INTELLIGENCE WAREHOUSE · 연구

Intelligence Warehouse는 토큰 소비를 96% 줄입니다

기업 AI의 상당수는 전체 프로그램을 프롬프트에 넣는 방식으로 구축됩니다. IW는 다르게 작동합니다. 오케스트레이션은 일반 Python으로 실행하고, 언어 추론이 실제로 필요한 지점에서만 모델을 호출합니다.

이 방식이 비용에 어떤 영향을 주는지 알아보기 위해 기업 업무 1,000개를 시뮬레이션했습니다. 프롬프트 기반 방식에 가능한 모든 이점, 특히 완전한 프롬프트 캐싱을 제공한 뒤에도 IW는 처리 토큰을 96% 줄이고 약 8분의 1의 비용으로 실행되었습니다.

저자 Akhil Singh · Nidhi Prabhu · Aniruddha Tammewar
발행 2026년 7월 · **방법** 고정 시드 몬테카를로, n=1,000
재현 시드 20260729

95.97%	87.54%	24.8배	7.9 → 2.2
업무당 처리 토큰 감소	기준 에이전트 캐싱 후 실행 비용 감소	토큰 처리량 차이	업무당 LLM 호출 기준 에이전트 → IW

두 수치를 함께 제시하는 이유는 서로 다른 대상을 측정하기 때문입니다. 처리 토큰은 모델이 수행하는 원시 작업량입니다. 실행 비용은 청구서에 반영되는 금액이며, 기준 에이전트의 반복 토큰 가격을 90% 줄이는 완전한 프롬프트 캐싱을 적용해 계산했습니다. IW는 두 지표 모두에서 앞섭니다.

01 프롬프트 기반 에이전트의 비용이 커지는 이유

프롬프트 기반 에이전트는 데모에서는 저렴해 보이다가 운영 환경에서 예상 밖의 비용을 발생시키기 쉽습니다. 그 이유는 작동 방식 자체에 있습니다.

한국어 번역본. 원문의 주장, 수치, 가정과 한계를 보존했습니다. 본 번역은 독립적인 검증이나 새로운 공식 심사를 의미하지 않습니다.

01 반복할 때마다 커지는 컨텍스트

일반적인 에이전트에서는 모델이 컨텍스트 창 안에서 전체 제어 루프를 실행합니다. 매 단계마다 시스템 프롬프트, 모든 도구 스키마, 검색기가 반환한 내용, 그때까지의 전체 대화를 다시 읽고 다음 단계를 만든 뒤 반복합니다. 따라서 8단계 업무의 비용은 단순히 프롬프트 8개의 비용이 아닙니다. 계속 커지는 프롬프트에 대해 매 단계 다시 비용이 청구됩니다.

프롬프트 캐싱은 도움이 되지만 근본 문제를 해결하지는 않습니다. 다시 읽는 토큰의 가격을 낮출 뿐, 모델이 그 토큰을 처리하는 일을 없애지는 않습니다. 캐시된 토큰도 매번 모델을 통과하며, 가격만 약 10분의 1이 됩니다. 하루 몇 번의 호출에서는 중요하지 않을 수 있지만 월 100만 건의 업무에서는 실제 청구 항목이 됩니다.

IW는 모델을 시스템 전체가 아닌 구성 요소 하나로 봅니다. 라우팅, 파싱, 계산, 단계 간 데이터 전달은 결정론적 Python 코드로 작성합니다. 지식은 마크다운 파일 네트워크에 저장하고, 실제 추론에 필요한 파일만 불러옵니다. 모델은 호출 그래프의 잎 노드에서, 올바른 답변에 필요한 만큼의 컨텍스트로만 호출됩니다. 이 논문이 묻는 것은 완료된 업무 한 건당 실제로 얼마나 절감되는가입니다. 비교에서는 기준 에이전트에 유리한 조건까지 반영합니다.

02 같은 기능을 구축하는 두 가지 방식

공정한 비교를 위해 두 시스템 모두 잘 구축되어야 합니다. 왼쪽은 의도적으로 약하게 만든 비교 대상이 아닙니다. 정적 접두부를 캐시하고 검색을 제한하며, 업무가 끝날 때까지 반복하는 유능한 에이전트입니다. 오른쪽은 IW 방식입니다.

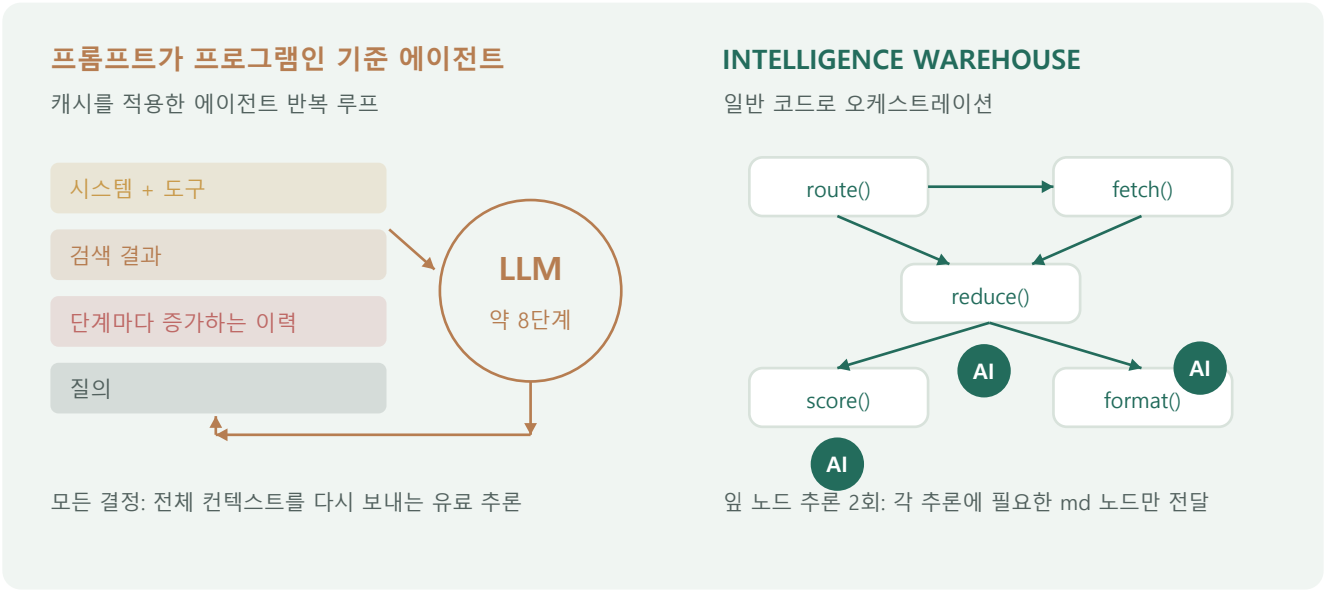


그림 1. 기준 에이전트는 매 단계 시스템·도구·검색·증가하는 이력·질의를 LLM에 재전송합니다. IW는 함수 호출로 제어 흐름을 실행하고, 필요한 잎 노드에서만 추론합니다.

두 시스템 모두 잎 노드에서 모델을 호출하므로, 차이는 그 지점 자체가 아닙니다. 차이는 그 사이를 연결하는 방식입니다. 에이전트에서는 라우팅 결정과 중간 결과가 컨텍스트 창을 통과할 때마다 토큰 비용이 반복 청구됩니다. IW에서는 이를 함수 호출 스택으로 연결합니다. CPU에서 실행되며 비용은 1센트의 일부 수준이고 토큰 계량에는 포함되지 않습니다.

03 토큰은 실제로 어디에 쓰이는가

절감은 서로 다른 세 가지 작동 원리에서 발생합니다. 이를 구분하면 하나의 대표 수치를 그대로 믿는 대신 계산 과정을 검토할 수 있습니다.

원리 1: 제어 흐름을 토큰에서 코드로 옮깁니다

에이전트에서는 모델이 컨텍스트 창 안에서 다음 행동을 결정하므로, 각 결정이 전체 작업 컨텍스트를 대상으로 하는 유료 추론입니다. IW는 이를 코드에서 결정합니다. 기준 에이전트가 한 업무에 약 8회의 모델 단계를 거치는 동안 IW는 보통 약 2회의 실제 추론만 필요합니다. 나머지는 토큰 비용을 발생시키지 않는 결정론적 코드입니다. 절감액의 가장 큰 부분이 여기서 나옵니다.

원리 2: 컨텍스트를 미리 쌓지 않고 필요할 때 불러옵니다

프롬프트 시스템은 필요할지도 모르는 모든 내용, 즉 전체 시스템 프롬프트, 모든 도구 스키마, 넉넉한 검색 청크, 몇 가지 예시를 컨텍스트 창에 넣습니다. IW는 잎 노드 추론에 실제로 필요한 마크다운 파일 한두 개와 이미 가져온 데이터만 전달합니다. 사용할지도 모르는 컨텍스트가 아니라 실제 사용하는 컨텍스트에 비용을 지불합니다.

원리 3: 이력을 재전송하지 않습니다

에이전트는 매 단계 늘어나는 대화 기록을 다시 보내므로 비용은 대략 단계 수의 제곱에 비례해 증가합니다. IW는 상태를 Python 변수에 저장하므로 각 추론의 크기가 작고 일정하게 유지됩니다. 기준 에이전트가 이력을 10분의 1 가격으로 캐싱하더라도 긴 업무에서는 재읽기 비용이 쌓입니다. IW는 애초에 이를 보내지 않습니다.



그림 2. 모델의 평균 추출값에 비례하여 그린 한 번의 호출 구성. 왼쪽은 캐시된 기준 에이전트의 한 단계로, 정적 접두부·새 검색 결과·재읽는 이력·출력으로 구성됩니다. 오른쪽은 IW 잎 노드 추론으로, 작은 지시문·지식 노드 1개·데이터 페이로드·출력으로 구성됩니다.

04 방법

이하 모든 결과는 하나의 고정 시드 시뮬레이션인 **iw-token-economics-sim.py**에서 나왔습니다. 실제 IW의 5개 산업 업무 유형에서 추출한 1,000개 업무를 실행했습니다. 원문에 제시된 수치는 손으로 입력한 예시가 아니라 스크립트 출력에서 직접 읽은 값입니다. 전체 모델은 재현 가능하며 업무별 데이터셋도 함께 공개했습니다.

IW에 불리하게 설정한 세 가지 조건

1. 기준 에이전트에는 완전한 프롬프트 캐싱을 적용했습니다. 정적 접두부는 한 번 기록한 뒤 10분의 1 가격으로 읽고, 늘어나는 이력도 같은 방식으로 캐시합니다. 원문은 이를 API에서 허용하는 가장 유리한 처리로 설명합니다.

2. IW에는 캐싱을 전혀 적용하지 않았습니다. 모든 IW 컨텍스트에 처음부터 끝까지 정상 입력 단가를 청구합니다. 이는 IW를 실제보다 불리하게 보이게 합니다.

3. 대표 수치는 금액 가중 방식입니다. 1,000개 전체 업무의 총 IW 비용을 총 기준 에이전트 비용으로 나누므로, 비싼 업무의 가중치가 가장 큽니다. 저렴하고 쉬운 업무에서의 절감만으로 평균을 부풀릴 수 없습니다.

이 세 조건을 모두 적용했으므로, 원문은 보고 수치가 상한보다 하한에 가깝다고 해석합니다.

비용 모델

하나의 비용 함수에 스위치 세 개가 있으며, 두 극단 설정이 각각 두 아키텍처입니다. 토큰 흐름은 신규 입력, 캐시 쓰기, 캐시 읽기, 출력의 네 범주로 추적하고, 공개된 Claude Sonnet급 요금으로 계산합니다.

범주	100만 토큰당 USD
신규 입력	3.00
출력	15.00
캐시 쓰기	3.75
캐시 읽기	0.30

```
비용(USD) = (신규 입력 × 3.00 + 캐시 쓰기 × 3.75
+ 캐시 읽기 × 0.30 + 출력 × 15.00) ÷ 1,000,000
```

스위치 세 개는 03절의 원리와 일대일로 대응합니다. m1은 제어 흐름을 코드로 옮기기, m2는 필요한 컨텍스트만 로딩하기, m3는 이력을 보내지 않기입니다. 모두 끄면 캐시된 기준 에이전트가 되고, 모두 켜면 IW가 됩니다. 양 끝점이 같은 함수를 공유하므로 06절의 분해는 정확합니다.

업무 포트폴리오

각 업무는 유형을 먼저 선택한 뒤 문서화된 삼각분포 범위에서 매개변수를 추출합니다. 에이전트 단계 수 S, 사용 가능한 도구 수 T, 검색 청크 수 K, 그리고 결정론적 작업이 아닌 실제 언어 추론 단계의 비율입니다. 마지막 비율이 IW 잎 노드 수 I를 정하는 핵심 변수입니다.

업무 유형	비중	단계 S	도구 T	청크 K	추론:단계
소매 재고 보충	28%	4-12	8-20	3-8	0.15-0.40
통신망 구축 배차	22%	6-18	12-30	2-7	0.12-0.35
PE 실사 질의	15%	3-9	5-14	6-18	0.25-0.60
공급망 예외 처리	20%	5-14	10-24	3-9	0.15-0.40
현장 운영 질의	15%	2-6	4-10	2-5	0.30-0.70

가장 큰 영향을 주는 가정은 **추론:단계 비율**, 즉 에이전트 단계 대부분이 결정론적이라는 주장입니다. 라우팅, 도구 선택, 파싱, 산술에는 실제로 언어 모델이 필요하지 않다는 것이 저자들의 판단이지만, 여전히 가정입니다. 07절에서는 이 비율을 4배 범위로 변화시켜 각 지점의 결과를 제시합니다.

05 결과

전체 1,000개 업무에서 IW는 처리 토큰을 **95.97%**, 실행 비용을 **87.54%** 줄였습니다. 비용 수치는 기준 에이전트에 완전한 캐싱을 적용한 뒤의 결과입니다.

두 수치가 다른 이유는 명확합니다. 캐싱은 기준 에이전트의 반복 토큰 가격을 10분의 1로 낮추므로, 달러 비용이 토큰 수보다 더 빠르게 줄어듭니다. 그래서 토큰 감소율 95.97%(24.8배)가 비용 감소율 87.54%(8배)보다 큽니다. 두 수치 모두 원문이 보고하는 정확한 결과입니다. 저자들은 청구서에 반영되며 IW에 불리한 조건에서 계산한 비용 수치를 중심으로 제시합니다.

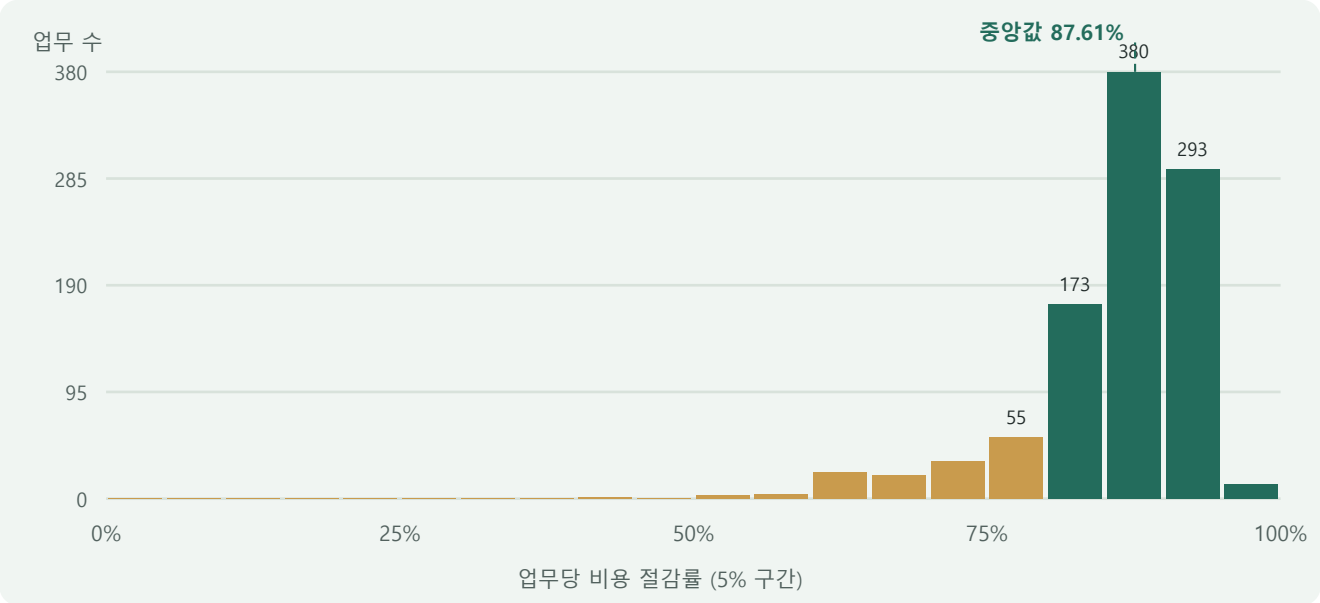


그림 3. 1,000개 업무의 건별 비용 절감률 분포. 중앙값은 87.61%이며 가운데 80%는 76.5%-92.56% 사이입니다. 왼쪽 꼬리도 실제 시뮬레이션 결과입니다. 단순한 현장 운영 질의는 제거할 제어 흐름이 적어 절감 폭이 가장 작습니다.

산업별 결과

산업별 절감률은 오케스트레이션이 얼마나 필요한지에 따라 달라집니다. 반복 루프가 길고 도구가 많은 통신망 구축 배차의 절감률이 가장 크고, 단순한 현장 운영 질의가 가장 작습니다. 예상되는 방향과 일치합니다.

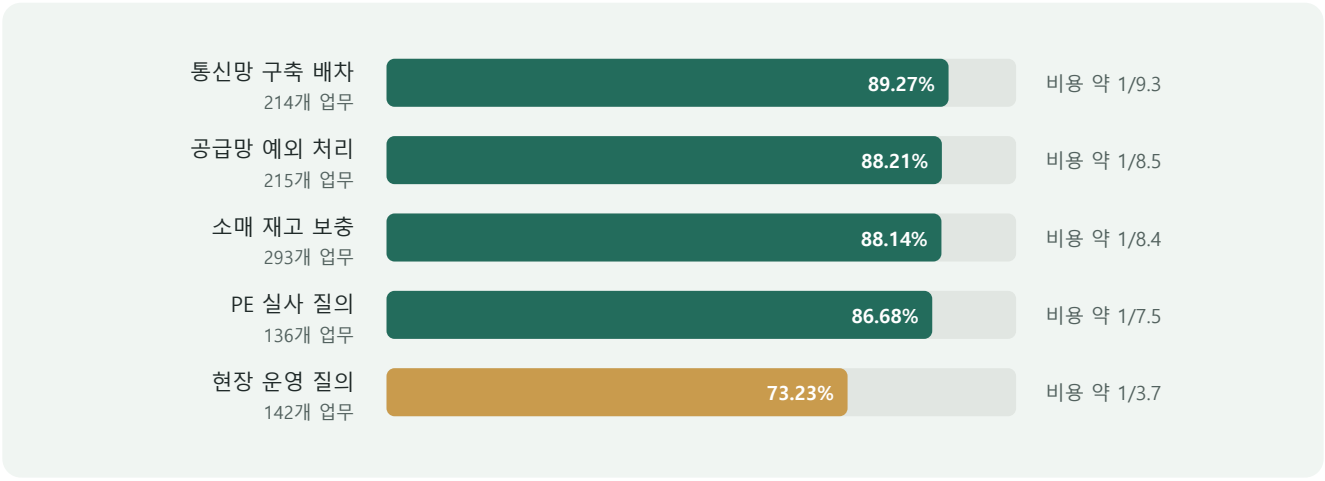


그림 4. 산업별 업무 수, 비용 절감률, 비용 감소 배수. 업무 수는 실제 1,000개 시뮬레이션 추출 결과이므로 설계 비중과 정확히 같지 않을 수 있습니다.

06 절감은 어디에서 발생하는가

대표 백분율 하나만으로는 의심하기 쉽고 검증하기 어렵습니다. 기준 에이전트와 IW는 같은 비용 함수의 두 설정이므로, 세 가지 원리의 Shapley 값으로 절감액을 분해할 수 있습니다. 이는 가능한 모든 적용 순서에서 각 원리가 만드는 기여의 평균입니다.

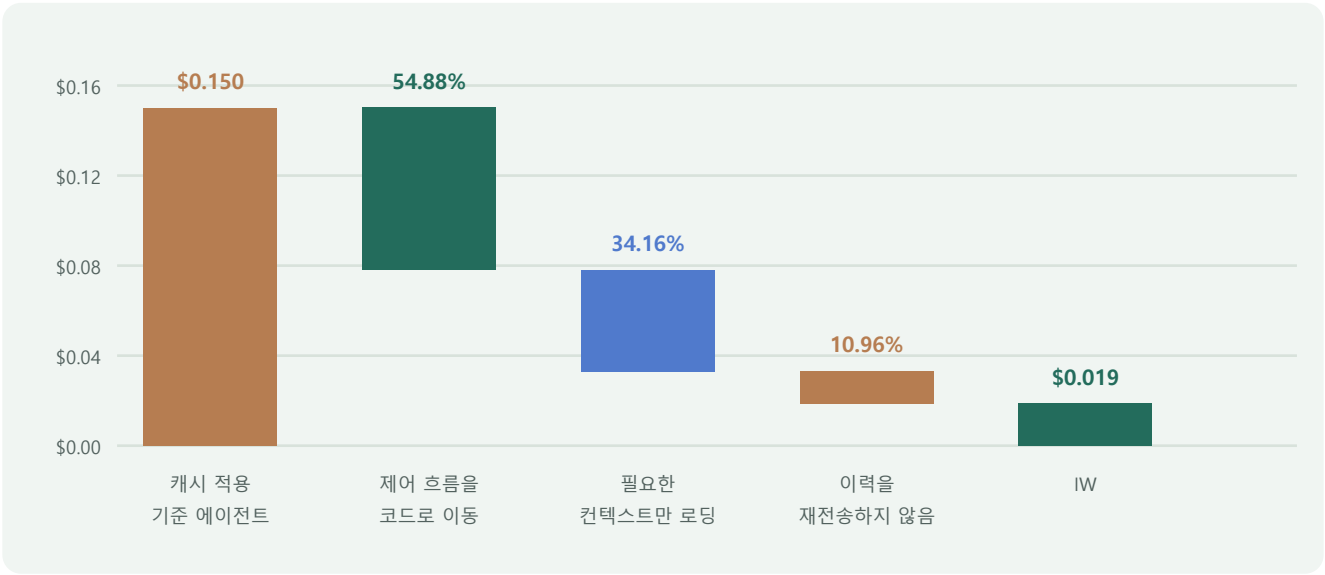


그림 5. 캐시된 기준 에이전트에서 IW로 이동할 때의 업무당 평균 비용. Shapley 방식으로 배분했으므로 적용 순서에 의존하지 않습니다. 표시된 54.88%, 34.16%, 10.96%는 **총 절감액 중 각 원리의 기여 비중**입니다. 제어 흐름의 코드화가 가장 크고, 필요한 컨텍스트만 로딩하는 효과가 그다음입니다. 이력 재전송 제거는 실제 효과가 있지만 가장 작습니다. 기준 에이전트의 캐싱이 이미 이 비용의 상당 부분을 줄였기 때문입니다.

이는 유용한 타당성 점검이기도 합니다. 이력 제거가 가장 큰 절감 요인으로 나왔다면 기준 에이전트에 캐싱을 제대로 적용하지 않았다는 뜻일 수 있습니다. 그 효과가 작게 나온 것은 실제로 캐싱을 반영했기 때문입니다.

07 가정의 스트레스 테스트

이견이 제기되기 쉬운 두 가지 가정을 변화시키며 결과를 확인했습니다.

추론:단계 비율

핵심 가정은 에이전트 단계 대부분이 결정론적이라는 것입니다. 이 가정이 틀렸을 가능성을 확인하기 위해 모든 업무의 IW 추론 수를 추정치의 절반부터 두 배까지 조정하고 전체 포트폴리오를 다시 실행했습니다.

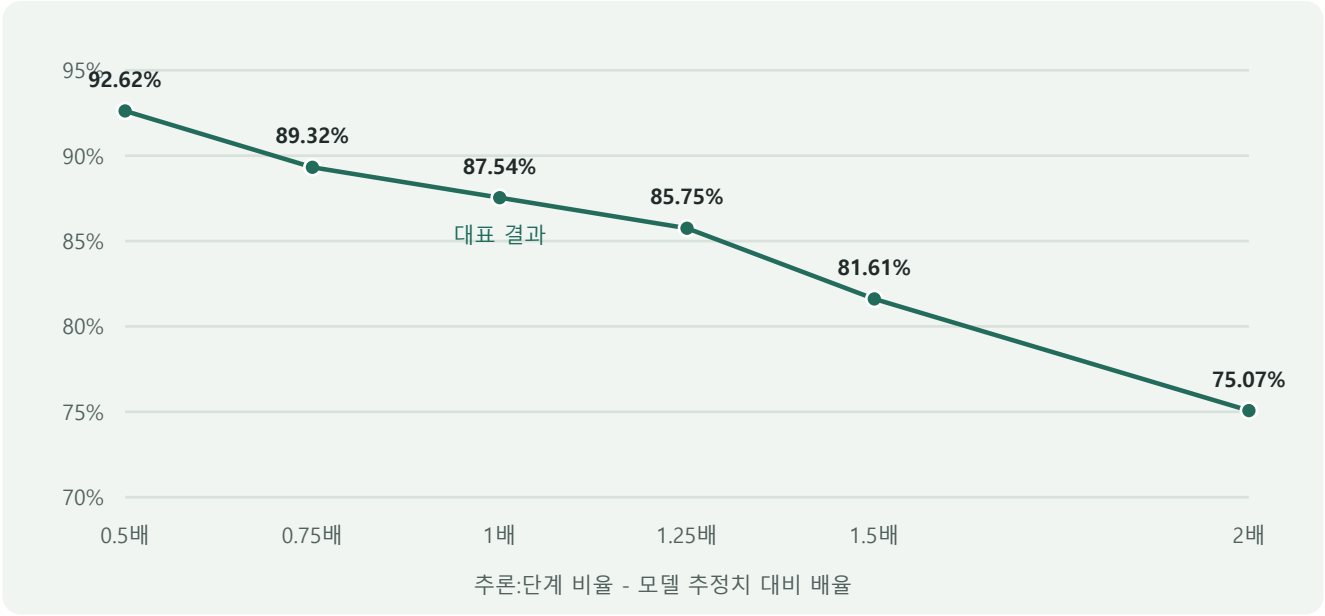


그림 6. 추론:단계 비율을 0.5배에서 2배로 바꾼 포트폴리오 비용 절감률. IW가 모델링한 것보다 실제 추론을 두 배 필요로 하더라도 실행 비용은 75.07% 감소합니다. 원문은 결과가 매우 좁은 가정 범위에만 의존하지 않는다고 해석합니다.

기준 에이전트의 캐싱

대표 결과는 이력을 포함해 기준 에이전트에 가장 유리한 캐싱을 제공합니다. 실제 배포에서는 검색 결과가 단계마다 컨텍스트를 바꾸기 때문에 이력이 안정적으로 캐시 가능한 접두부가 되지 못하는 경우가 많습니다. 그 경우 기준 에이전트 비용이 커지고 IW의 우위가 확대됩니다.

기준 에이전트의 캐싱 조건	비용 절감률
이력 전체 캐시 - 대표 결과, 기준 에이전트에 가장 유리	87.54% · 하한
이력 캐시 없음 - 원문이 더 일반적이라고 보는 운영 조건	90.4% · 더 가능성 높은 경우

원문은 두 값 중 더 작은 수치를 보고합니다. 더 현실적이라고 보는 경우의 절감률은 더 큽니다.

08 규모가 커질 때의 의미

업무 한 건의 비용은 약 0.15달러와 0.0187달러입니다. 기업은 한 건이 아니라 수백만 건을 실행하므로 차이는 업무량에 비례해 커집니다. 원문은 모델을 바꾸는 것이 아니라 호출 빈도와 컨텍스트 크기만 바꾸므로 모델과 출력 품질은 그대로라고 설명합니다.

월간 업무량	기준 에이전트 / 월	IW / 월	연간 절감액
10만 건	15,009달러	1,871달러	157,660달러
100만 건	150,090달러	18,707달러	1,576,596달러
1,000만 건	1,500,905달러	187,075달러	15,765,959달러

같은 모델, 같은 답변. 원문이 제시하는 차이는 오케스트레이션이 실행되는 위치입니다.

09 비용과 한계를 함께 보기

장점만 나열하는 논문은 충분하지 않습니다. IW가 부담하는 비용과 이 분석이 틀릴 수 있는 지점을 함께 제시합니다.

기준 에이전트에는 없지만 IW가 부담하는 비용

그래프 구축. 엔지니어가 마크다운 네트워크와 Python 함수를 미리 작성합니다. 실제 비용이 들지만 제대로 된 에이전트도 무료로 만들 수 있는 것은 아닙니다. 프롬프트 엔지니어링, 평가 도구, 검색 파이프라인 등이 필요하므로 공정한 비교는 IW와 비용 0의 비교가 아니라 두 구축 비용의 차이입니다.

연산. 모든 함수 호출은 CPU를 사용합니다. 원문은 클라우드 요금 기준으로 업무당 1센트의 일부 수준이며, 대체하는 토큰 비용보다 3-4자릿수 규모로 작다고 설명합니다.

사전 설계 작업. IW는 구축 단계에서 무엇이 결정론적이고 무엇이 모델을 실제로 필요로 하는지 구분하도록 요구합니다. 에이전트가 실행 중에 알아서 판단하게 하는 것보다 일이 많습니다. 그 대신 위의 비용 절감과 함께, 열린 에이전트 루프가 제공하기 어려운 결정론성과 감사 가능성을 얻습니다.

이 분석이 틀릴 수 있는 지점

추론:단계 비율은 모델링 선택입니다. 모든 결과가 의존하는 핵심 가정입니다. 07절은 0.5배부터 2배까지 결과를 보여 줍니다. 원문은 추론 비중이 약 0.35배 아래일 때 두 아키텍처가 수렴하기 시작한다고 서술합니다. 그 경계를 아는 것이 중요합니다.

토큰 크기는 한 고객의 로그가 아니라 문서화된 범위에서 가져왔습니다. 현장에서 직접 측정한 값이 아니라 대표값입니다. 실제 배포에서는 자체 수치로 스크립트를 다시 실행해야 하며, 스크립트는 이를 반영할 수 있도록 작성되었습니다.

측정 대상은 비용이지 역량이 아닙니다. 일부 문제는 에이전트가 열린 반복 루프를 탐색해야 합니다. 저자들의 주장은 대부분의 기업 업무가 그런 형태는 아니라는 것입니다. 추론 지점이 몇 개 있는 알려진 프로세스라면 에이전트 방식의 가격을 지불하는 것은 필수가 아니라 선택이라는 주장입니다.

10 재현하기

원문의 모든 수치는 실제 시뮬레이션 실행 결과입니다. 고정 시드를 사용하므로 다른 컴퓨터에서도 같은 값을 얻을 수 있으며, 단순한 예시 수치가 아닙니다.

```
python3 iw-token-economics-sim.py
→ iw-token-economics-data.json (모든 집계 수치)
→ iw-token-economics-tasks.csv (1,000개 업무의 행별 데이터)
```

sim · iw-token-economics-sim.py: 외부 의존성이 없는 Python 표준 라이브러리 스크립트. 하나의 비용 함수, 세 스위치, Shapley 분해, 두 민감도 분석을 포함합니다.
data · iw-token-economics-data.json: 원문 웹 페이지가 그림에 사용하는 집계 데이터.
tasks · iw-token-economics-tasks.csv: 매개변수, 두 비용, 두 절감률을 포함한 전체 업무별 표.
seed · 20260729: 값을 바꾸면 개별 업무 분포는 조금 바뀌지만 집계값은 거의 달라지지 않는다고 원문은 설명합니다.

자체 환경에서 검증하려면 공급업체 요금, 자체 업무 구성, 실행 추적에서 얻은 토큰 크기로 바꾸어 실행하십시오. 오케스트레이션을 토큰보다 코드에서 실행하는 편이 유리하다는 대표 결론은 특정 추출값보다 아키텍처에 달려 있다는 것이 원문의 주장입니다.

한 문장으로 요약한 원문의 주장. 오케스트레이션을 프롬프트에서 코드로 옮기면, 캐싱상 모든 이점을 부여한 기준 에이전트와 비교하여 1,000개 업무에서 토큰 처리량은 95.97%, 실행 비용은 87.54% 감소합니다. 같은 모델과 답변을 다른 위치에서 실행하며, 하나의 고정 시드 스크립트로 재현할 수 있습니다.

원문 및 재현 자료
저자: Akhil Singh, Nidhi Prabhu, Aniruddha Tammewar · 2026년 7월
원문: intelligencewarehouse.com/IW-Token-Economics-Whitepaper.pdf
연구 페이지: intelligencewarehouse.com/go/token-economics
원문 그림 데이터: iw-token-economics-data.json · 시드 20260729 · n=1,000 · Claude Sonnet급 요금

번역 주: 0.35배 경계 문장은 원문 표현을 그대로 보존했습니다. 이 번역은 해당 경계나 출력 품질 동등성을 별도로 검증한 결과가 아닙니다. 그림은 원문에 포함된 데이터와 동일한 공개 연구 페이지의 수치로 다시 그렸습니다.