis
```

```
int main() {
    using namespace analysis;
    const std::string path = "data.txt";
    const int K = DEFAULT_TOP_K;

    const auto maybe = load_nonnegatives(path);
    if (!maybe) {
        std::cerr << "Error: failed to open " << path << "\n";
        return 1;
    }
    const Vec& data = *maybe;
    const double m = mean(data);
    const double md = median(data);
    const auto freq = frequencies(data);
    const auto top = top_k(freq, K);

    std::cout << "Count : " << data.size() << "\n";
    std::cout << "Mean : " << m << "\n";
    std::cout << "Median: " << md << "\n";
    std::cout << "Top-" << K << " frequencies:\n";
    for (const auto& [value, cnt] : top) {
        std::cout << value << " -> " << cnt << "\n";
    }
}
```

data.txt (예시)

1 3 5 3 3 10 5 7 7 7 7 0 1 2 2 2 2 2 100 100 100 100 100 100 -1 -2

Prompting

Very subtle

```
// after.cpp (C++17+ refactor & perf-optimized)
```

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <numeric>
#include <string>
#include <string_view>
#include <unordered_map>
#include <utility>
#include <vector>

namespace stats {

using i64 = long long;
using Frequency = std::pair<int,int>;

constexpr int DEFAULT_TOPK = 5;

// Small comparators are good inline targets (helps inlining & branch prediction).
inline bool freq_then_value_desc(const Frequency& a, const Frequency& b) noexcept {
    // sort by frequency desc, then value asc (ties broken deterministically)
    if (a.second != b.second) return a.second > b.second;
    return a.first < b.first;
}
```

[2차 시도 예]

Song, Young Ohk

Prompting

```
/// Read non-negative integers from file. Uses reserve() to reduce reallocations.
/// [[nodiscard]] encourages callers to use the returned value.
[[nodiscard]] std::vector<int> read_nonnegatives(std::string_view filename) {
    std::ifstream fin(std::string(filename));
    if (!fin) {
        throw std::runtime_error("Failed to open file: " + std::string(filename));
    }

    std::vector<int> data;
    data.reserve(1024); // reserve to avoid frequent growth (heuristic)

    int x;
    while (fin >> x) {
        if (x >= 0) {
            data.emplace_back(x); // emplace_back avoids extra copy/move
        }
    }
    return data; // NRVO/Move elision
}

/// Mean via accumulate with 64-bit sum to avoid overflow.
[[nodiscard]] inline double mean(const std::vector<int>& v) noexcept {
    if (v.empty()) return 0.0;
    const i64 s = std::accumulate(v.begin(), v.end(), i64{0}); // 64-bit sum minimizes overflow risk
    return static_cast<double>(s) / static_cast<double>(v.size());
}
```

Song, Young Ohk

Prompting

Very subtle

```
/// Mean via accumulate with 64-bit sum to avoid overflow.
[[nodiscard]] inline double mean(const std::vector<int>& v) noexcept {
    if (v.empty()) return 0.0;
    const i64 s = std::accumulate(v.begin(), v.end(), i64{0}); // 64-bit sum minimizes overflow risk
    return static_cast<double>(s) / static_cast<double>(v.size());
}

/// Median via nth_element (linear-time average); copy to avoid mutating input.
/// For even n: average of middle two; compute max of lower half in O(n).
[[nodiscard]] double median(std::vector<int> v) {
    if (v.empty()) return 0.0;

    const std::size_t n = v.size();
    const std::size_t mid = n / 2;

    std::nth_element(v.begin(), v.begin() + static_cast<std::ptrdiff_t>(mid), v.end());
    const int upper_mid = v[mid];

    if (n % 2 == 1) {
        return static_cast<double>(upper_mid);
    }
    // For even n, median = (max of lower half + upper_mid)/2
    const int lower_mid = *std::max_element(v.begin(), v.begin() + static_cast<std::ptrdiff_t>(mid));
    return (static_cast<double>(lower_mid) + static_cast<double>(upper_mid)) / 2.0;
}
```

Song, Young Ohk


```

/// Frequency count using unordered_map; reserve() to reduce rehashes; ++pre-increment is efficient.
[[nodiscard]] std::unordered_map<int,int> frequencies(const std::vector<int>& v) {
    std::unordered_map<int,int> freq;
    // Reserve ~2x to lower load factor and minimize rehashing under skewed distributions.
    freq.reserve(v.size() * 2 + 1);

    for (const int value : v) {
        ++freq[value]; // default-initialize to 0, then pre-increment (avoids extra find/branch)
    }
    return freq;
}

/// Top-K by frequency using nth_element then sorting the top segment for stable presentation.
[[nodiscard]] std::vector<Frequency> top_k(const std::unordered_map<int,int>& freq, int K) {
    std::vector<Frequency> items;
    items.reserve(freq.size()); // single allocation for vectorization of map items
    for (const auto& [value, count] : freq) {
        items.emplace_back(value, count);
    }

    if (K <= 0 || items.empty()) return {};
    if (static_cast<std::size_t>(K) >= items.size()) {
        std::sort(items.begin(), items.end(), freq_then_value_desc); // small N: full sort is fine
        return items;
    }

    auto kth_it = items.begin() + K;
    // nth_element yields top-K in first K positions, order unspecified (expected linear-time)
    std::nth_element(items.begin(), kth_it, items.end(), freq_then_value_desc);
    items.erase(kth_it, items.end()); // keep only K items (cheap erase of the tail)
    // Order the top-K nicely for output (O(K log K))
    std::sort(items.begin(), items.end(), freq_then_value_desc);
    return items;
}

} // namespace stats

```

Prompting

```
int main(int argc, char** argv) {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    try {
        // CLI allows easier testing; defaults to "data.txt".
        const std::string filename = (argc > 1) ? argv[1] : "data.txt";

        // I/O
        auto data = stats::read_nonnegatives(filename);

        // Basic stats
        const double m = stats::mean(data);           // accumulate with 64-bit sum
        const double med = stats::median(data);        // nth_element-based median
        const auto freq = stats::frequencies(data);    // unordered_map with reserve & pre-increment

        // Top-K (nth_element + final sort of top segment)
        constexpr int K = stats::DEFAULT_TOPK;
        const auto topk = stats::top_k(freq, K);

        // Output
        std::cout << "Count : " << data.size() << '\n';
        std::cout << "Mean   : " << m << '\n';
        std::cout << "Median: " << med << '\n';
        std::cout << "Top-" << K << " frequencies:\n";
        for (const auto& [value, count] : topk) { // structured bindings improve readability
            std::cout << value << " -> " << count << '\n';
        }
    } catch (const std::exception& ex) {
        // Explicit error surface for better debuggability & testability.
        std::cerr << "[Error] " << ex.what() << '\n';
        return 1;
    }
    return 0;
}
```

data.txt (예시)

1 3 5 3 3 10 5 7 7 7 7 0 1 2 2 2 2 2 100 100 100 100 100 100 100 -1 -2

Prompting

- 2번의 요청으로 얻어진 코드를 다시 비교 분석
- 최종 2 코드를 이용한 최적의 코드 생성
- 차이점 비교 예시(A: 첨부 code1, B: 첨부 code2)
 - 에러 처리 철학: A=예외 throw/try-catch, B= std::optional 반환(noexcept) → "실패를 값으로 다룸".
 - 입력 경로: A=CLI 인자 지원, B=고정 "data.txt" → A가 테스트/재사용성 우수.
 - 현대 C++ 사용 범위: 둘 다 좋지만, A는 [[nodiscard]]/noexcept/constexpr/string_view/전용 비교자 등 언어 기능 활용이 더 폭넓음. B는 std::optional을 도입했지만 <execution>는 미사용.
 - 미세 성능 튜닝: A는 unordered_map.reserve(v.size()*2+1)로 리해시 여유를 크게 잡음(고빈도/스큐 데이터에 유리). B는 reserve(v.size()).
 - Top-K/중앙값 알고리즘: 둘 다 nth_element 기반으로 동일한 성능 특성. 구현 디테일만 조금 다름.

Song, Young Ohk

Prompting

[1차 + 2차를 통한 최종 코드 생성]

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <numeric>
#include <optional>
#include <string>
#include <string_view>
#include <unordered_map>
#include <utility>
#include <vector>

namespace stats {

using i64 = long long;
using Vec = std::vector<int>;
using FreqMap = std::unordered_map<int,int>;
using Pair = std::pair<int,int>;

constexpr int DEFAULT_TOPK = 5;           // 기본 Top-K
constexpr double FREQ_LOAD_FACTOR = 1.6; // 빈도 맵 reserve 튜닝 상수

// 빈도 내림차순, 값 오름차순(동률) 비교자 - sort/nth_element 모두에서 재사용
struct ByCountDescThenValueAsc {
    bool operator()(const Pair& a, const Pair& b) const noexcept {
        if (a.second != b.second) return a.second > b.second; // count 큰 값 먼저
        return a.first < b.first;                             // 값이 작을수록 먼저
    }
};
```


Prompting

Very subtle

CHAT

Very subtle

```
// [입력] 파일에서 음수 제거하고 비음수만 읽기
// - 최적화: vector.reserve로 재할당 감소, emplace_back 사용
// - 정책: 예외 대신 optional로 실패를 값으로 표현
[[nodiscard]] inline std::optional<Vec>
load_nonnegatives(std::string_view path) {
    std::ifstream fin(std::string(path));
    if (!fin) return std::nullopt;

    Vec v;
    v.reserve(1024); // [최적화] 대략적 초기 용량
    int x;
    while (fin >> x) {
        if (x >= 0) v.emplace_back(x); // [최적화] emplace_back (push_back 동일 비용, 의미 명확)
    }
    return v;
}

// [평균] 64bit 누적으로 오버플로 방지, 비어 있으면 0.0
// - 최적화: std::accumulate + i64 합산
[[nodiscard]] inline double mean(const Vec& v) noexcept {
    if (v.empty()) return 0.0;
    const i64 sum = std::accumulate(v.begin(), v.end(), i64{0});
    return static_cast<double>(sum) / static_cast<double>(v.size());
}
```

AI

Prompting

Very subtle

CHAT

Very subtle

```
// [중앙값] nth_element로 전체 정렬 없이 O(N) 평균 복잡도로 계산
// - 짝수 길이일 때 하한 중앙값은 [begin, mid) 구간의 max로 취득
[[nodiscard]] inline double median(Vec v) {
    if (v.empty()) return 0.0;
    auto mid = v.begin() + static_cast<std::ptrdiff_t>(v.size() / 2);
    std::nth_element(v.begin(), mid, v.end()); // [최적화] 부분 선택
    if (v.size() & 1) {
        return static_cast<double>(*mid);
    } else {
        const int hi = *mid;
        const int lo = *std::max_element(v.begin(), mid); // [최적화] 하한 중앙값만 탐색
        return (static_cast<double>(lo) + static_cast<double>(hi)) / 2.0;
    }
}

// [빈도] unordered_map + reserve, 전위 증가
// - 분포가 치우친 경우를 고려해 여유 있는 reserve로 리해시 감소
[[nodiscard]] inline FreqMap frequencies(const Vec& v) {
    FreqMap freq;
    freq.reserve(static_cast<size_t>(v.size() * FREQ_LOAD_FACTOR) + 1); // [최적화]
    for (const int val : v) {
        ++freq[val]; // [최적화] 전위 증가
    }
    return freq;
}
```


Prompting

Very subtle

```
// [Top-K] 전체 정렬 회피: nth_element로 상위 K를 앞으로 모은 뒤, 앞 K만 정렬
[[nodiscard]] inline std::vector<Pair>
top_k(const FreqMap& freq, size_t K) {
    std::vector<Pair> items;
    items.reserve(freq.size()); // [최적화] 단일 재할당 방지
    for (const auto& kv : freq) items.emplace_back(kv.first, kv.second);

    if (K == 0 || items.empty()) return {};
    if (K >= items.size()) {
        std::sort(items.begin(), items.end(), ByCountDescThenValueAsc{}); // 전부 정렬
        return items;
    }

    auto nth = items.begin() + static_cast<std::ptrdiff_t>(K);
    std::nth_element(items.begin(), nth, items.end(), ByCountDescThenValueAsc{}); // [최적화]
    items.resize(K); // 뒤는 제거
    std::sort(items.begin(), items.end(), ByCountDescThenValueAsc{}); // 상위 K만 정렬
    return items;
}

} // namespace stats
```

Song, Young OhK

```

int main(int argc, char** argv) {
    using namespace stats;

    // CLI: 파일명(옵션) + K(옵션)
    const std::string_view input_path = (argc > 1) ? std::string_view{argv[1]} : std::string_view{"data.txt"};
    const int K = (argc > 2) ? std::max(0, std::atoi(argv[2])) : DEFAULT_TOPK;

    // 입력 로드 (예외 대신 실패를 값으로)
    auto maybe = load_nonnegatives(input_path);
    if (!maybe) {
        std::cerr << "[error] failed to open file: " << input_path << "\n";
        return 1;
    }

    const Vec& data = *maybe;

    // 통계 계산
    const double m = mean(data);
    const double md = median(data);
    const auto freq = frequencies(data);
    const auto top = top_k(freq, static_cast<size_t>(K));

    // 출력 (빈 데이터도 안전)
    std::cout << "File : " << input_path << "\n";
    std::cout << "Count: " << data.size() << "\n";
    std::cout << "Mean : " << m << "\n";
    std::cout << "Median: " << md << "\n";
    std::cout << "Top-" << K << " frequencies:\n";
    for (const auto& [value, cnt] : top) {
        std::cout << " " << value << " -> " << cnt << "\n";
    }

    return 0;
}

```


Prompting

• 개선 포인트

- SRP 분리: 입력/평균/중앙값/빈도/Top-K를 함수로 분리 → 테스트/재사용 용이.

• 알고리즘 최적화:

- median: `std::nth_element`로 $O(N)$ 근사 (정렬 $O(N \log N)$ 회피).
- Top-K: 전체 정렬 대신 `nth_element` + 부분 정렬 → 시간 절약.
- 빈도: `unordered_map` + `reserve()`로 해시 충돌/재해시 최소화.

• 메모리 최적화:

- `reserve`, `emplace_back`, 불필요 복사 제거.
- 값 복사가 필요한 곳만 명시적으로 복사.

• 현대 C++ 스타일:

- `constexpr`, `[[nodiscard]]`, `noexcept`, 구조적 바인딩, 람다 비교자.
- RAII + 예외 안전(스마트 포인터 불필요한 설계 → 자연스런 소유권).
- 정확성/안전성: 파일 오픈 실패 처리, 빈 컨테이너 처리.

Song, Young Ohk

Prompting

- 2차 미세 최적화를 위한 "리뷰 프롬프트" 선택적으로 이용

[프롬프트 예시]

방금 생성한 after.cpp를 다음 기준으로 2차 점검/미세 최적화해줘.

- 함수별 noexcept/inline/constexpr 적용 타당성 재검토
- 파라미터 전달방식(const&, 값 이동) 점검 및 복사 최소화
- unordered_map 리저브 크기/로드팩터 힌트 가능성 검토
- nth_element/partial_sort 비교 및 경계 케이스(짝수 중앙값) 주석 보강
- 불필요한 include/using 제거, 컴파일 경고 0 보장
- 주석: 최적화 포인트의 '이유'를 더 명확히(1줄)
- 변경 diff와 전/후 빌드 플래그 제안(-march, -mtune 등)은 별도 섹션으로

Song, Young Ohk

Prompting

• Cursor Rules 활용

```
---
description: "C++ 코드 리팩토링 + 성능 최적화를 표준화합니다. SRP 분리, 알고리즘/메모리 최적화, 현대 C++ 스타일, RAII 준수."
globs:
  - "**/*.cpp"
  - "**/*.cc"
  - "**/*.cxx"
  - "**/*.hpp"
  - "**/*.h"
---
```

C++ Refactor & Optimize Rules

- ****설계 원칙****
 - SRP(단일 책임)로 모듈/함수 분리, 중복 제거, 명확한 네이밍.
 - RAII 준수: new/delete 직접 사용 금지. 표준 컨테이너/스마트 포인터 사용.
 - 인터페이스는 테스트 용이성(순수 함수, 작은 단위) 우선.
- ****알고리즘 최적화****
 - 중앙값: `std::nth_element`로 전체 정렬 회피.
 - Top-K: 전체 정렬 금지 → `std::nth_element` + 필요한 구간만 정렬/`partial_sort`.
 - 빈도: `std::unordered_map` + `reserve()` + 전위 증가(++cnt).
 - 누적합은 오버플로우 완화 위해 64-bit 누적 사용.

- ****메모리/성능 최적화****
 - 컨테이너는 `reserve()`/`shrink_to_fit()` 적절히 사용.
 - 삽입은 `emplace_*` 우선, 불필요 복사 제거, 필요 시 이동(`std::move`).
 - I/O 파싱은 실패 처리 필수. 예외 안전성 보장.
- ****현대 C++ 스타일(C++17+)****
 - `[[nodiscard]]`(무시하면 위험한 반환), `noexcept`, `constexpr`(가능 시) 적용.
 - range-based for, `auto`, 구조적 바인딩, 람다 적극 활용.
 - 불필요한 헤더 금지(`<bits/stdc++.h>` 금지), 경고 0 목표.
- ****품질 바****
 - 빠른 주석: 각 최적화 포인트 옆에 1줄 근거 설명.
 - 컴파일 플래그: `-std=c++17` 이상, `-O3`에서도 경고 없이 빌드.
 - 파일 실패/빈 컨테이너 등 경계 조건 처리.

Prompting

• Cursor Rules + 프롬프트

@파일지정

@Rule지정

Rule에 SRP 분리 + 알고리즘/메모리 최적화 + 현대 C++ 스타일로 리팩토링해줘.

요구:

- 입력/평균/중앙값/빈도/Top-K를 독립 함수로 분리하고 main은 조립만 수행.
- nth_element/partial_sort로 전체 정렬 회피, unordered_map + reserve로 빈도 계산.
- [[nodiscard]], noexcept, constexpr(가능 시) 적용, RAII 준수.
- 각 최적화 지점 옆에 1줄 근거 주석.
- 컴파일 플래그: -std=c++17 이상, -O3에서도 경고 0.
- 결과는 단일 `after.cpp`로 제시.

결과는 단일 `after.cpp`로 제시. -> 필요에 따라서 ...

Song, Young Ohk