Combination Method (Ko Method) — A Discovery.

An LLM workflow pattern for precision design tasks

Discoverer
Daewon Ko

Discovery date
2026-06-01

Origin context
deLight house Phase 03 — CAD stair design (interior feature stair, 2,500 mm rise)

First validation
Same day. 14R × 178.57 mm × 280 mm tread combination passed all numeric and semantic thresholds.

Refinement
Same day, subsequent dialogue. Agent routing and compound threshold pattern formalized.

Principle enrichment
2026-06-02 (v1.1). Core principle “Aggregation + Combination” articulated. Case-study trace + derived sub-principles added.

Co-validator

Claude (Anthropic Sonnet 4.6)

TL;DR

LLM에게 전체 도면 또는 정밀 설계를 단일 호출로 수행하도록 지시한 경우, 결과의 정확도가 일관되게 부족함이 관찰된다. 본 문서는 그 한계를 우회하기 위해 고안한 워크플로 패턴을 정리한 기록이다.
LLM을 generator가 아니라 *aggregator*와 *refiner*로 사용한다. 조합(Combination)과 필터링은 외부 알고리즘 또는 더 가벼운 모델에 위임한다.
본 방법론의 본질은 두 단계로 요약된다 — **(A) 데이터 집약 + (B) 조합 사용**. 도메인의 분산된 지식을 카테고리로 압축한 뒤, 그 카테고리들을 알고리즘적으로 조합하여 후보를 탐색한다.
본 문서는 완성된 이론이 아닌, 실제 작업 중 발견된 패턴의 첫 기록이다. 다른 도메인에 적용하면서 점진적으로 정교화될 예정이다.

---

1. 어떻게 발견하게 되었는가

deLight house 건축 컨셉 프로젝트의 일환으로 계단 평면 설계 작업 중, LLM에게 단일 호출로 전체 평면 생성을 지시한 경우 결과의 정확도가 일관되게 부족함을 확인하였다. 단높이, 단너비, 랜딩 폭 등 핵심 치수가 의도된 비례에 부합하지 않거나, 수치 자체가 입력 제약을 충족하지 못하는 경우가 반복 발생하였다.
이에 다음과 같은 단계적 지시 방식을 적용한 결과, 일관된 출력이 확보되었다:

“비슷한 설계들을 학습해줘”
“필요한 변수 카테고리로 나눠줘”
“그 변수들의 조합 중 임계치 통과하는 것 찾아줘”
“그 후보를 내 요구사항에 맞게 다듬어줘”
단일 호출로 처리되지 못하던 작업을 단계별로 분해하고, 각 단계의 인지적 복잡도에 적합한 도구를 매칭하는 방식으로 성공적으로 수행되었다. **핵심 관찰**: LLM의 강점(인용, 다듬기, 문맥 판단)과 약점(정밀 수치, 고차원 조합 탐색)을 라우팅 차원에서 분리할 경우, 약점을 우회하면서 강점만 선택적으로 활용 가능하다.

---

2. 방법론 — 원리 · 단계 · 사례

2.1 핵심 원리: 데이터 집약 + 조합 사용

본 방법론의 본질은 두 단계로 요약된다.
Phase A — 데이터 집약 (Aggregation) 도메인의 분산된 지식을 구조화된 카테고리로 압축한다. LLM이 자체 training 분포에서 유사 사례, 표준, 정상 범위를 인출하고, 이를 의미적 클러스터로 분할하여 변수 ontology를 도출한다. 인간이 직접 수행하면 수십 시간의 문헌 조사가 필요한 작업을, LLM의 internalized prior를 활용하여 단일 호출로 압축한다.

Phase B — 조합 사용 (Combination) 도출된 카테고리의 노드들을 조합 알고리즘으로 탐색하여 후보해를 생성하고, 두 단계 임계치(수치적·의미적)로 걸러낸다. 조합 탐색 자체는 LLM이 아닌 결정론적 알고리즘(Python)에 위임한다. 의미적 평가만 lightweight LLM에 위임한다.

이 두 단계의 분리가 핵심이다. LLM 단일 호출로 "전체 답"을 생성하려는 시도는 다음 네 가지 이유로 실패한다.

(1) Autoregressive 한계 — LLM의 토큰 예측은 직전 문맥에 강하게 좌우되며, 다변수 정밀 수치를 동시에 만족시키는 출력을 생성하기 어렵다. 한 변수를 고정하면 다른 변수가 어긋난다. 예: "단높이 178.57 mm, 단너비 280 mm, 14단, 총 2,500 mm" 4개 조건을 단일 호출로 만족시키려 할 때, LLM은 종종 3개를 만족시키나 마지막 1개가 어긋난다.

(2) 정밀 수치의 tokenization 손실 — "178.5714"와 같은 분수가 토큰 시퀀스로 변환되는 과정에서 정확도가 보장되지 않는다. 산술 자체가 attention-based로 수행되므로 결정론적 계산이 보장되지 않는다.

(3) 카테고리 구조 부재의 비효율 — 도메인의 변수 구조를 명시적으로 식별하지 않으면, LLM은 유의미한 후보를 좁히지 못하고 평균적·안전한 출력으로 수렴한다 (mode collapse). 결과: 항상 "흔한" 답을 내고 사용자의 특수 요구에 fit되지 않는다.

(4) 조합 탐색의 인지적 부담 — 6개 변수 × 각 5개 값 = 15,625개 조합을 LLM이 직접 탐색하는 것은 cost·정확도 모두 비효율이다. 결정론적 알고리즘이 압도적으로 적합하다.

따라서 본 방법론은 LLM이 강한 작업(인용, 클러스터링, 다듬기, 문맥 판단)과 LLM이 약한 작업(다변수 정밀 산술, 고차원 조합 탐색)을 라우팅 차원에서 명시적으로 분리한다.

2.2 4단계 + Human Gate

Step 1 · Data Acquisition (학습)

사용자의 요구 조건과 환경 명세를 입력으로 받아, LLM이 자체 training 지식에서 유사 사례, 표준, 정상 범위를 인출한다. 도메인 priors를 다음 단계로 전달하는 데이터 채집 단계.
작동 원리 — LLM의 training corpus는 수많은 도메인의 implicit prior distribution을 내장하고 있다. 이 단계는 그 distribution에서 task-relevant subset을 명시적으로 추출하는 *amortized prior elicitation*에 해당한다. 직접 코딩으로는 수십~수백 시간의 문헌 조사가 필요한 작업을 단일 호출로 압축한다.

적정 모델: Sonnet 4.6. 새로운 도메인이거나 cross-domain 지식이 요구되는 경우 Opus를 일시적으로 사용.

Step 2 · Decomposition (카테고리 분리)

본 단계는 두 sub-step으로 분리된다. 둘은 인지적 복잡도가 상이하다.
작동 원리 — 도메인의 가변 차원을 *axis*로 분리하는 것은 후속 조합 탐색의 search space를 효율적으로 정의하기 위해 필수적이다. 카테고리 ontology가 있으면 random sampling의 차원이 명시되며, "어디까지 변경 가능하고 어디는 고정인가"라는 design freedom이 식별된다. 이 명시적 분리가 없으면 LLM은 모든 변수를 동시에 추론하려다 mode collapse한다.

Step 2a · Category Discovery

도메인이 새로워 변수 구조 자체가 미정인 경우, 데이터에서 의미적 클러스터를 추출하여 카테고리 구조를 도출한다. 추론 비중이 높은 작업이므로 Sonnet 또는 Opus가 요구된다.
왜 reasoning depth가 요구되는가 — 카테고리 발견은 비지도 학습에 가까운 작업이다. 의미적으로 응집된 클러스터를 형성하기 위해서는 도메인 ontology에 대한 깊은 contextual reasoning이 요구된다.

Step 2b · Category Application

기 정의된 ontology에 데이터를 분류하여 채우는 작업. 추론 부담이 낮으므로 Haiku로 충분하다.
왜 Haiku로 충분한가 — ontology가 정해진 상태에서의 분류는 단순 mapping task이다. Reasoning depth 요구가 낮으므로 lightweight model로 충분하다.

Ontology Caching (Learned Routing) — 핵심 아이디어. 한 번 도출한 카테고리 구조를 메모리에 저장하면, 동일 도메인의 후속 작업에서 Step 2a를 생략하고 2b만 실행 가능하다. 시스템이 시간에 따라 점진적으로 효율화되는 cumulative learning 효과가 발생한다.
이론적 위치 — Ontology caching은 *amortized inference*의 한 형태이다. 일반 LLM 호출에서는 매번 동일한 도메인 구조를 reasoning부터 시작하는 비용이 발생하나, 카테고리를 메모리에 저장하면 reasoning 결과가 reuse된다. 이는 meta-learning에서의 "few-shot에서 zero-shot으로의 전환"과 구조적으로 유사하다.

Step 3 · Combinatorial Sampling + Compound Threshold

카테고리 노드를 무작위로 조합하여 후보 해를 생성하고, 두 단계 임계치로 필터링한다.
작동 원리 — 카테고리 ontology가 정의되면, 도메인의 design space는 cartesian product $N_1 \times N_2 \times \cdots \times N_k$로 정의된다. 이 공간에서 후보를 random sampling 또는 grid-search 방식으로 enumerate하고, validity criteria에 따라 필터링한다. 핵심은 *cheap-first, expensive-later* 순서로 필터를 적용하는 것이다.

Step 3a · Numeric Filter (수치적 필터)

Python을 통한 결정론적 검증으로 정확한 등식, 부등식, 코드 준수, 범위 체크를 수행한다. LLM 호출 없이 1,000개 후보를 1초 이내에 50개 수준으로 축소 가능하다.

왜 결정론적 검증이 첫 단계인가 — 수치 검증(R × N = total_rise, 2R + T ∈ [580, 660] 등)은 모호함 없이 boolean 평가가 가능하다. 동일 검증을 LLM이 수행하면 (a) hallucination 가능성, (b) 비용 1000× 증가, (c) 속도 저하가 발생한다. *결정론 가능한 작업은 결정론에 맡긴다*는 것이 본 단계의 settling principle.

정보이론적 근거 — 첫 단계가 후보 공간을 95% 줄이면, 두 번째 단계의 LLM 호출 비용도 95% 절감된다. 이는 단순 비용 절감을 넘어, expensive operation이 informed candidates에만 적용되도록 함으로써 의사결정 품질을 동시에 향상시킨다.

Step 3b · Semantic Filter (의미적 필터)

Haiku의 의미적 판단으로 미적·논리적 정합성을 평가한다. 1차 필터를 통과한 50개 후보를 5~10개 수준으로 선별한다.
왜 Haiku가 적합한가 — “이 조합이 자연스러운 비례인가?”, “사용자 의도와 맥락이 맞는가?” 같은 의미적 판단은 reasoning depth 요구가 중간 정도이다. Sonnet으로 가능하지만 over-spec이며, 50 candidates × Sonnet 호출은 비용 낭비.

이중 필터의 의의 — 수치만으로는 잡히지 않는 의미적 오류가 있다. 예: 단높이 200 mm와 단너비 200 mm 조합은 수치적으로 Blondel을 통과하지만 사용자 의도(comfort, sculptural intent)와 어긋날 수 있다. 의미 필터가 이를 잡아낸다. 한편, 의미 필터에 모든 1,000 후보를 거치면 비용·시간 모두 과도. 따라서 수치 1차 + 의미 2차 = compound threshold.

Step 4 · LLM Refinement (다듬기)

임계치를 통과한 후보를 사용자의 정확한 요구사항에 맞춰 서술적·문맥적으로 정합화한다. 누락된 디테일, 라벨링, 단위, 설명을 보완하는 단계. Sonnet 4.6 적정.
작동 원리 — 임계치를 통과한 후보는 수치·의미 양면에서 valid하나, narrative·문맥 측면에서 사용자의 정확한 요구사항에 fit되지 않을 수 있다. 이 단계는 valid candidate를 사용자 의도에 맞춰 narrative-level coherence를 부여한다. 이 작업은 LLM이 압도적으로 잘 수행하는 영역이다 (text fluency, contextual adjustment, labeling).

Step 5 · Human Confirmation Gate (사용자 컨펌)

이 게이트가 부재할 경우, LLM이 자의적으로 결과를 확정하게 된다. 잘못된 결과가 그대로 진행될 위험이 가장 큰 지점이므로, 인간 결재를 명시적으로 강제하는 것이 핵심이다.
왜 자동화하지 않는가 — LLM이 자의적으로 결과를 확정하면, 잘못된 출력이 그대로 downstream에 적용된다. Human-in-the-loop는 단순 검토를 넘어 *autonomy boundary*를 명시적으로 정의하는 control mechanism이다.

2.3 Case Study — deLight house 인테리어 계단 (step-by-step trace)

본 방법론을 실제 적용한 첫 사례의 결정 과정 추적.

Input — 사용자 요구: 1F → 2F 인테리어 feature stair. 총 상승 2,500 mm. switch-back 형태. 평면 footprint 2,880 × 2,400 mm.

Residential rise per riser: 150-200 mm typical
Tread (going): 250-300 mm typical
Blondel formula: 2R + T ∈ [580, 660] for comfort
Pitch: 25°-40° comfortable for residential
Switch-back convention: 7+7 with mid-landing, or 6+1+7 with winder corner
Sculptural reference candidates: Phillips Exeter Library (Louis Kahn), 14R proportion
Step 2 · Decomposition — 변수 카테고리 도출:
N (riser 개수): {12, 13, 14, 15, 16} *(discrete)*
T (tread, mm): continuous [220, 300]
분할 방식: {6+1+7 winder, 7+1+6 winder, 7+7 mid-landing, 6+8 asymmetric}
Landing geometry: {squircle R=500 n=5, square, rectangular}
Stringer depth (mm): [180, 280]
Step 3a · Numeric Filter (Python) — N에 대한 enumerate + R, T 도출 + Blondel·pitch 평가:
N
R = 2500/N
T (mm)
Blondel 2R+T
Pitch
Pass?
12
208.33
280
696.67
36.65°
☒ Blondel 상한 초과
13
192.31
280

664.62
34.49°
☒ Blondel 상한 초과
14
178.57
280
637.14
32.51°
✓
15
166.67
280
613.33
30.78°
✓
16
156.25
280
592.50
29.17°
✓
→ 14R, 15R, 16R 통과. 5개 → 3개 후보로 압축.

Step 3b · Semantic Filter (Haiku 위임) — 의미적 판단:

14R: traditional architectural proportion. Phillips Exeter Library의 Kahn 비례와 정렬. sculptural intent fit.
15R: 한 단당 rise가 낮아 elevation에서 step density가 높음. less generous per-step proportion.
16R: 너무 dense. Sculptural intent 약화.
→ **14R 선정**. semantic intent 일치도 최상.
Step 4 · Refinement (Sonnet) — 14R × 178.57 × 280 + switch-back 7+7 + squircle landing (R=500, n=5) 조합에 narrative 부여. DXF 좌표 생성, 평면도 라벨링, ontology 메모리 caching.

Step 5 · Human Gate — 사용자 컨펌: “14R로 가자” 확정. v1 commit.

비용 비교 (this case)

Approach	Estimated calls	Estimated cost
Single-pass Opus (정밀 도달까지 반복)	5-10 호출	$1.00 - $3.00
Combination Method (this run)	Sonnet 3회 + Haiku 1회 + Python 1회	$0.05 - $0.10

절감

≈ 95%
비용 절감을 넘어 (a) **재현성** 보장 (같은 input → 같은 output), (b) **디버깅 가능성** (어느 단계에서 어긋났는지 추적 가능), (c) **후속 작업 효율화** (ontology cache로 동일 도메인 재작업 시 2a 생략).

---

3. Agent Routing Architecture

Stage	Model / Tool	Why
Step 1 · Data Acquisition	Sonnet 4.6	Domain pattern recall
Step 2a · Category Discovery	Sonnet 4.6 (or Opus)	Unknown structure → reasoning
Step 2b · Category Application	Haiku	Known ontology, cached
Step 3a · Numeric Filter	Python (no LLM)	Deterministic equations
Step 3b · Semantic Filter	Haiku	Aesthetic / logical judgment
Step 4 · Refinement	Sonnet 4.6	Narrative, fit to user request

Step 5 · Confirmation Gate
Human
Prevents auto-commit drift

Routing Flow

Input: 조건 + 환경
  ↓
Step 1. 학습 → Sonnet 4.6
  ↓
Step 2a. 카테고리 발견 → Sonnet 4.6 (새 도메인)
Step 2b. 카테고리 적용 → Haiku (캐시된 ontology)
  ↓
Step 3a. 수치 필터 → Python (LLM 없음)
Step 3b. 의미 필터 → Haiku
  ↓
Step 4. 다듬기 → Sonnet 4.6
  ↓
Step 5. 사용자 컨펌 → Human

Configuration
Per Run (est.)
Relative
All-Opus (single-pass generation)
$0.50 - $2.00
100%
All-Sonnet (single-pass generation)
$0.15 - $0.50
≈ 30%
Combination Method (routed)
$0.05 - $0.20
≈ 10%
단순한 비용 절감을 넘어, 정확도, 재현성, 디버깅 가능성 모두 향상된다. 각 단계가 격리되어 있으므로, 잘못된 출력이 어느 단계에서 발생했는지 추적 가능하다.

4. Applicability (적용 가능 영역)

LLM single-pass 정확도가 떨어지는 정밀 작업 중, 다음 조건을 만족하는 경우 적합하다:
수치 또는 의미 임계치로 후보를 평가할 수 있음
변수가 카테고리로 분해 가능함
정답이 다수 존재하여 조합 탐색에 의미가 있음
단답이 아닌 구조화된 산출물이 필요함

Verified Domains

CAD · 정밀 수치 설계 (검증 완료: deLight house 인테리어 계단 14R × 178.57 × 280 mm)

Candidate Domains (미검증)

데이터 모델링 / 스키마 설계
UI 그리드 시스템 / 레이아웃 토큰
가구, 발코니 등 다변수 제약 충족 설계
Ontology 노드 매핑 및 유사도 기반 분류

Not Applicable (적용 부적합)

정답이 하나로 정해진 단답 작업 — 직접 질의가 더 효율적
창의적 자유 작업 — 생성 자체가 목적인 경우
변수가 1~2개인 단순 케이스 — 오버엔지니어링

5. Derived Sub-Principles

본 방법론의 핵심 원칙 — *데이터 집약 + 조합 사용* — 에서 다음 6개 부수 원칙이 도출된다. 이 원칙들은 본 case study에서 발견되었지만, 정밀성·구조성을 요구하는 모든 LLM workflow 설계에 일반화 가능하다.

Sub-Principle 1 · LLM as Aggregator-Refiner, Not Generator

LLM을 "처음부터 끝까지 답을 생성하는 도구"가 아닌, **지식을 압축(aggregate)** 하고 **결정된 답을 다듬는(refine)** 도구로 사용한다. 실제 *탐색·결정*은 외부 알고리즘에 위임한다. LLM의 강점은 distribution 내 sampling과 contextual coherence이며, 약점은 multi-variable optimization과 precision arithmetic이다. 강점만 선택적으로 호출하는 라우팅이 본 원칙의 핵심.

Sub-Principle 2 · Complexity-to-Capability Matching (Model Routing)

작업의 인지적 복잡도에 모델의 능력을 맞춘다. 모든 단계에 max-capability model (Opus)을 쓰는 것은 비용 낭비이며, 모든 단계에 min-capability model (Haiku)을 쓰면 정확도가 부족하다. **각 단계의 reasoning depth requirement를 식별하고 그에 맞는 모델로 routing**한다. 이는 단순 cost optimization이 아닌 quality-cost frontier 최적화이다.

Sub-Principle 3 · Determinism Where Possible

검증 가능한 등식·부등식·범위 체크는 LLM이 아닌 Python(또는 결정론적 도구)에 맡긴다. 결정론 가능한 작업을 확률적 도구에 맡기는 것은 (a) 불필요한 hallucination 위험, (b) 비용 증가, (c) 디버깅 불가능성을 야기한다. *"if it can be a function, make it a function"*.

Sub-Principle 4 · Compound Threshold (Cheap-First Filtering)

필터를 둘 이상 직렬 적용할 때, **cheap-first, expensive-later** 순서로 배치한다. 결정론(1차) → 의미(2차) → narrative(3차). 각 단계가 후보를 좁히므로, expensive operation은 informed candidates에만 적용된다. 정보이론적으로 entropy reduction을 단계적으로 수행하는 구조.

Sub-Principle 5 · Cumulative Memory (Ontology Caching)

한 번 발견한 카테고리 구조는 메모리에 저장하여 후속 작업에서 재사용한다. 매번 reasoning을 처음부터 시작하지 않으므로 시스템이 시간에 따라 효율화된다. *Learned routing* 패턴. Amortized inference 또는 meta-learning의 한 instantiation으로 볼 수 있음.

Sub-Principle 6 · Human-at-Confirmation, Not Generation

인간 결재는 *마지막 단계*에서만 강제한다. 생성·필터링·다듬기는 자동화하되, 결과 commit은 인간이 승인. autonomy boundary를 명시. Alignment 측면에서, 자동화 가능한 단계와 인간이 통제해야 할 단계를 의도적으로 구분하는 control architecture.

일반화: LLM Workflow 설계 패턴

이 6개 부수 원칙은 deLight house CAD 작업에서 발견되었지만, 정밀성·구조성을 요구하는 다음 도메인에 적용 가능성이 예측된다:
데이터 모델링 / 스키마 설계 — 컬럼 후보 generation + 정합성 filter + naming refinement
UI Layout — 그리드 토큰 조합 + 비례 filter + design refinement
Config / Hyperparameter tuning — 조합 + validation filter + final selection
Document drafting — 단락 구조 후보 + coherence filter + narrative refinement
Recipe / Formulation design — ingredient 조합 + nutritional filter + taste refinement
Curriculum / Syllabus design — 모듈 조합 + 학습목표 filter + 흐름 refinement
Code generation (precision-sensitive) — function signature 조합 + type/contract filter + refactor refinement
각 도메인에서 "데이터 집약 + 조합 사용"이라는 핵심 패턴이 어떻게 instantiate되는지가 후속 검증 과제이다.

---

6. Known Limitations

본 방법론의 미해결 약점. 후속 연구 및 실험을 통해 보완해야 할 부분.
순수 랜덤 샘플링은 변수 5개 이상에서 비효율적 — 조합 폭발. 변수별 prior range 좁히기 가이드가 필요.
임계치 정의 자체가 도메인 의존적 — 무엇이 "충분한지" 정량화 가능해야 작동.

현재 버전은 매 작업이 독립적 — Ontology caching이 일부 해결하나, 본격적 cumulative learning은 미구현.
단일 목적함수 — 다목적 최적화(미적, 기능적, 코드 준수 등) 가중치 미지원.
에이전트 간 context loss 가능성 — 단계 분리 시 정보 손실. 정교한 패스워크 설계 요구.

7. Open Questions

다음 단계 탐구 과제:
임계치의 동적 조정 메커니즘 (Adaptive threshold based on convergence)
Multi-objective 가중치를 사용자 의도에서 추출하는 방법
Ontology caching의 일반화 한계 — Cross-domain transfer 가능성
변수 5개 이상에서 효율적인 priors 추출 방법 (베이지안 prior, 도메인 knowledge graph 등)
Refinement 단계의 LLM hallucination 검출·교정

8. Closing

본 문서는 발견의 첫 기록이다. 발견의 흐름을 보존하면서, 다른 도메인의 적용 사례가 축적되면 follow-up note로 확장할 예정이다.
Combination Method (Ko Method)는 LLM workflow 설계에서 “LLM은 generator가 아니라 aggregator + refiner”라는 관점을 정밀 설계 작업에 적용한 사례 연구이다. 이 관점이 정밀성을 요구하는 다른 도메인에도 일반화 가능한지 여부가 다음 단계 검증의 핵심이다.

본 v1.1 개정에서 핵심 원리(“데이터 집약 + 조합 사용”)의 이론적 근거와 deLight house 사례 추적, 6개 부수 원칙이 추가되었다. 후속 v1.2 이상에서는 미검증 도메인 적용 시도가 case study로 누적될 예정이다.

Combination Method (Ko Method) — A Discovery. v1.1 · 2026-06-02 · Daewon Ko