

MORDOR / PIPELINE POD

Visual Summary

내용시각화 Summary 고도화

개념, 일정, 작업 상태, 구현물 로직 통합 정리

생성일	2026-08-29
오너	정원석 (MORDOR)
규약	Visual Summary v0.5 (2026-08-02 확정)
현재 단계	4단계 진행 중. 2026-09 말 dev/stg, 10월 중순 production 판정 목표
원문 SSoT	miridih/mordor-wiki : docs/projects/mordor/visual-summary/
구현 저장소	miridih/aippt-prisonbreak : src/lib/visual-summary, src/app/visual-summary

이 문서는 원문 문서들과 구현 코드를 읽어 조합한 파생물이다. SSoT 가 아니므로 값이 어긋나면 원문이 맞다.
원문 폴더의 README, TODO, spec, design, plan, reference 및 저장소 코드를 근거로 한다.

목차

개요: 이 문서가 무엇이고 Visual Summary 가 무엇인가	4
이 문서의 목적과 한계	4
배경: 왜 Summary 를 고도화하는가	4
한눈에 보기	5
5단계 로드맵 요약	5
용어 사전	7
사람과 소유 경계	8
이후 절 안내	8
규약: Visual Summary 스키마 v0.5	10
무엇을 표현하는 형식인가	10
설계 원칙 셋	10
structure, params, decoration: 셋의 역할 구분	10
필드 전체 구성	11
YAML 예시 셋	17
Capability(성립 조건) 테이블의 구조	18
fallback(강등) 사다리	18
복잡도 레벨 셋: 질의가 어디까지 복잡해질 수 있나	22
복합 구조 fallback: 노드가 여럿일 때 어느 노드부터 포기하나	24
structure projection: fallback 이 넘어갈 때 내용을 옮기는 규칙	25
alternatives 와 fallback 순서의 차이	26
꾸밈구조(decoration): 등급 하나에서 축 여럿으로	26
포맷 실측과 다시 볼 조건	29
자기정합 검사	29
일정과 작업 상태	30
지금 한 줄로 말하면	30
로드맵 5단계 진행 상황	30
Jira 구조	30
할 일 SSoT(TODO.md)의 절 구조	31
확정된 결정과 폐기된 결정	31
가장 급한 병목 셋 (2026-08-22 기준)	32
5단계 로드맵 지도의 T-번호 작업 상태	32
시간순으로 본 일정	32
위험과 감수하는 것	33
구현물 로직: PoC 파이프라인과 코드 구조	34
왜 PoC 를 만들었는가	34
전체 흐름 (여덟 단계)	34
실서비스 코드와 PoC 코드의 경계	35

왜 검색 경로를 두 갈래로 나란히 두었는가	35
결과물 품질을 재는 방법	36
결함을 생성, 검색, 대치 중 누구 탓으로 돌리는가	37
지금까지 나온 기준선 수치 (2026-08-15 갱신, 입력 50건, 컴포넌트 65개, 상위 10건 기준)	37
정책 결정 중 눈여겨볼 것	37
comp-sign v4.3.8 이 실제로 하는 일 넷	38
태그 군 셋이 랭킹을 정한다	38
검색 fallback 군 넷: 후보가 어디서 오나	39
대치 입력 용어 세 층	41
fallback 이후 대치 계층 다섯	41
평가 pool 을 고정하는 규칙 다섯	42
코드가 어디 있는가	43
Eregion 화면	43
명령어	44
실사용 데이터 E2E 실측 (2026-08-12, gpt-5.4-mini)	44
아직 남은 것	44
부록: 카테고리화 필드 전량 명세	45
이 부록이 무엇인가	45
A. 대분류 21종	45
B. 중분류 45종: 언제 고르나	46
C. 45종의 성립 조건과 실사용 빈도	49
D. <code>params</code> 전량 27종	51
E. <code>structure</code> 전량	53
F. 알려진 격차 다섯	56
G. 45종의 fallback 순서 전량	57
H. fallback 엮기 131개 전량과 적용 규칙	60
I. 열린 결정 전량 (2026-08-28 기준)	65
J. 자기정합 검사 14가지 전량	68
K. 꾸밈구조 형 정의와 해석 예시	69
부록: 카테고리 45종 데이터 구조 예시	72
이 부록을 읽는 법	72
1. 나열과 비교	72
2. 순서와 흐름	75
3. 계층과 범위	77
4. 수치	79
5. 텍스트와 사람	83
6. 페이지 역할	85
7. 병치와 포함: 이중중첩 컴포넌트	86

개요: 이 문서가 무엇이고 Visual Summary 가 무엇인가

이 문서의 목적과 한계

이 문서는 미리캔버스 내용시각화(사용자가 쓴 글을 슬라이드 디자인으로 바꿔 주는 기능)의 하위 트랙인 **Visual Summary(내용시각화 Summary 고도화)** 를 한 번에 통독하기 위해 만든 조합 문서다. 개념, 일정, 작업 상태, 구현물의 로직을 한 흐름으로 붙였다.

이 문서는 **SSoT(단일 진실 공급원)**가 아니다. 원문은 `miridih/mordor-wiki` 저장소의 `docs/projects/mordor/visual-summary/` 폴더이고, 이 문서는 그 폴더의 문서들과 `miridih/aippt-prisonbreak` 저장소의 실제 구현 코드를 읽어 조합한 파생물이다. 값이 어긋나면 원문이 맞다. 원문 폴더 안의 역할 분담은 다음과 같다.

원문 위치	그 문서가 갖는 권한
<code>visual-summary/README.md</code>	프로젝트 허브. 로드맵, 용어 고정, semantic tag 분포
<code>visual-summary/TODO.md</code>	할 일과 정할 것의 SSoT
<code>visual-summary/spec/schema-v0.5.md</code>	규약(스키마) SSoT. 2026-08-02 확정
<code>visual-summary/spec/capability-table.md</code>	카테고리 41종의 성립 조건과 fallback 순서
<code>visual-summary/design/</code>	왜 그렇게 설계했는가
<code>visual-summary/plan/poc-guide.md</code>	처음 보는 사람의 진입점
<code>visual-summary/DASHBOARD.md</code>	진행 현황 요약. 파생물이고 SSoT 아님

오늘은 **정원석(MORDOR)** 이고, 원문 허브의 최종 갱신은 2026-08-17 이다. Confluence 의 원래 허브 페이지(2704247075)는 2026-08-01 부터 갱신을 멈추고 보존만 한다. 옮긴 이유는 둘이다. Confluence 는 MCP 로 갱신할 때 결과가 불확실하고, AI 에게 SSoT 를 컨텍스트로 넣기에도 저장소 쪽이 낫다는 판단이다 (2026-07-31 [#mordor](#)스쿼드 [공지](#)).

배경: 왜 Summary 를 고도화하는가

먼저 오늘 무엇이 돌고 있는지부터 본다. 사용자가 발표 주제나 본문 텍스트를 넣으면, LLM 이 그것을 **Markdown Summary**(마크다운 형식의 중간 요약문)로 바꾼다. 그 다음 시스템은 이 마크다운을 임베딩(문장을 숫자 벡터로 바꾼 표현)으로 만들어, 이미 만들어져 있는 디자인 컴포넌트들의 임베딩과 유사도를 비교해 비슷한 것을 찾는다. 찾은 컴포넌트의 텍스트 자리에 사용자 내용을 넣으면 슬라이드가 나온다.

이 방식의 문제는 하나다. **마크다운은 "무엇을 말하는가"만 담고 "어떤 구조로 보여야 하는가"를 담지 못한다.** 사용자가 "매출이 3년간 늘어난 추이"라고 썼을 때 그것이 시간 순서를 가진 타임라인인지, 세 값을 비교하는 막대 차트인지, 단순 나열인지가 마크다운 텍스트에는 구조로 적히지 않는다. 임베딩 유사도는 단어가 비슷한 컴포넌트를 찾을 뿐이라, 구조가 맞는 컴포넌트를 골라 주지 못한다.

Visual Summary 는 이 잃어버린 구조를 **명시적으로 적는 중간 표현**이다. 사용자 입력을 "컴포넌트 몇 개이고, 각 컴포넌트는 어떤 의미 유형(리스트, 단계, 비교표 등)이며, 그 안에 항목이 몇 개이고 항목끼리 어떤 관계인가"로 구조화해서 적는다. 형식은 YAML 로 확정했다 (2026-08-01 초안, 2026-08-02 v0.5 확정).

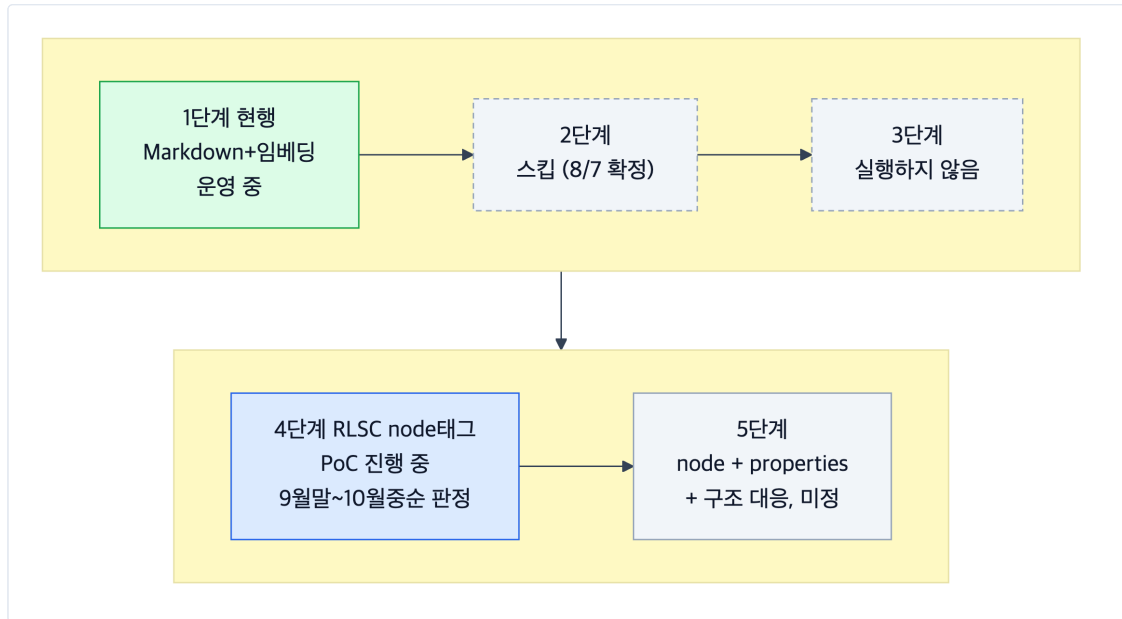
한눈에 보기

질문	짧은 답
무엇을 바꾸나	Summary 형식(Markdown → YAML), 검색 방식(임베딩 → semantic tag + 구조), 태그를 다는 범위(안 달다가 컴포넌트 단위, 최종적으로 node 단위)
지금 어느 단계인가	1단계(현행)가 운영 중이고, 2단계와 3단계는 건너뛰기로 정해졌다. 실제 목표는 4단계다
4단계 목표 시점	2026년 9월 말 dev/stg 반영, 10월 중순 production 판정 (2026-08-04 확정)
규약은 확정됐나	확정됐다. v0.5, 2026-08-02. 원래 마감 8/7 보다 닷새 앞섰다
실물이 도는가	Eregion 화면(/visual-summary)에서 질의 갈래 전체가 돈다 (2026-08-12). 실서비스 코드는 아니고 PoC 다
가장 큰 미해결	RAT pod 소관 I.I.S. 모듈이 2026년 6월 중순부터 정지 상태라 node 단위 태그를 PoC 가 대행 생산하고 있다
데이터 쪽 가장 큰 제약	검색 대상 컴포넌트의 semantic tag 가 List 한 종류에 81.1% 쏠려 있다. 사용자가 원하는 구조는 훨씬 다양하다

5단계 로드맵 요약

단계마다 바뀌는 것은 네 가지다. Summary 형식, 검색 방식, 태그를 다는 범위, 그리고 **그 태그가 무엇을 가리키는가** 이다.

단계	Summary 형식	검색 방식	태그 범위	상태
1단계 (현행)	Markdown	임베딩 유사도	태그 없음	운영 중
2단계	Markdown	semantic tag	컴포넌트 하나에 태그	사실상 스킵 (2026-08-07 확정)
3단계	YAML	semantic tag + 구조 질의	컴포넌트 하나에 태그	실행하지 않음, Jira 상 Cancelled (2026-08-06)
4단계	YAML	semantic tag + RLSC 내부 node 태그	컴포넌트 안 node 하나하나	진행 중. 9월 말 dev/stg 목표
5단계	YAML	4단계와 같음	node 하나하나 + properties (카테고리 고유 속성) + 구조 대응	미정



5단계 추출기는 두 벌을 함께 낸다. 한 번의 호출로 둘이 나온다(2026-08-23 확정, `plan/node-tag-and-property-poc-blueprint.md` 1.0.1절).

산출물	무엇인가	어휘는 어디서 오나	없으면 무엇을 모르나
properties (카테고리 고유 속성)	피라미드의 꼭대기가 어느 쪽인가, 순서가 어느 방향인가처럼 그 카테고리에서만 뜻이 있는 표현 값	<code>capability-table.md</code> 2절 "선택 속성" 열 (<code>reads.params</code>). 질의 쪽 <code>params</code> 와 같은 어휘	이 컴포넌트를 어떻게 그리는가
구조 대응 (structure map)	경계 안의 어느 노드가 <code>label</code> 인지 <code>items</code> 인지 <code>value</code> 인지 <code>period</code> 인지의 대응표	<code>capability.ts</code> 의 <code>reads.content</code> 에 <code>label</code> 과 <code>items</code> 를 더한 것	사용자 내용의 어느 조각을 어느 노드에 넣는가

둘 다 `iis_semantic_component` 테이블의 칸(`properties`, `structure`)으로 들어가고 RLSC 스키마 자체는 건드리지 않는다.

property (속성)라는 말이 두 뜻으로 쓰인다. 이 문서는 아래로 고정한다 (원문 SSOT 는 허브 `README.md` 2절 표기 규칙).

무엇	질의 쪽 (생성기가 적는다)	색인 쪽 (추출기가 뽑는다)
카테고리 고유 속성	<code>params</code>	<code>properties</code>
구조	<code>structure</code> . 내용 자체	구조 대응(structure map). 노드와 필드의 대응표

`params` 와 `properties` 는 같은 어휘를 쓰지만 같은 것이 아니다. 한쪽은 요구("이렇게 그려 달라")이고 다른 쪽은 실측("이 컴포넌트는 이렇게 그려져 있다")이다. 둘을 맞대는 것이 5단계 T-8 이다. **맨 이름 '속성'이나 'property' 를 단독으로 쓰지 않는다.** 넓은 뜻은 한 자리에만 있다. 5단계 국면 이름 I.I.S. (Semantic Component Property) 의 'Property' 가 위 오른쪽 열 두 칸을 합쳐 부르는 이름이고, 그래서 이 이름은 줄여 쓰지 않는다.

단계를 나누는 진짜 이유는 태그의 모호성이다. 태그를 컴포넌트 단위로만 달면, 그 태그가 컴포넌트의 최상위(root) 자체가 그 유형이어서 붙은 것인지, 하위 어딘가에 그 유형의 노드가 있어서 컴포넌트 전체가 받은 것인지 구분할 수 없다. 실데이터가 이것을 보여 준다. 한 컴포넌트에 태그가 최대 4개까지 붙고(UNICORN-77432), 예를 들어 `design_object_id` 4757227 에는 Comparison Table, Hierarchical Tree, List, Highlighted Message 넷이 함께 붙어 있다. 최상위 하나가 이 넷을 동시에 가질 수는 없으니 하위에 여러 유형의 노드를 품고 있다는 뜻이다. 4단계에서 `node` 단위 태그가 붙으면 검색과 내용 대치를 정확한 범위에 걸 수 있다.

용어 사전

같은 말을 같은 뜻으로 쓰기 위해 원문 허브가 고정한 용어다. 처음 보는 사람이 가장 많이 막히는 것부터 적는다.

용어	쉽게 말하면
RLSC	디자인 하나의 레이아웃 구조와 각 요소의 역할(제목, 설명, 마커 등)을 담은 트리 데이터. 디자인을 AI 가 읽을 수 있게 만든 구조 표현이다
design object / 컴포넌트	검색과 대치의 단위가 되는 디자인 조각. 슬라이드 한 장 안의 의미 있는 묶음 하나
semantic tag	"이 콘텐츠가 어떤 의미 구조인가"를 가리키는 분류값. List, Steps, ComparisonTable 같은 값이다. '중분류', 'category class', '시멘틱 태그'가 모두 같은 말이고 semantic tag 로 통일한다
semantic taxonomy (v7 단어 장)	쓸 수 있는 semantic tag 값의 목록. 단어장 밖의 값은 못 쓴다
Summary	사용자 입력을 검색과 대치에 쓰려고 변환한 산출물
Summary 생성기	사용자 입력을 받아 Summary 를 써 내는 LLM 모듈. 오너는 허재영, 황현진
Visual Summary	Summary 를 의미 구조로 구조화하는 스펙. v0.1 부터 v0.5 까지 왔고 현행은 v0.5
capability (성립 조건)	어떤 semantic tag 로 배치하려면 최소한 무엇이 있어야 하는가의 조건. 예를 들어 비교표는 비교 대상이 둘 이상 있어야 성립한다
fallback (강등)	요청한 카테고리가 성립하지 않거나 검색 결과가 없을 때 다음 카테고리로 내려가는 규칙
복잡도 레벨	Visual Summary 하나가 가질 수 있는 위상의 범위. L1(컴포넌트 하나), L2(최상위 병치까지), L3(컴포넌트 경계를 넘는 중첩까지) 셋이고 누적이다. 산출물 한 건의 모양이 아니라 생성기에 허용된 범위를 가리킨다. '질의'라고 부르지 않는다. 질의로 바뀌는 것은 질의 변환기를 거친 뒤다. 실서비스 생성기는 L2 까지이고 PoC 생성기는 이미 L3 를 낸다
structure	Summary YAML 안의 entities, relations, dataset. 검색과 fallback 과 projection 의 입력이다
projection (투영)	fallback 으로 다른 카테고리로 내려갈 때 원래 구조의 내용을 도착 카테고리가 읽는 자리로 옮겨 담는 규칙
comp-sign	이석님 소유의 npm 패키지이자 모듈. 컴포넌트 구조를 해시로 만들어 검색하고, 텍스트를 사용자 내용으로 바꿔 넣는다. 실서비스 경로다
Search Signature v2	컴포넌트 쪽 검색용 정적 데이터. RLSC 와는 별개 contract 다
I.I.S. (증분 정보 구조화기)	RLSC 를 추출한 뒤 부족한 정보를 AI 모듈로 더 뽑아 원래 RLSC 에 속성을 덧붙이는 모듈 계열. Incremental Information Structuriser
I.I.S. (Semantic Component Boundary)	그 계열 중 4단계판. 컴포넌트 안 어느 노드 묶음이 어느 semantic category 인지 경계를 정한다. 코드 식별자 <code>iisSemanticComponentBoundary</code>
I.I.S. (Semantic Component Property)	같은 계열 5단계판. 경계가 잡힌 묶음에서 둘 을 뽑는다. 카테고리 고유 속성(<code>properties</code>)과 구조 대응(<code>structure map</code>)이다. 코드 식별자 <code>iisSemanticComponentProperty</code> . 이 이름의 'Property' 만 넓은 뜻이고 줄여 쓰지 않는다
params / properties / 구조 대응	같은 것을 총마다 다르게 부른다. 카테고리 고유 속성은 질의 쪽에서 <code>params</code> , 색인 쪽에서 <code>properties</code> 다. 노드가 <code>structure</code> 의 어느 필드인지의 대응표는 구조 대응 이라 부르고 질의 쪽 <code>structure</code> (내용)와 구분한다
꾸밈구조 (decoration)	컴포넌트에 붙는 꾸밈 선호. RLSC 의 role <code>Decoration</code> (장식 요소)과는 다른 것이고 앞에 무엇이 붙는지로 구분한다
Eregion	Visual Summary PoC 화면이 들어 있는 내부 개발용 앱 경로. <code>/visual-summary</code>

용어	쉽게 말하면
RAT pod	I.I.S. 본편을 소관하는 다른 팀 단위. 담당 류민
MORDOR	RLSC 추출과 구조화 파이프라인을 담당하는 팀 및 시스템 이름. 이 저장소가 그 코드베이스다

헛갈리기 쉬운 것 셋.

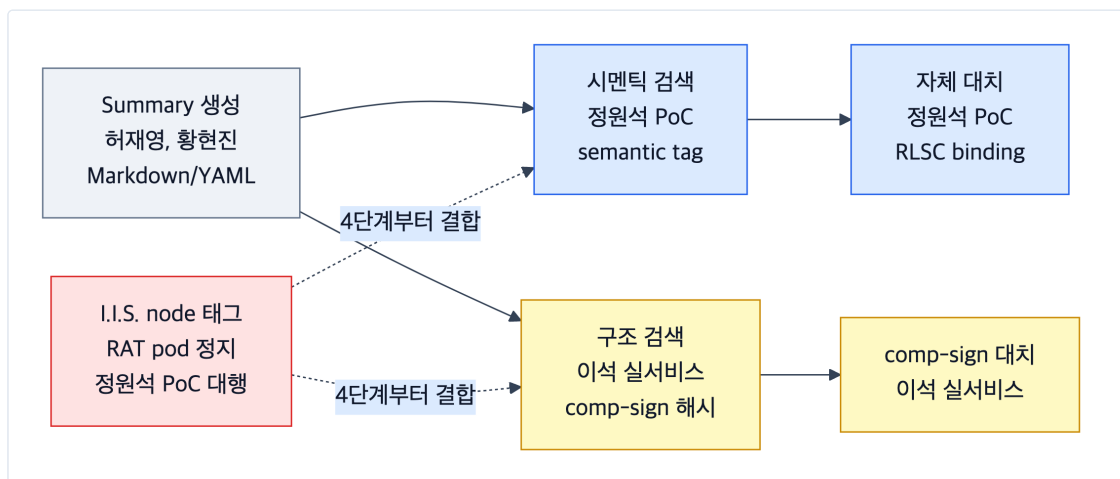
첫째, semantic tag(의미 분류)와 Layout Container type(물리 배치, HStack/VStack/Grid)은 다르다. 단어장에 배치명과 겹치는 라벨이 있어 헛갈리기 쉽다.

둘째, Content Archetype(이 페이지가 문서에서 어떤 역할인가, 표지/목차/본문/마무리)과 semantic tag(이 콘텐츠가 어떤 의미 구조인가)는 서로 다른 것을 분류한다. 지금 실데이터에는 이 둘이 섞여 있다.

셋째, 색인 쪽과 질의 쪽이 다르다. Search Signature v2 는 이미 존재하는 컴포넌트를 색인하는 쪽이고, Summary 의 structure 는 사용자 입력을 질의로 만드는 쪽이다. 분포 수치를 볼 때 이 둘을 섞으면 결론이 뒤집힌다.

사람과 소유 경계

사람	무엇을 갖는가
정원석 (MORDOR PE)	Visual Summary 규약과 스펙, 이 폴더 문서 SSoT, PoC 구현, 질의 변환기 스펙
김수현 (미리캔버스 PM)	v7 단어장 오너, 제품 배포 여부 최종 판단, 생성기 정책 판단
허재영 (프로덕트 그룹 TPM)	골든셋과 판정 실무, Summary 생성기 프롬프트 튜닝
황현진 (DX개발그룹 PE)	추출 모듈, Layout Description 추출과 카탈로그 라우팅 분류기
이석	comp-sign 패키지. 구조 검색, 랭킹, 내용 대치의 실서비스 경로
류민 (RAT pod)	I.I.S. 본편
최운식 (BE)	파이프라인에 태깅 정보 적재, 시그니처 v2 에 태그 검색 적용



이후 절 안내

절	무엇에 답하나
2	규약이 정확히 무엇을 정했나 (스키마 v0.5, 카테고리 성립 조건, fallback)

질	무엇에 답하나
3	일정과 작업 상태, Jira 구조, 아직 열려 있는 결정
4	PoC 가 어디까지 왔고, 코드가 어디에 있으며 어떤 로직으로 도는가

규약: Visual Summary 스키마 v0.5

무엇을 표현하는 형식인가

Visual Summary 는 사용자 입력을 컴포넌트 몇 개, 각 컴포넌트의 의미 유형, 내부 항목 수와 관계로 구조화해 적는 YAML 이다. 최상위는 항상 `components[]` 배열이고, 컴포넌트가 하나여도 길이 1 배열로 적는다. 표면 형식은 YAML 로 확정했다(2026-08-01 초안, 2026-08-02 v0.5 확정). LLM 이 YAML 을 직접 생성하고, 검증 단계에서 JSON 으로 변환해 스키마를 검사한다.

형식 선택 근거. YAML 은 JSON 보다 구두점 토큰이 적어 같은 내용을 더 적은 생성 토큰으로 쓴다. KDL(대안으로 검토한 다른 텍스트 형식)은 토큰이 12.9% 더 적지만 채택하지 않았다. 이유는 셋이다. 사전 조사에서 LLM 생성 신뢰성 벤치마크가 YAML 은 풍부하고 KDL 은 0건이었고, 2026-07-13 v0.2 때부터 이미 YAML 로 예시가 쌓여 있었으며, 규약 확정 마감(2026-08-07)까지 KDL 전환 비용을 감당할 시간이 없었다. 이 결정은 사후 검증 대상으로 남겨 뒀고, 실제로 KDL 이 뚜렷하게 앞서면 재검토한다.

설계 원칙 셋

규약 1절이 정한 원칙이다. **생성 단계에서 최적 semantic category(의미 분류값)를 정한다.** 생성기가 입력을 읽고 어느 카테고리로 그릴지 골라 `type` 에 적는다. 검색은 그 카테고리를 `capability`(성립 조건)와 대조해 실제로 그릴 수 있는지 확인하고, 못 그리면 `fallback`(강등) 순서를 따라간다.

structure 의 필드 이름을 카테고리에 묶지 않는다. `venn.sets` 가 아니라 `entities` 이고, `profile.avatar` 가 아니라 `entity.avatar` 다. 필드 이름이 중립이면 특정 필드를 못 읽는 카테고리는 그 필드를 그냥 지나칠 뿐 오류가 나지 않는다. 이것이 `fallback` 이 안전하게 도는 이유다.

카테고리마다 다른 것은 params 가 담는다. `structure` 는 내용이고 `params` 는 표현 선택이다. 이 구분이 이 절 전체의 핵심이므로 아래에서 따로 다룬다.

이 세 원칙 위에 예외 없는 규칙 셋이 서 있다. 첫째, 어떤 `structure` 든 리스트로 읽히는 `entities` 를 가진다(`dataset` 만 있으면 거기서 파생한다). 둘째, 선언되지 않은 것은 무시한다(관계나 슬롯을 못 읽는 컴포넌트는 그냥 안 읽는다). 셋째, 중첩은 평탄화된다(`entity.components[]` 를 못 읽는 컴포넌트는 자식의 `entities` 를 부모의 `items` 로 편다). 이 셋이 성립하기 때문에 `fallback` 순서가 언제나 끝나고, 컴포넌트가 무엇을 못 그리든 결과가 비지 않는다.

structure, params, decoration: 셋의 역할 구분

같은 스키마 안에 "내용을 담는 곳"이 셋으로 나뉘어 있어서 헷갈리기 쉽다. 표로 정리한다.

구분	무엇을 담나	카테고리마다 다른가	fallback 시
structure	내용. <code>entities</code> , <code>relations</code> , <code>dataset</code>	아니다. 필드 이름이 모든 카테고리에서 같다	그대로 옮긴다(P1~P8 규칙)
params	표현 선택. <code>orientation</code> , <code>apex</code> , <code>icon</code> , <code>header</code> 같은 것	그렇다. 카테고리마다 허용 키가 다르다(<code>capability-table.md</code> 2절)	버린다(P2). 도착 카테고리가 선언하지 않은 표현 선택은 뜻이 없기 때문이다
decoration	꾸밈 선호. <code>ornament</code> , <code>background</code> 두 축	아니다. 카테고리랑 무관하다	컴포넌트 → page → 카테고리 기본값 → 시스템 전역 기본값 → <code>indifferent</code> 순으로 해석한다

params 를 열린 객체로 둔 이유. 카테고리마다 표현 선택 속성이 다르다. 41종의 속성을 전부 스키마 조건문으로 쓰면 스키마가 어휘에 묶여 카테고리가 늘 때마다 스키마 버전이 올라간다. 그래서 스키마는 "객체다"까지만 검사하고, 실제 허용 키와 기본값은 capability 테이블이 카테고리별로 정한다.

decoration 이 params 도 structure 도 아닌 이유. params 는 카테고리별 표현 선택인데 꾸밈구조는 카테고리과 무관하고, structure 는 내용인데 꾸밈구조는 내용이 아니다. 그래서 structure 의 형제로 별도 필드를 둔다.

핵심과 장식 사이에 한 층이 더 있다(열린 자리). 간트차트 가장자리에 붙은 짧은 주석처럼 내용은 담고 있지만 그 카테고리의 성립 조건에는 관여하지 않는 조각이 실사용에 있다. 장식이 아니므로 지우면 내용이 줄고, 핵심이 아니므로 없어도 그 카테고리는 성립한다. 색인 쪽은 이 층을 이미 갖고 있다. 신영빈이 Steps 대상으로 구현하고 류민이 규칙을 설계한 **RLSC 노드 분류 CORE / DESC / DECO / OUT** 이 그것이고(2026-07-31), DESC 가 정확히 "core 는 아니지만 설명 역할을 하는 노드, 있으면 채우는 슬롯"이다 (design/content-binding-contract.md 1.1절).

층	색인 쪽(RLSC 노드 분류)	질의 쪽(스키마 v0.5)
핵심	CORE . 그 카테고리를 성립시키는 노드	structure.entities 와 성립 조건이 요구하는 필드
위성	DESC . 있으면 채우고 없으면 비우는 설명 노드	전용 자리가 없다. entity.items 뒤에 섞이거나 별도 컴포넌트로 쪼개진다
꾸밈	DECO . 비우거나 제거 가능	decoration 두 축(ornament , background)

즉 3단 구분은 색인 쪽에만 있고 **질의 쪽 스키마에는 대응이 없다**. 지금 v0.5 에서 위성 조각은 entity.items 에 딸려 들어가거나 Enumeration.List 컴포넌트 하나로 떨어져 나가는데, 앞은 "핵심 항목의 하위 설명"과 구분되지 않고 뒤는 원래 컴포넌트와의 종속 관계를 잃는다. 이 자리를 여는 것은 요소 대치가 동작한 뒤에 값이 생기고(지금은 DESC 슬롯을 채울 주체가 없다), 축을 늘리는 결정은 Element.Separator 를 세 번째 꾸밈 축으로 열지와 같은 성격의 미결이다(2절 마지막 "아직 정하지 않은 것"). **아직 TODO** 번호가 붙어 있지 않은, 이 문서가 새로 지목하는 구멍이다.

필드 전체 구성

규약이 정의하는 필드를 계층별로 정리한다. 원문은 TypeScript 타입과 JSON Schema 두 벌로 적혀 있고 둘은 같은 것을 정의한다 (SSoT 는 JSON Schema).

최상위 (VisualSummary)

필드	필수	무엇인가
version	예	현행 "0.5.3" . 형식은 MAJOR.MINOR.PATCH 이고 검증은 MAJOR·MINOR 만 대조한다
title	예	빈 문자열 불허
components[]	예	1개 이상
relations[]	아니오	page 수준 관계. 자리만 예약, 생성기는 채우지 않는다
decoration	아니오	page 수준 꾸밈 선호. 자리만 예약, 생성기는 채우지 않는다

규약을 고치면 버전을 올린다(2026-08-29 신설). 종전에는 v0.5 를 자리에서 네 번 고치면서 버전을 그대로 뒀고, 그래서 "지금 규약이 무엇인가"를 버전으로 가리킬 수 없었다. 형식은 MAJOR.MINOR.PATCH 이고 각 자리 0~99 다.

자리	언제 올리나	예
MAJOR	기존 산출물이 실제로 무효가 된다	v0.4 → v0.5 의 "최상위가 항상 components[] "
MINOR	기존 산출물은 유효하고 새로 쓸 수 있는 것이 는다	entities + columns 대안 경로 추가

자리	언제 올리나	예
PATCH	유효성이 안 바뀐다 . 문구 정정, 실제로 지켜지던 것의 명문화	type 필수화(0.5.1), 대분류 제거(0.5.2), enum 45종 반영(0.5.3)

판정은 원리가 아니라 실측으로 한다. type 필수화는 원리적으로 required 를 좁히는 변경이지만 무효가 되는 산출물이 실제로 0 건이라 PATCH 다(골든셋 470/470 이 이미 type 을 갖고 있었다). 실측 없이 좁히는 변경은 MINOR 이상으로 본다. 검증기는 MAJOR 과 MINOR 만 대조하고 PATCH 는 보지 않는다. PATCH 가 유효성을 안 바꾼다고 정의해 놓고 patch 차이로 옛 산출물을 떨어뜨리면 자기모순이다. 파일 이름도 MAJOR.MINOR 만 따라간다. 그래서 이 개정에도 schema-v0.5.md 와 visual-summary-0.5.schema.json 은 그대로다. 규칙의 SSoT 는 규약 0.1절이다.

version 은 LLM 이 쓰지 않는다. 프롬프트가 "version 은 쓰지 않는다"고 지시하고 후처리 코드(generator/postprocess.ts)가 상수를 붙인다. 2026-08-20 100회 실행에서 모델도 100/100 정확히 적었으므로 품질 개선이 아니라 책임 분리다. 모델이 판단할 것이 아닌 값은 모델에게 맡기지 않는다.

title 이 필수인 이유는 소비처가 있어서다. 대체가 페이지 제목 슬롯(headSlot , 없으면 첫 headline 유닛의 제목 슬롯)에 이 값을 넣는다(binding.ts 의 pageTitle).

컴포넌트 (Component)

필드	필수	무엇인가
id	아니오	<code>^[a-zA-Z0-9_-]+\$</code> . 생성기는 채우지 않고 파싱 뒤 후처리가 부여한다
type	예 (0.5.1)	생성기가 고른 카테고리. 결합값(대분류, 중분류) 45종만 받는다(0.5.2 , 0.5.3). 적을 의무는 있고, 검색이 그대로 따를 의무는 없다
alternatives[]	아니오	못 쓸 때 대신 쓸 카테고리를 선호 순서로
params	아니오	표현 선택. 허용 키는 capability 테이블이 카테고리별로 정한다
structure	예	내용. 아래 Structure 참고
decoration	아니오	이 컴포넌트에만 적용되는 꾸밈 선호

type 은 2026-08-29 에 필수가 됐다(정원석). 종전에는 optional 이었고 근거는 "필수로 걸면 '카테고리를 못 고르겠다'가 곧 생성 실패가 된다" 였는데, 뒤집혔다.

근거는 이 단계의 존재 이유다. 설계 원칙 첫째가 "생성 단계에서 최적 semantic category 를 정한다"이다. 어떤 시각 표현형으로 그 릴지를 먼저 정해야 거기서부터 표현 방식이 따라 나온다. type 을 비우고 내보내는 것은 그 판단을 검색 쪽으로 미루는 것이고, **시각화 책임을 회피하는 것이다**. 원칙 1과 optional 은 처음부터 어긋나 있었고 이 개정이 그것을 맞춘다.

'적을 의무'만 바뀌고 '따를 의무'는 그대로다. 필수가 된다고 type 이 지시가 되지는 않는다. 검색은 여전히 성립 조건과 색인 공급을 보고 fallback 으로 벗어난다. 못 고르겠는 경우의 답은 비우는 것이 아니라 **종단 셋 중 하나를 고르는 것이다**. 셋 다 개수 조건만 보므로 어떤 내용이든 하나는 성립한다. 고를 수 없는 입력은 없다.

깨지는 산출물은 없다. 골든셋 v1 의 컴포넌트 470개가 전부 이미 type 을 갖고 있고 PoC 프롬프트도 적으라고 지시하고 있었다. 실제로 지켜지던 것을 규약에 올리는 개정이라 버전은 PATCH 로 올렸다(0.5.0 → 0.5.1).

같은 날 두 번째 개정으로 대분류 단독도 막혔다(0.5.2). type 과 alternatives 는 **결합값(대분류, 중분류)만** 받는다. 종전에는 중분류를 못 정하면 Hierarchy 만 적을 수 있었고 검색이 그 안 전체를 후보로 봤는데, 그것도 같은 성격의 회피다. **대분류는 그릴 수 있는 표현형이 아니다**. Hierarchy 로는 아무것도 못 그리고 그릴 수 있는 것은 Hierarchy.HierarchicalTree 나 Hierarchy.Pyramid 다. 못 정하겠으면 **종단 셋 중 하나를 고른다**. 대분류 개념 자체는 남는다. fallback 근접도의 "대분류 일치는

강한 신호"와 규약 6절 셋째의 신호로 **값이 아니라 축으로** 계속 쓴다. 여기서도 깨지는 산출물이 없다(골든셋의 컴포넌트 `type 454` 개가 전부 결합값, 생성기 프롬프트도 결합값 45종만 준다).

`alternatives[]` 를 4단계와 5단계에서 **결지는 정해지지 않았다**. 스키마는 optional 로 열려 있고 PoC 코드(`fallback.ts`)는 정적 fallback 순서보다 먼저 보도록 이미 구현했지만, 정적 순서보다 나은 결과를 낸다는 실측이 없어 생성기 정책으로 켜고 끄게 두었다.

구조 (Structure): 네 필드 각각은 optional 이고, 대신 **entities 와 dataset 중 적어도 하나**라는 조건이 필드 넷에 걸쳐 있다 (JSON Schema 의 `anyOf`). 그래서 아래 표에는 필드별 필수 열을 두지 않는다.

필드	무엇인가	언제 있어야 하나
<code>entities[]</code>	항목 목록. 1개 이상	<code>dataset</code> 이 없으면 필수
<code>relations[]</code>	이 컴포넌트 안 항목 간의 관계	항상 선택
<code>dataset</code>	수치 데이터. <code>Measure</code> 또는 <code>Matrix</code>	<code>entities</code> 가 없으면 필수
<code>columns[]</code>	<code>entities</code> 로 표를 적을 때 헤더. <code>dataset</code> 없이도 쓴다	항상 선택

항목 (Entity): `label` 만 필수이고 나머지는 그 카테고리가 읽을 수 있으면 읽는다.

필드	무엇인가	주로 읽는 카테고리
<code>label</code>	항목 이름. 빈 문자열 불허	공통
<code>items[]</code>	하위 설명 목록	공통
<code>value, unit</code>	수치와 단위	BigNumber, Gauge 등
<code>period</code>	기간 표기	Timeline, GanttChart
<code>emphasis</code>	강조 여부	HighlightedMessage
<code>weight</code>	상대 비중	PieChart, DonutChart 계열 fallback
<code>speaker</code>	화자	DialogueChat
<code>author, source</code>	인용 출처	QuoteTestimonial
<code>avatar, organization</code>	인물 정보	ProfileCard
<code>values[]</code>	<code>columns</code> 와 위치로 대응하는 나머지 열 값	병합 헤더가 있는 표(2.5.1절 방식)
<code>children[]</code>	하위 항목(같은 컴포넌트 안 계층)	Hierarchy, NestedScope 등
<code>components[]</code>	중첩된 컴포넌트(컴포넌트 경계를 넘는 계층)	생성기는 아직 채우지 않는다

이름이 중립적으로 지어진 이유가 여기서 드러난다. `avatar` 와 `organization` 은 ProfileCard 가, `speaker` 는 DialogueChat 이, `author` 와 `source` 는 QuoteTestimonial 이 읽지만, 필드 이름 자체에는 그 카테고리 이름이 들어가지 않는다. 그래야 다른 카테고리로 fallback 됐을 때도 그 필드가 오류 없이 무시된다.

structure 도 검사한다. 다만 검사하는 주체가 다르다. `params` 만 검사하고 `structure` 는 안 하는 것이 아니라, **스키마가 할 수 있는 일은 스키마가 하고 못 하는 일만 capability 가 대신한다**. 갈리는 이유는 두 필드의 모양이 다르기 때문이다.

	structure	params
스키마 모양	필드 이름이 고정 목록이고 zod 가 <code>.strict()</code> 다	열린 객체(Record<string, unknown>). 카테고리마다 키가 달라 스키마에 못 박을 수 없다
어휘 밖 이름을 적으면	스키마가 error 로 막는다. <code>entity.avatar</code> 같은 오타나 지어낸 필드가 여기서 죽는다	스키마가 못 잡는다. 무엇을 적어도 통과한다
그래서 capability 가 하는 일	성립 조건 판정. 개수, <code>dataset</code> 모양, 필수 내용(<code>Gauge</code> 의 <code>measure.value</code> , <code>HierarchicalTree</code> 의 <code>children</code> 등)	허용 키 판정. 그 카테고리가 선언하지 않은 키를 <code>warning</code> 으로 센다

즉 `structure` 는 두 겹으로 검사받는다. 필드 이름, 타입, 필수 조건은 스키마가 전량 보고(`entities` 1개 이상, `label` 빈 문자열 불허, `entities|dataset` 중 하나 등), 카테고리별 최소 요구는 `capabilityCheck.ts` 가 본다. `params` 가 검사를 한 겹밖에 못 받는 쪽이고, `capability` 의 키 검사는 스키마가 못 하는 일의 대역이다.

진짜로 검사하지 않는 것은 하나뿐이다. "그 카테고리가 읽지 않는 내용 필드가 실려 있는 것". `List` 에 `avatar` 를 적는 경우다. 이것은 설계 원칙 셋 중 "선언되지 않은 것은 무시한다"에 따라 오류가 아니고, fallback 때 P3 으로 `items` 맨 뒤에 붙어 보존된다. `params` 의 대응 상황(그 카테고리가 안 읽는 키)이 `warning` 인 것과 비대칭인데, 갈리는 근거는 사라지느냐. 안 읽는 `params` 는 P2 로 버려지므로 잘못 적으면 의도가 조용히 소멸한다. 안 읽는 내용 필드는 P3 으로 보존되므로 소멸하지 않는다. 검사는 조용히 사라지는 쪽에 건다.

그래도 남는 물음이 있다. `List` 에 `avatar` 를 적은 것은 내용이 안 사라질 뿐 생성기가 카테고리를 잘못 골랐다는 신호이기는 하다 (그 내용이면 `ProfileCard` 있어야 한다). 실패로 올릴 일은 아니지만 `info` 로 세어 생성 품질 지표에 넣을 수는 있다. `capabilityCheck.ts` 에 이미 같은 성격의 `info` 등급(자손을 합쳐야 성립하는 경우)이 있다. 아직 정해지지 않았다(B-33).

공용 zod 스키마의 한계는 `type` 쪽에도 있다. `type: KPIKeyMetric.Gauge` 인데 `measure` 가 없어도 스키마는 통과시킨다. 그 구멍을 메우는 것이 아래 표의 등급 판정이다.

판정	등급	무엇을 뜻하나
자기 <code>structure</code> 로 성립한다	통과	정상
자손 컴포넌트(<code>entity.components</code>)를 합치면 성립한다	<code>info</code>	데이터는 있고 적절한 자리가 다르다
fallback 그래프로 성립하는 도착지가 있다	<code>warning</code>	설계상 정상. <code>strictCapability</code> 를 켤 때만 실패로 올린다
마지막 fallback 셋까지 성립하지 않는다	<code>error</code>	어떤 카테고리로도 못 그린다. 항상 검증 실패
그 카테고리가 선언하지 않은 <code>params</code> 키	<code>warning</code>	허용 키는 <code>capability.json</code> 의 <code>reads.params</code> 가 정한다

등급을 셋으로 나눈 이유는 "Gauge 인데 `dataset` 이 없다"가 그 자체로 결함이 아니기 때문이다. `type` 은 권고이고, 성립하지 않으면 검색이 fallback 으로 내리는 것이 정상 동작이다. 문서화도 되어 있다. 규약 5절과 6절과 `capability-table.md` 2절과 3절이 조건을, 코드 쪽은 `capabilityCheck.ts` 머리말이 이 등급 나눅의 근거를 담는다. 45종 전량과 코드의 대조는 `capability.test.ts` 가 한다.

관계 (Relation)

필드	무엇인가
<code>type</code>	<code>intersects</code> (교집합) 또는 <code>compares</code> (비교). 이 둘만 허용

필드	무엇인가
of[]	관계로 묶이는 entity 의 label . 파싱 뒤 id 로 바뀐다
desc	그 관계에만 속하는 설명. 예를 들어 두 원의 교집합 영역 텍스트

관계 어휘를 새로 만드는 기준은 하나다. 구조로는 적을 수 없는 제3의 내용을 가진 관계만 어휘로 만든다. 배열 순서나 entity 내부 필드, children 으로 이미 적히는 것은 별도 관계로 만들지 않는다. 이 기준으로 "전후 변화"(화살표에 없는 텍스트가 실사용에서 관측되는지 미확인, 보류)와 "문제와 해결/원인/결합"(entity 하나로 이미 표현 가능, 기각)이 걸러졌다.

데이터셋 (Dataset): 두 모양(shape) 중 하나다.

모양	필드	무엇인가
measure (단일 수치)	value , unit , min , max , target , delta , thresholds[]	BigNumber, Gauge 처럼 값 하나가 핵심인 카테고리
matrix (행렬)	columns[] , rows[][]	표와 차트 계열 전부. 표현 방식은 데이터가 아니라 type 이 정한다

matrix 의 첫 열은 항상 범주 축이다. column.role 이 category (범주), value (수치), x / y / size (산점도와 버블), start / end (간트), date (달력)로 각 열의 성격을 밝히고, 생략하면 첫 열은 category , 나머지는 value 로 간주한다.

병합과 계층 헤더가 있는 표는 matrix 대신 entities 를 쓴다(2026-08-06 추가). 병합 셀이 있는 실사용 표가 matrix 성립 조건에 걸려 통째로 탈락하는 사례가 나와서, ListingTable 과 ComparisonTable 두 카테고리에만 대안 구조를 열었다. columns (헤더 정의)와 entities (행)를 함께 쓰고, entity.label 이 범주축(첫 열), entity.values[] 가 나머지 열이다. 상위 범주가 하위 행을 거느리는 부모-자식 구조는 entity.children[] 로 적는다.

이 경로가 담는 병합은 "첫 열의 세로 병합" 하나다. columns[] 는 평평한 한 줄이고 Column 에는 병합을 적을 자리(colspan 같은 것)가 없다. 그래서 헤더가 2단인 표(가로로 병합된 상위 헤더 아래 하위 헤더가 붙는 모양)는 질의 쪽에서 표현할 수 없다. 지금 41종 어느 차트와 표 카테고리도 그런 구조를 읽지 않기 때문에 자리를 열지 않았다. 세로 병합 중에서도 담기는 것은 범주축 하나가 하위 행을 거느리는 계층형뿐이고, 축이 둘로 갈라지는 독립적 rowspan(주축과 무관하게 특정 열 하나만 여러 행에 걸쳐 묶이는 것, 예를 들어 강의 4행 중 "주말반" 열만 2행씩 묶이는 표)은 트리 하나로 담기지 않아 범위 밖이다(TODO B-29).

아래 예시가 바로 그 계층형 세로 병합이다. 첫 열의 PROJECT 01 셀 하나가 아래 두 행(성장률 , 매출)에 걸쳐 있는 표이고, 그 병합을 entity.children[] 로 적는다. columns 네 개는 병합 없는 평평한 헤더 한 줄이다.

```
version: "0.5.3"
title: 분기별 프로젝트 성과
components:
  - type: AttributeComparison.ComparisonTable
    structure:
      columns:
        - { label: 항목 }
        - { label: 1Q }
        - { label: 2Q }
        - { label: 3Q }
      entities:
        - label: PROJECT 01
          children:
            - { label: 성장률, values: ["35%", "67%", "55%"] }
            - { label: 매출, values: [14000, 25000, 18000] }
```

이 경로로 채워진 표는 `matrix` 를 요구하는 카테고리(Quadrant, GanttChart, Calendar 등)의 fallback 도착지가 되지 않는다. 역지로 `matrix.rows` 로 접근하면 병합 의미가 조용히 사라지기 때문이다.

꾸밈구조 (Decoration)

필드	대응하는 RLSC role	값
<code>ornament</code>	<code>Element.Decoration</code> , <code>LayoutContainer.Decoration</code>	<code>prefer</code> (선택), <code>indifferent</code> (무관, 기본값), <code>avoid</code> (회피)
<code>background</code>	<code>Element.Background</code> , <code>LayoutContainer.Background</code>	위와 동일

`require` (필수)는 없다. 꾸밈을 필수로 걸면 후보가 0이 되어 전부 List 로 강등되기 때문이다. 해석은 축마다 따로 하고 구체적인 쪽을 우선한다. 컴포넌트 → page → 카테고리 기본값 → 시스템 전역 기본값 → `indifferent` 순이다.

값의 모양은 셋으로 갈라 판정한다(2026-08-29 확정). 셋을 한 덩어리로 묶으면 "객체는 안 된다"와 "점수는 안 된다"가 뒤섞인다.

모양	판정
<code>enum</code> (<code>prefer</code> · <code>indifferent</code> · <code>avoid</code>)	4단계와 5단계가 쓰는 것이 이것이다. 지금 규약에 있는 형태다
객체 (축 아래에 구체적인 값을 담는 것)	나중 단계의 자리다. 기각이 아니라 4·5단계 범위가 아닐 뿐이고, 그 단계에서 무엇을 담을지는 정하지 않았다
수치 점수 (<code>ornament: 0.8</code> 같은 가중치)	앞으로도 두지 않는다. 값에 점수를 매기는 것이 의미가 없다고 판단했다. 채점 코드를 건드리지 않기로 한 결정과 어긋나고 <code>entity.weight</code> (number)가 다른 뜻으로 이미 있다는 종전 근거도 그대로다

그래서 "`prefer` 가 몇 점인가"는 물음 자체가 성립하지 않는다. 남는 것은 **점수가 아닌 형태로** 검색이 이 선호를 순위에 어떻게 반영하는가, 그리고 fallback 보다 먼저 보는가다. 값 말고 늘어날 수 있는 것은 **축의 개수**다. `Element.Separator` 를 세 번째 축으로 열지가 미결이고(2절 마지막), 새 축을 열 때는 기본값을 `indifferent` 로 두면 기존 동작이 바뀌지 않는다.

```

version: "0.5.3"
title: 분기 실적
decoration:                                # page 수준. 아래 컴포넌트의 기본값이 된다
  ornament: prefer
  background: prefer
components:
  - type: Enumeration.List                  # 물려받는다. 둘 다 prefer
    structure:
      entities: [ { label: 신규 계약, items: [12건] } ]
  - type: ProcessFlow.Steps
    decoration:
      background: indifferent              # 이 축만 되돌린다. ornament 는 prefer 그대로다
    structure:
      entities:
        - { label: 계약 검토 }
        - { label: 체결 }

```

1단계에서 생성기는 `decoration` 과 `entity.components[]`, page 수준 `relations` 를 채우지 않는다. 스키마에 자리만 열려 있고 소비할 주체(요소 대치)가 아직 동작하지 않기 때문이다.

page 수준에 있는 것은 `relations` 와 `decoration` 둘뿐이고, "페이지를 어떻게 배치하는가"는 규약 밖이다. page 수준 `relations` 는 컴포넌트 블록끼리의 의미 관계를 적을 자리이지 배치 지시가 아니고(그래서 컴포넌트 블록에 optional `id` 를 함께 열어 두었다, TODO B-15), `decoration` 은 꾸밈 선택이다.

page layout 규약은 5단계 범위에도 없다(2026-08-29 확정). 5단계가 뽑는 것은 노드 태그와 `properties` 와 구조 대응이지 배치가 아니다. 다만 계획이 없는 것이 아니라 미룬 것이다. 배치를 다루자는 결정은 5단계 이후로 잡혀 있고, 그때까지 질의 쪽 스키마에 좌표나 그리드를 넣지 않는다. 지금 넣으면 배치를 실제로 정하는 쪽(검색이 고른 컴포넌트와 그것이 속한 템플릿)과 두 벌로 갈린다.

그 전에 풀려야 할 것 둘이 먼저다. 복합 구조 fallback 에서 N3 과 N4 를 반복하면 최상위 컴포넌트 수가 늘어나는데 그 상한이 없고 (B-32), 여러 조각으로 쪼갠 검색 결과를 한 페이지로 조립하는 오케스트레이션 코드도 없다(I-11). 개수와 조립이 정해지지 않은 상태에서 배치 규약을 먼저 쓸 수 없다.

YAML 예시 셋

관계(`relations`)를 쓰는 예시. 벤 다이어그램처럼 두 항목의 교집합에 별도 텍스트가 붙는 경우다.

```
version: "0.5.3"
title: 장수기업 HR의 핵심 과제
components:
  - type: RelationalComparison.VennDiagram
    structure:
      entities:
        - { label: 전통, items: [유산, 정체성, 신뢰] }
        - { label: 변화, items: [혁신, 적응, 미래] }
      relations:
        - type: intersects
          of: [전통, 변화]
          desc: 두 요소를 조화롭게 관리하는 것이 장수기업 HR의 핵심
```

`of` 에는 entity 의 `label` 을 그대로 적는다(파싱 뒤 id 로 치환). `desc` 는 어느 entity 에도 속하지 않는 내용이라서 관계로 뽑아 적는다.

`params` 와 `alternatives` 를 함께 쓰는 예시.

```
version: "0.5.3"
title: 매슬로 욕구 5단계
components:
  - type: Hierarchy.Pyramid
    params: { apex: last }
    alternatives:
      - Progression.Staircase
      - Enumeration.List
    structure:
      entities:
        - { label: 생리적 욕구 }
        - { label: 안전 욕구 }
        - { label: 사회적 욕구 }
        - { label: 존경 욕구 }
        - { label: 자아실현 }
```

`apex` (어느 쪽이 꼭대기인가)는 안 적어도 된다. 기본값이 `last` 다. `alternatives` 는 생성기가 입력 내용을 보고 미리 좁혀 두는 대안 목록이고, 적으면 검색이 정적 fallback 순서보다 먼저 참고한다.

type 을 문자열 하나로 적는 이유. 대분류가 중분류의 이름공간이다. Pyramid 는 Hierarchy 아래에만 있고 BarChart 는 MagnitudeRanking 아래에만 있어서, Hierarchy.Pyramid 처럼 점으로 이어 적는다. 대분류 21종과 결합(대분류.중분류) 41종을 합쳐 값 62개 목록으로 스키마가 조합을 강제하므로 Hierarchy.BarChart 처럼 어긋난 조합은 검증에서 걸린다. 저장 키(DB, comp-sign)는 밑줄 표기(Enumeration_List)를 쓰고, 검색 모듈이 받는 즉시 점을 밑줄로 바꾼다. 이 변환은 코드 한 곳에만 있다.

Capability(성립 조건) 테이블의 구조

docs/projects/mordor/visual-summary/spec/capability-table.md 가 semantic tag 45종 전량에 대해 세 가지를 정의한다. 성립 조건(이게 없으면 그 종류로 배치가 불가능한 것), 선택 속성과 기본값(안 적어도 되는 것), fallback 순서(그 종류가 색인에 없을 때 물러날 곳)다.

45종이라는 숫자의 경위. v7 semantic taxonomy(단어장)는 49종이다. 여기서 지도 전용 4종(Distribution, Location, Highlight, Flow)을 뺀 45종이 이 표의 범위다. 지도 데이터가 있어야 성립하는 유형이라 structure 로 표현할 대상이 아니고 당분간 Summary 생성 대상에서도 뺀다(2026-08-17 스킴드 확인). 이전에 41종으로 적혔던 것은 페이지 역할 4종(Opening, TableOfContents, SectionDivider, Ending)을 빼고 세던 값이다. 페이지 역할 4종은 2026-08-06(B-24)에 어휘로 편집해 표에 넣었지만, 성립 조건 자체는 C-4(페이지 역할 태그를 검색 풀에서 어떻게 다룰지)가 담혀야 채울 수 있어 아직 미정으로 남아 있다.

성립 조건은 최소로 줄인다(2026-08-02 결정). 필수/선택/무시 세 등급으로 나누는 대신, 성립 조건은 배치 자체가 불가능한 것만 남기고 나머지는 기본값을 가진 선택 속성으로 옮겼다. 기본값은 표현 선택에만 준다. 피라미드에서 어느 쪽이 꼭대기인지(apex)는 표현 선택이라 기본값을 줘도 되지만, 숫자가 얼마인지(value)는 내용이라 기본값을 주면 없는 데이터를 지어내는 셈이 되므로 성립 조건에 남긴다.

대표 카테고리 예시다. 45종 전량은 5절 부록 C 에 있다.

카테고리	개수 조건	내용 조건
Enumeration.List	1 이상	없음
AttributeComparison.Versus	정확히 2	없음
AttributeComparison.ComparisonTable	열 2 이상	matrix 또는 entities + columns
RelationalComparison.SWOT	정확히 4	없음
RelationalComparison.VennDiagram	2 이상 3 이하 (실측)	없음
ProcessFlow.Steps	2 이상	없음
RelationalComparison.Quadrant	1 이상	matrix 필수(항목마다 x, y)

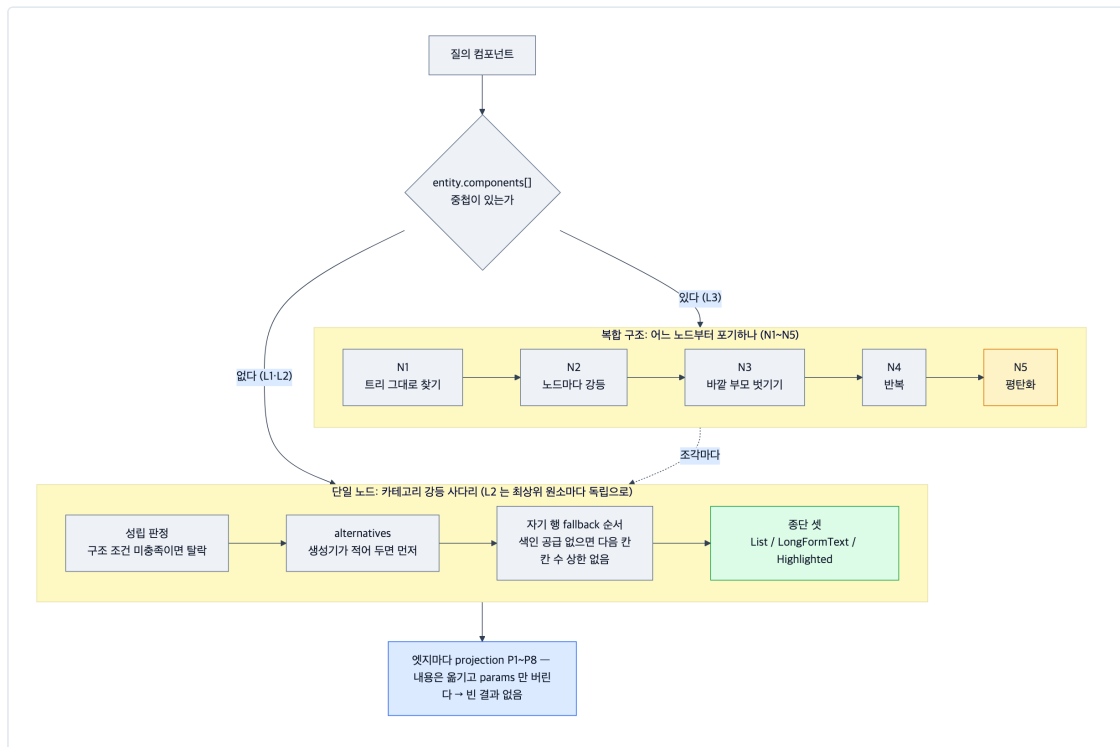
"실측" 표시가 붙은 값은 배치 엔진(jari) 0.12.0 의 LAYOUT_CONTENT_CONTRACTS 에서 그대로 가져온 것이라 근거가 확실하다. 표시가 없는 값은 단어장 정의문에서 유도한 설계 판단이다.

fallback(강등) 사다리

fallback 은 요청한 카테고리가 성립하지 않거나 검색 결과가 없을 때 다음 카테고리로 내려가는 규칙이다. 규약 5절이 정한 다섯 가지다.

1. 카테고리의 성립 조건을 못 채우면 그 카테고리가 후보에서 빠진다. 성립 조건은 개수 범위와 내용 존재만 본다.
2. 후보가 없으면 fallback 순서를 따라 내려간다. **상한을 두지 않는다.** 근접도 순서만 지키면 몇 칸이든 허용한다(2026-08-17 개정. 종전에는 "최대 두 칸"이었다가 폐지됐다).

3. 마지막 fallback 은 셋 중 하나다. `entities` 가 2개 이상이면 `Enumeration.List` , 1개이고 여러 문장이면 `TextBasedCommunication.LongFormTextLayout` , 1개이고 한 문장이면 `TextBasedCommunication.HighlightedMessage` 다. 셋 다 개수 조건만 보므로 언제나 하나는 성립하고, 그래서 빈 결과가 사용자에게 가지 않는다.
4. 선택된 카테고리가 못 읽는 내용은 버리지 않고 옮긴다. 어디로 옮기는지는 다음 절의 projection 규칙 8가지가 정한다.
5. `type` 이 가리킨 카테고리의 성립 조건을 `structure` 가 못 채우면 구조가 우선이다. 그 카테고리는 후보에서 빠지고 fallback 순서로 넘어간다.
6. 질의가 중첩이나 병치로 **노드를 여럿** 가지면, 1번부터 5번과 별개로 어느 노드를 먼저 포기하는지를 정해야 한다. 순서는 다섯 단계이고 다음 절에서 다룬다(2026-08-28 신설).



그림을 읽는 법. 갈림길은 하나다. 중첩이 없으면(L1 과 L2) 아래 줄(강등 사다리)만 돌고, L2 는 최상위 원소마다 그것을 독립으로 돌린다. 중첩이 있으면(L3) 위 줄(N1~N5)이 먼저 돌아 질의를 조각으로 나눈 뒤 **조각마다** 아래 줄이 다시 돈다. 두 줄은 예산이 따로따로(N-R4). 사슬이 막히는 이유는 둘인데(요청 카테고리의 성립 조건 미충족, 색인 공급 없음) 아래 줄의 첫 칸과 셋째 칸에 각각 걸린다. 앞의 것은 대개 생성 결함이고 규약이 그것을 오류 대신 **fallback** 으로 흡수한다. 근거는 바로 아래 절이다. 어느 경로로 가든 끝은 중단 셋이라 빈 결과가 없고, 칸을 하나 옮길 때마다 그 엣지의 projection 이 붙어 내용은 옮겨지고 `params` 만 버려진다. 위 줄의 N1 과 N2 는 이중중첩에서는 지금 정책상 언제나 비어 있어(D-7), 실질적으로 N3 에서 시작한다.

사슬이 막히는 이유 둘과, 한 단계씩 따라간 예시 셋

말로만 읽으면 "카테고리를 내린다"가 무엇을 하는 일인지 잡히지 않는다. 실제로 무엇이 판정되고 무엇이 옮겨지는지 세 경우로 따라간다. 앞의 둘이 사슬이 막히는 이유 둘에 각각 대응하고, 셋째가 노드가 여럿일 때다.

먼저 하나를 분명히 해 둔다. 요청한 `type` 의 성립 조건을 그 컴포넌트의 `structure` 가 못 채우면, 그것은 잘못 생성된 **Visual Summary** 다. 생성기 프롬프트가 "정의의 개수 조건과 dataset 요구를 채울 수 있는지 먼저 확인하고, 못 채우면 정의의 '못 쓸 때 대안' 순서를 따른다"고 지시하기 때문에, 못 채우는 `type` 을 적은 것은 지시 위반이다. 그런데도 규약은 이것을 검증 오류로 두지 않는다. 이유가 셋이다.

왜 오류로 하드 실패시키지 않나

- 1 `type` 은 권고다(규약 6절). 최종 결정은 검색이 하므로, 권고가 틀렸다는 것만으로 문서 전체를 버릴 근거가 못 된다

왜 오류로 하드 실패시키지 않나

- 2 데이터가 다른 자리에 적혀 있을 수 있다. `entity.components[]` 중첩 안에 있으면 자손을 합쳐 성립하는 경우가 있어, 이건 결함이 아니라 "적힌 자리가 다른 것"이다
- 3 빈 결과보다 낫다. 하드 실패시키면 재생성 루프를 돌아야 하고 그것마저 실패하면 사용자에게 아무것도 못 준다. `fallback` 으로 흡수하면 내용은 다 살아서 나간다

대신 생성 품질 신호로 센다. 코드(`capabilityCheck.ts`)가 이 경우를 `warning` 등급으로 세고(`capabilityFallbackCount`), `strictCapability` 를 켜면 검증 실패로 올린다. 어떤 카테고리로나 못 그리는 경우만 항상 `error` 다. 즉 기본 설정에서는 재생성으로 고치지 않고 `fallback` 이 흡수한다. 이것이 얼마나 자주 일어나는지는 아직 실측이 없다.

성립 미충족이 결함이 아닌 경우도 있다. 도착지에서의 미충족이 그것이다. 아래 예시 1의 마지막 칸에서 `HighlightedMessage` 가 "정확히 1"인데 항목이 셋이라 P8 로 접어 받는데, 이건 생성기가 틀린 게 아니라 강등 경로가 원래 그렇게 도는 것이다. `type` 을 아예 안 적는 것은 2026-08-29 부터 스키마 위반이다. 종전에는 `optional` 이라 결함이 아니었다.

예시 1. 생성기가 성립하지 않는 카테고리를 골랐다. `Gauge` 인데 `measure` 가 없다.

```
components:
  - type: KPIKeyMetric.Gauge
    structure:
      entities:
        - { label: 서울, value: 82, unit: "%" }
        - { label: 부산, value: 61, unit: "%" }
        - { label: 대구, value: 47, unit: "%" }
```

생성기는 달성률이니 게이지라고 판단했다. 그런데 `Gauge` 의 성립 조건은 `dataset.shape: measure` 가 필수이고 `measure.value` 도 필수다(부록 C). 이 산출물은 수치를 `entity.value` 로 적었지 `dataset` 을 적지 않았다. 위에 적은 대로 이것은 생성 결함이다. 게다가 값이 셋이라 게이지(값 하나가 핵심인 카테고리) 자체가 애초에 어울리지 않는 입력이었을 가능성이 높다. 구조가 우선이므로 `Gauge` 는 후보에서 빠지고(규칙 5번), 검증은 이 컴포넌트를 `warning` 하나로 세면서 통과시킨다. 여기서부터 자기 행을 따라간다.

칸	도착	판정	무엇이 일어나나
1	<code>KPIKeyMetric.BigNumber</code>	성립	개수 1 이상이고 필수 내용이 <code>entity.value</code> 인데 셋 다 갖고 있다. 여기서 멈춘다

걸리는 규칙은 P1 P2 P3 이다. `entities` 셋은 순서까지 그대로 간다(P1). 버릴 `params` 는 없다. `KPIKeyMetric` 둘은 `params` 키를 아예 선언하지 않는 16종에 속하기 때문에 P2 가 할 일이 없다. 이 질의에 `dataset` 이 있었다면 `min`, `max`, `target`, `thresholds` 는 `BigNumber` 가 안 읽는 값이라 각 `entity` 의 `items` 맨 뒤에 이름: 값 으로 붙었을 것이다(P3). 읽는 것은 "목표 대비 어디쯤인가"라는 의미 하나뿐이고 숫자와 라벨은 하나도 안 없어진다.

같은 질의에서 색인이 알으면 사슬이 더 간다. `BigNumber` 태그를 단 컴포넌트가 색인에 하나도 없으면 다음 칸으로 내려간다.

칸	도착	판정	무엇이 일어나나
2	<code>ProportionComposition.PictogramChart</code>	보류	<code>matrix</code> 가 필수인데 이 질의엔 없다. <code>measure → matrix</code> 를 접는 규칙 P9 가 아직 미확정(B-30)이라 지금은 못 간다
3	<code>MagnitudeRanking.BarChart</code>	보류	같은 이유다
4	<code>TextBasedCommunication.HighlightedMessage</code>	P8 로 성립	개수 조건이 "정확히 1" 인데 셋이라, 초과분 둘을 마지막 항목의 <code>items</code> 로 접어서 받는다(P8). 이 칸의 미충족은 생성 결함이 아니라 강등 경로가 원래 그렇게 도는 것이다

예시 2. 색인 공급 없음. 구조는 멀쩡한데 그 태그를 단 컴포넌트가 없다.

```
components:
- type: AttributeComparison.Versus
  structure:
    entities:
      - { label: 재택근무, items: [출퇴근 시간 0분] }
      - { label: 사무실 근무, items: [즉시 소통] }
    relations:
      - type: compares
        of: [재택근무, 사무실 근무]
        desc: 협업 밀도에서 갈린다
```

Versus 의 성립 조건은 "정확히 2" 하나이고 이 산출물은 정확히 2다. 생성은 흠이 없고 구조도 통과한다. 예시 1 과 다른 점이 여기다. 이쪽은 생성기를 탈탈 것이 없다. 그런데 Versus 태그를 단 컴포넌트가 색인에 없으면(질의 쪽 비중은 3.11% 인데 색인 쪽 공급은 별개다) 후보가 0이 되고, 이유가 구조가 아니라 공급이어도 사슬은 똑같이 내려간다. Versus 행은 ComparisonTable → List 다.

칸	도착	무엇이 일어나나
1	AttributeComparison.ComparisonTable	entities 가 matrix 의 행으로 퍼지고(P6), 관계의 desc ("협업 밀도에서 갈린다")가 of 첫 항목인 재택근무의 items 맨 뒤로 간다(P7)
2	Enumeration.List	여기도 안 되면 종단이다. entities 가 2개 이상이므로 종단 셋 중 List 가 성립한다. desc 는 여기서도 P7 로 보존된다

두 이유는 문서에서 다르게 다뤄진다. 구조 조건 미충족은 카테고리마다 다르므로 부록 G 의 행별 이유에 적혀 있고, 색인 공급 없음은 어느 후보에나 걸릴 수 있어 행별로 따지지 않는다. 대신 모든 사슬의 끝을 항상 성립하는 종단 셋으로 맺어 대비한다. 유형별 색인 공급 실측은 아직 없다. 부록 C 의 비중은 질의(수요) 분포이고 공급 근거가 아니다.

예시 3. 노드가 여럿일 때, 어느 노드부터 포기하나. 위 둘은 컴포넌트 하나 안의 일이고, 질의가 중첩(L3)이면 그전에 "어느 노드를 먼저 포기하나"를 정해야 한다. 다음 절의 N1~N5 가 그것이고, 한 단계씩 따라가는 예시는 6절 부록 7.5 에 있다.

이 사다리는 2026-08-13 리뷰에서 v2 로 고쳐졌다. v1 초안(김수현, 허재영, Google Sheet)을 정원석이 검토해 원칙 넷을 정했고, 그 넷이 지금 표의 모양을 정한다.

#	원칙
1	List 금지 원칙을 채택하지 않는다. "페이지역할, 단일메시지, 인물, KPI 11종은 List 풀백 금지" 제안(류민, 2026-08-05 계통도 분석)은 참고 자료로만 남긴다. 대안 없는 금지는 쓰지 않는다. 내용이 유실되거나 대체 자체가 안 되는 것이 List 로 대체되는 것보다 나쁘다고 판단했다
2	우선순위는 근접도로 정한다. 의미가 가까운 유형을 먼저 둔다. 의미의 근거는 단어장 정의문이 공유하는 핵심 개념이다. 대분류 일치는 강한 신호지만 배제 조건은 아니다. Funnel 과 Steps 는 대분류가 다르지만 "단계"를 공유해 가깝다고 본다. 반대로 개수나 축 형태가 닮았다는 구조적 이유만으로는 순서를 정하지 않는다
3	모든 행이 반드시 무언가로 끝난다. v1 에서 완전히 공란이던 6행(Cycle, FlowChart, Opening, SectionDivider, Ending, Flow)을 채웠다. 서로만 가리켜 상대가 없으면 둘 다 막히던 상호참조 4쌍(BigNumber 와 Gauge, Pyramid 와 NestedScope, Calendar 와 GanttChart, Location 과 Highlight)에도 각자 안전망을 붙였다
4	대체 후보 수 상한을 폐지한다. 종전 "최대 두 칸"은 실측 근거 없는 설계 판단이었다. 근접도 순서만 지키면 몇 칸이든 허용한다

사슬이 막히는 이유가 둘이고, 표는 하나만 행별로 적는다. '구조 조건 미충족'(성립 조건을 내용이 못 채운다)은 카테고리마다 다르므로 행별 이유에 적혀 있다. '색인 공급 없음'(성립 조건은 채우는데 그 유형으로 태깅된 컴포넌트가 색인에 없거나 알다)은 어느 후보에

나 걸릴 수 있어 행별로 따지지 않는다. 대신 모든 사슬의 끝을 항상 성립하는 종단 셋으로 맺어 대비한다. **유형별 색인 공급 실측은 아직 없다.** 부록 C 의 비중은 질의(수요) 분포이고 공급 근거가 아니다.

서로를 가리키는 쌍이 있다. ListingTable 과 ComparisonTable , Steps 와 Timeline , Pyramid 와 NestedScope , BigNumber 와 Gauge , BarChart 와 LineAreaChart , PieChart 와 DonutChart , DialogueChat 와 QuoteTestimonial , ProfileCard 와 ResumeCV 여덟 쌍이 서로를 후속 우선순위의 첫 자리로 가리킨다. 순환처럼 보이지만 순환이 아니다. 검색 모듈이 "자기 행만" 따라가므로 도착 카테고리의 행으로 넘어가는 일이 없다. 예를 들어 Steps 의 fallback 순서는 Timeline → Staircase → List 이고 Timeline 의 fallback 순서는 Steps → Staircase → List 인데, Steps 에서 출발해 Timeline 후보가 없으면 다음은 Staircase 이지 Timeline 행에 적힌 Steps 로 되돌아가지 않는다.

대표 fallback 체인 몇 가지다. 45종 전량은 5절 부록 G 에 있고, 엣지 131개 전량과 엣지마다 발동하는 projection 규칙은 부록 H 다.

카테고리	fallback 순서	한 번 따라갈 때 잃는 것
Enumeration.List	ListingTable	행과 열의 대응. List 자체는 entities 1개 이상이면 항상 구조적으로 성립하지만, 색인에 후보가 없으면 물러난다
ProcessFlow.Steps	Timeline → Staircase → List	Timeline (시간축), Staircase (상승 방향)를 차례로 잃는다
Hierarchy.Pyramid	NestedScope → Staircase → List	NestedScope 와 상호참조라 각자 안전망이 없으면 둘 다 막힌다
KPIKeyMetric.Gauge	BigNumber → PictogramChart → BarChart → HighlightedMessage	목표 대비 위치라는 의미가 단계마다 열어지다가 마지막에 강조 형으로 끝난다
RelationalComparison.SWOT	List → ListingTable	내부/외부, 긍정/부정의 2축 배치. Quadrant 나 ComparisonTable 경우 대신 List 를 1순위로 확정했다
TimelineSchedule.GanttChart	Timeline → Calendar → ListingTable → List	기간의 길이와 병렬 진행. 같은 TimelineSchedule 대분류인 Calendar 를 안전망으로 먼저 거친다

차트 계열은 별도의 작은 순환 구조를 이룬다. PieChart , TreeMap , PictogramChart , StackedBarChart , ComboChart , RadarChart , BubbleChart 가 모두 BarChart 로 모이고, BarChart 와 LineAreaChart 는 서로를 가리키며, 최종적으로 ComparisonTable 을 거쳐 List 로 끝난다.

엣지 수의 변천. fallback 그래프의 엣지(카테고리 대 카테고리 연결) 수는 여러 차례 다시 계산됐다. 2026-08-04 에 51개에서 75개로(행을 경로로 잘못 읽어 누락된 엣지 25개를 보정), 2026-08-17 에 75개에서 111개로, 다시 125개로 늘었고, 2026-08-21 ComboChart 개정으로 131개가 됐다. **현재 값은 131개이고 부록 H 가 전량이다.** 문서에 남아 있는 "128개"는 전부 이력 서술이라 현재 값을 주장하는 자리가 아니다(부록 F 4번, 2026-08-29 오답 판정).

복잡도 레벨 셋: 질의가 어디까지 복잡해질 수 있나

앞 절까지의 모양들은 **Visual Summary 하나가 가질 수 있는 위상의 범위로 셋으로 묶인다.** 그 범위를 가리키는 이름이 복잡도 레벨 이고, SSOT 는 규약 8.8절이다. **레벨은 산출물 한 건의 모양을 분류하는 이름이 아니라 생성기에 허용된 범위의 이름이다.** 그래서 누적 이고(L1 < L2 < L3), L3 를 컨 생성기가 컴포넌트 하나짜리 Visual Summary 를 내는 것은 정상이다.

'질의'라고 부르지 않는다. 복잡도는 **생성기 산출물(Visual Summary)** 의 성질이고, 그것이 검색용 질의로 바뀌는 것은 질의 변환기 (query.ts)를 거친 뒤다. 이 문서가 '질의 쪽'이라고 쓸 때는 색인 쪽과 대비하는 진영 이름이지 변환기 산출물이 아니다.

레벨	이름	최상위 components[]	entity.components[]
L1	단일 컴포넌트 (single component)	길이 1	안 쓴다
L2	병치 위상 (juxtaposition topology)	여럿 허용	안 쓴다
L3	전체 위상 (all topology)	여럿 허용	쓴다

레벨을 가르는 축은 하나다. 컴포넌트 경계를 넘는 계층(`entity.components[]`)을 쓰느냐. `entity.children[]` (같은 컴포넌트 안 계층)은 어느 레벨에서나 쓴다. 그것은 한 컴포넌트 내부의 내용 구조이지 위상이 아니다. L1 과 L2 를 가르는 것은 그다음 축인 최상위 원소 수다.

레벨	fallback 이 무엇을 쓰나	그 레벨을 켜려면 무엇이 있어야 하나
L1	앞 절의 강등 사다리만. 복합 순서는 쓸 일이 없다	없다. 지금 도는 것이 이것이다
L2	사다리가 최상위 원소마다 독립으로 돈다. 빗길 부모가 없어 N1~N5는 쓰지 않는다	조각마다 검색을 호출하고 결과를 한 페이지로 조립하는 오케스트레이션(I-11). 색인의 후보 단위가 컴포넌트 하나라 한 후보가 최상위 원소 둘을 함께 담는 일이 없다
L3	다음 절의 N1~N5 전부와 규칙 여섯	위의 것 전부에 더해, 생성기가 <code>entity.components[]</code> 를 채우는 정책, 중첩 후보의 대치 로직(D-7 이 "기존 조합 컴포넌트와는 별개의 대치 로직이 필요하다"고 적은 것), N3~N4 반복으로 최상위 원소가 늘어날 때의 상한(B-32)

생성기가 돌이고 지금 레벨이 서로 다르다. 섞어 읽으면 "지금 어디까지 되는가"가 뒤집힌다.

생성기	지금 레벨	근거
Summary 생성기(실서비스, 허재영·황현진)	L2 까 지	규약 1절이 "생성기는 <code>entity.components[]</code> 를 채우지 않는다"고 정한 그 생성기다. 실사용 로그 10건 실행에서 입력 10건이 컴포넌트 15개를 냈으므로 L2 는 관측된다(4절)
PoC 생성기(정원석, MOR-1910)	이미 L3	프롬프트에 <code>##</code> 이중 컴포넌트 절이 있어 포함을 <code>entity.components</code> 로 적으라고 지시하고, 골든셋 v1 에 중첩 산출물이 들어 있다

그래서 "L3 는 자리만 열려 있다"는 말은 생성 쪽에는 더 이상 맞지 않는다. 비어 있는 것은 그다음 두 칸이다. 조각마다 검색을 호출하고 결과를 조립하는 오케스트레이션(I-11)과 중첩 후보의 대치 로직(D-7)이다. 4단계 실서비스에서 어느 레벨까지 켜지는 정해진 바 없다.

레벨을 통제하려면 프롬프트가 블록으로 떼어져야 한다. 레벨을 올리는 지시가 흩어져 있으면 "L2 로 내려라"가 한 군데를 지우는 일이 아니라 문장을 골라내는 일이 되고, 그때부터 프롬프트 판본마다 실제 레벨이 달라진다. 지금은 절반쯤 되어 있다. 위상 지시가 `##` 이중 컴포넌트 한 절에 뭉쳐 있어 물리적으로는 한 덩어리지만, `##` 컴포넌트 블록 절에 "`entity.components` 는 포함 관계를 적을 때만 쓴다" 한 줄이 밖에 남아 있고, 포함(L3)과 병치(L2)가 한 절 안에 섞여 있어 레벨 경계로 갈리지 않으며, 프롬프트 조립(`fillPrompt`)이 `{ {TOKEN} }` 치환만 해서 블록을 넣고 뺄 수단 자체가 없다. 필요한 모양과 검사 방법은 규약 8.9절에 적었다. 아직 구현되지 않았다.

'병치'라는 말이 두 층에서 쓰인다. 질의 쪽 병치(여기의 L2)는 최상위 `components[]` 원소가 여럿인 것이고 컴포넌트 경계가 이미 갈려 있다. 색인 쪽 Juxtaposition (규약 8.7절 위상 관계)은 이중중첩으로 판정된 한 컴포넌트 안에서 서로 다른 semantic category 의 시각 영역이 겹치지 않고 나란히 놓인 것이다. 방향이 반대다. 앞은 나눠서 찾는 쪽이고 뒤는 이미 하나로 뭉쳐 있는 것을 부르는 이름이다. L2 질의의 조각들이 각각 검색돼 한 페이지에 놓이면 겉모습은 닮지만 후보 단위가 다르므로 같은 것이 아니다.

레벨과 카테고리 강등 칸 수는 서로 다른 예산이다. L3 질의라고 해서 카테고리를 더 많이 내릴 수 있는 것도, 덜 내려야 하는 것도 아니다. 다음 절 N-R4 가 정한 것과 같은 구분이다.

복합 구조 fallback: 노드가 여럿일 때 어느 노드부터 포기하나

앞 절의 사다리는 **노드 하나**의 카테고리를 내리는 규칙이다. 질의가 중첩(`entity.components[]`)이나 병치(최상위 `components[]` 여럿)로 노드를 여럿 가지면, 그것과 별개로 **어느 노드를 먼저 포기하는지**를 정해야 한다. 2026-08-28 에 정원석이 확정했고 SSoT 는 `spec/capability-table.md` 3.6절이다. 그전까지 이 자리는 `design/content-binding-contract.md` 3.4절에 "아직 한 문서에 못 박지 않았다"로 열려 있었다.

다섯 단계다. 위에서부터 시도하고 후보가 나오면 멈춘다.

단계	무엇을 하나	없는 것
N1	질의 트리 전체를 그대로 담은 후보를 색인에서 찾는다	없다
N2	트리 모양을 고정한 채 노드마다 앞 절의 fallback 순서를 따라 내려간다	노드마다의 카테고리 표현
N3	가장 바깥 부모를 벗겨 자식을 최상위 <code>components[]</code> 로 올리고 N1 부터 다시 시작한다	벗긴 부모의 카테고리 표현과 형제 사이의 공간적 인접성
N4	남은 노드가 모두 단일 카테고리가 될 때까지 N3 을 반복한다	위와 같다. 깊이마다 한 번씩
N5	규약 1절 "중첩은 평탄화된다"대로 자식 <code>entities</code> 를 부모 <code>items</code> 로 편다	자식 전부의 카테고리 표현

사다리는 최상위 `components[]` 원소마다 독립으로 돈다. 최상위 병치는 이미 컴포넌트 경계가 갈려 있어 벗길 부모가 없고, 색인의 후보 단위도 컴포넌트 하나이므로 한 후보가 최상위 원소 둘을 함께 담는 일이 없다.

왜 위부터 벗기나. 부모를 벗기는 것과 자식을 벗기는 것이 대칭이 아니다. 부모를 벗기면 질의 1개가 질의 여럿으로 갈라지고, 질의가 작아지므로 후보 풀이 커진다. 자식의 내용과 카테고리 표현은 전부 남는다. 반대로 자식을 벗기면 벗긴 자식의 내용이 지워지거나 부모로 접히는데, 부모는 여전히 복합 구조라 매칭 난이도가 그대로다. **내용을 지불하고 검색 가능성을 얻지 못한다.** 그래서 N3 은 위부터만 간다.

왜 평탄화가 마지막인가. 규약 1절은 종전에 "중첩은 평탄화된다"만 정하고 있었고, 그것이 유일한 확정 기법이였다. 평탄화와 떼어내기(N3)는 **둘 다 텍스트를 하나도 버리지 않는다**. 다른 것은 표현이다. 평탄화는 자식 전부의 카테고리 표현을 잃는 대신 부모의 배치를 지키고, 떼어내기는 자식의 카테고리 표현을 지키는 대신 부모의 배치를 잃는다. 표현을 더 많이 남기는 쪽을 먼저 두기로 정했으므로 떼어내기가 앞이고 평탄화가 뒤다. 평탄화는 앞 절의 마지막 fallback 셋으로 끝나므로 반드시 성립하고, 그래서 복합 구조에서도 빈 결과가 나오지 않는다.

규칙 여섯이 붙는다.

#	규칙
N-R1	벗긴 부모의 <code>label</code> , <code>items</code> , <code>relations[]</code> . <code>desc</code> 는 버리지 않고 승격된 첫 조각의 첫 entity <code>items</code> 맨 뒤로 보낸다. P3 과 P7 과 같은 방식이다
N-R2	형제는 큰 묶음부터 그리디로 나눈다. 4형제면 <code>[4]</code> , <code>[3,1]</code> , <code>[2,2]</code> , <code>[2,1,1]</code> , <code>[1,1,1,1]</code> 순이다. 연속한 형제만 묶고 순서를 바꾸지 않는다
N-R3	색인 태깅 어휘의 <code>Unclassified</code> 는 카테고리가 아니라 경계다. 질의 <code>type</code> 이 될 수 없으므로 그 자리를 카테고리로 검색하지 않고 부모의 슬롯 경계로만 쓴다
N-R4	N 단계 수와 카테고리 fallback 칸 수는 서로 다른 예산이다. N3 한 번이 카테고리 강등 칸을 소모하지 않는다
N-R5	제안, 미확정. 부모의 대분류가 <code>Hierarchy</code> 면 N3 을 건너뛰고 N5 로 간다. 깊이 3 이상 계층 질의가 실측에 없어 판정 근거가 없다(B-31)

#	규칙
N- R6	N2 의 조합 순서는 강등 칸수 합이 작은 것부터다. 합이 같으면 뿌리에서 먼 노드를 먼저 내린다. 뿌리를 내리면 자식이 들어갈 슬롯이 함께 바뀌기 때문이다

N1 은 지금 정책상 언제나 비어 있다. D-7 이 이중중첩 템플릿을 검색 풀에서 제외하기로 확정했으므로(2026-08-11 체크인, 전체 약 2%), 뿌리가 Unclassified 인 템플릿이 후보로 걸리지 않는다. 같은 이유로 N2 도 이중 구조에서는 후보가 0이다. 모양이 이중중첩인 채로 남기 때문이다. **즉 이중 질의는 실질적으로 N3 에서 시작한다.** 동종중첩(자식 type 이 부모와 같은 경우)은 D-7 밖이라 N1 과 N2 가 살아 있다. D-7 을 되돌릴지는 열려 있다(D-8). D-7 의 결정 근거가 "대부분 조합 Fallback 만으로 해결될 것으로 예상"이었으므로, N3 부터 N5 의 결과 품질이 그 예상의 검증 자료가 된다.

이 확정으로 닫힌 것과 새로 열린 것.

항목	어떻게 됐나
질의 쪽 중첩을 평탄화한 뒤 fallback 할지, 구조를 유지한 채 노드 태그로 자를지 (I-12)	닫혔다. 유지한다
평탄화와 떼어내기 중 어느 쪽이 내용을 더 잘 보존하는지	닫혔다. 떼어내기가 먼저다
병치를 쪼개고 조각마다 검색하고 결과를 조립하는 코드 (I-11)	열려 있다. 검색 재료만 있고 오케스트레이션이 없다
이중중첩 템플릿을 검색 풀에 되돌릴지 (D-8)	열려 있다
계층 대분류에서 N3 을 건너뛸지 (B-31)	열려 있다
N3 과 N4 반복으로 최상위 컴포넌트 수가 늘어날 때 상한 (B-32)	열려 있다

한 단계씩 따라가는 예시는 6절 부록 7.5 에 있다.

structure projection: fallback 이 넘어갈 때 내용을 옮기는 규칙

fallback 은 카테고리를 바꾸는 일이라, 출발 카테고리가 쓰던 필드를 도착 카테고리가 못 읽는 경우가 생긴다. 이때 "어디로 옮기는지"를 정하는 것이 projection 규칙 여덟 가지(P1~P8)이고, 미확정 제안이 둘(P9, P10) 더 있다.

규칙	무엇을 하나
P1	<code>entities</code> 는 그대로 간다. 순서와 <code>label</code> , <code>items</code> , <code>children</code> 은 어느 엣지에서도 손대지 않는다
P2	도착이 선언하지 않은 <code>params</code> 는 버린다. 표현 선택이지 내용이 아니기 때문이다
P3	도착이 안 읽는 내용 필드는 그 entity 의 <code>items</code> 맨 뒤에 이름: 값 형태로 붙인다
P4	<code>matrix</code> 를 못 읽는 곳으로 가면 <code>entities</code> 로 편다. 행마다 entity 를 만들고 첫 열이 <code>label</code> , 나머지 열이 <code>items</code> 로 들어간다
P5	<code>measure</code> 를 못 읽는 곳으로 가면 <code>value</code> 와 <code>unit</code> 을 entity 로 옮긴다. 나머지는 P3 을 따른다
P6	<code>entities</code> 만 있는데 <code>matrix</code> 가 필요하면 행으로 편다. <code>label</code> 이 첫 열, <code>items</code> 가 나머지 열이다
P7	도착이 관계를 못 그리면 <code>relation.desc</code> 를 <code>of</code> 첫 항목의 <code>items</code> 맨 뒤로 보낸다
P8	도착의 개수 상한을 넘으면 초과분을 마지막 항목의 <code>items</code> 로 접는다
P9 (제안, 미확정)	<code>measure</code> 를 <code>matrix</code> 가 필수인 곳으로 보내면 <code>value</code> 를 1행짜리 <code>matrix</code> 로 접는다(Gauge → PictogramChart 근거)

규칙	무엇을 하나
P10 (제안, 미 확정)	<code>matrix</code> 를 <code>measure</code> 가 필수인 곳으로 보내면 대표값 1행을 <code>value</code> 로 쓰고 나머지는 <code>items</code> 로 흘러보낸다. 대표값을 고르는 기준(첫 행, 합계, 최대값)이 아직 없다

내용은 버리지 않고 표현만 버린다. P2 만 버리는 규칙이고 나머지는 전부 옮기는 규칙이다. `orientation`, `apex`, `icon` 같은 표현 선택은 도착 카테고리에서 뜻이 없으므로 버려도 내용이 줄지 않는다.

실제 엣지에 규칙이 어떻게 겹쳐 적용되는지 예시 셋.

출발 → 도착	적용 규칙	무엇이 옮겨지나
<code>BarChart</code> → <code>List</code>	P1 P2 P3 P4	<code>matrix</code> 의 행이 <code>entities</code> 가 되고, <code>unit</code> 과 <code>value</code> 가 <code>items</code> 로 간다. <code>orientation</code> , <code>sort</code> 는 표현 선택이라 버린다
<code>BigNumber</code> → <code>HighlightedMessage</code>	P1 P2 P3 P5 P8	<code>value</code> 와 <code>unit</code> 이 entity로 옮겨지고, <code>delta</code> 와 <code>emphasis</code> 도 함께 간다. 1개를 넘으면 나머지는 마지막 항목의 <code>items</code> 로 접는다
<code>ComparisonTable</code> → <code>Versus</code>	P1 P2 P4 P8	<code>matrix</code> 의 행이 <code>entities</code> 가 되고, 2개를 넘으면 나머지를 마지막 항목에 접는다 (Versus는 정확히 2가 조건이므로)

alternatives 와 fallback 순서의 차이

둘 다 "다른 카테고리로 갈 수 있다"는 점은 같지만 보는 것이 다르다.

	fallback 순서	alternatives
어디 있나	capability 상수. 카테고리마다 미리 박혀 있다	생성물. 입력마다 다르다
무엇을 보나	카테고리 대 카테고리만 본다	그 입력의 내용을 본다
언제 정하나	규약을 쓸 때 한 번	생성할 때마다

fallback 순서는 정적이다. `Gauge` → `BigNumber` → `List` 는 게이지에 무엇이 담기든 같다. 반면 생성기는 입력 내용을 이미 읽었으므로, 같은 `Hierarchy.Pyramid` 라도 층마다 크기 차이가 의미를 지니면 `Progression.Staircase` 가 낫고 그냥 3단 분류면 `Enumeration.List` 로 가도 손해가 적다는 것을 알 수 있다. 이런 판단을 `alternatives` 에 미리 적어 두면 검색이 정적 fallback 순서보다 먼저 참고한다. **다만 이것은 가정이고 아직 실측이 없다.** 생성기가 고른 `alternatives` 가 정적 순서보다 실제로 나온 결과를 내는지는 검증된 적이 없어서 optional 로 열어 두고 생성기 정책으로 켜고 끌 수 있게 했다.

꾸밈구조(decoration): 등급 하나에서 축 여럿으로

`structure` 가 내용이고 `params` 가 카테고리별 표현 선택이라면, 꾸밈구조는 **카테고리와 무관한 표현 선호도**. 그림 장식이 붙은 컴포넌트를 원하는지, 배경이 깔린 것을 원하는지를 적는다. 설계 의도의 SSoT 는 `design/decoration-model.md` 이고 확정 표기는 규약 2.1절, 2.2절, 2.7절이다.

7/24 회의는 이것을 누적 등급 셋(N , $N+\alpha$, $N+\beta$)으로 제안했다. 같은 레이아웃 하나에서 여러 벌을 만들어 두고 사용자 수요에 맞는 것을 고른다는 발상이다. 세 등급의 원 정의는 다음과 같다(원문은 `archive/schema-v0.4.md` 4-1절에 보존한다).

등급	원 정의	왜 그 등급인가
N	그 카테고리의 최소 필요 조건만 만족하는 버전. 예를 들어 Steps 라면 단계 항목만 있는 것	필수. 언제나 만든다

등급	원 정의	왜 그 등급인가
N+α	N 에 타이틀과 디스크립션까지 살린 버전	입력에 제목·설명이 있는 사용자를 위한 것
N+β	N+α 에 요소 배치와 장식까지 살린 버전	장식은 지을 수 있어도 없는 것을 더하기는 어려우니, 장식 보존본을 처음부터 함께 내보낸다

$N < N+\alpha < N+\beta$ 로 뒤 등급이 앞을 포함하는 모양이다. 2026-08-03 에 정원석이 이 모양을 기각하고 축 여럿으로 바꿨다. **기각 근거가 다섯이다.**

#	왜 등급 하나로는 안 되나
1	한 등급 안에 서로 다른 것이 섞인다. 회의 원문은 N+β 를 "요소 배치와 장식까지"로 적는데, 배치가 정교한데 그림이 없는 컴포넌트와 그림이 많은데 배치가 단순한 컴포넌트를 구분해 요구할 수 없다. N 과 N+α 의 경계는 더 나뉜다. 타이틀과 디스크립션 유무는 내용 이고 장식 양은 표현 선택 인데 한 축에 섞여 있다
2	누적이라서 표현할 수 없는 조합이 있다. "꾸밈은 많지만 디스크립션은 없는 것"에 등급 이름이 없다
3	축이 늘 때 끼워 넣을 자리가 없다. 테두리나 그림자가 새로 필요해지면 "N+β 안쪽"이라고 말할 수밖에 없고, 그 판단이 문서마다 달라진다. 축으로 두면 새 축을 열고 기본값을 <code>indifferent</code> 로 두면 기존 동작이 안 바뀐다
4	채점 실물이 이미 축별이다. <code>comp-sign</code> <code>DEFAULT_SCORING_WEIGHTS</code> 가 <code>illustrationBonus</code> , <code>titleFontSizePenalty</code> , <code>styleMismatch</code> , <code>visualSiblingSplit</code> 로 항목별로 나뉘어 있다. 등급으로 뭉치면 "rich 를 요구했는데 왜 이 점수인가"를 설명할 수 없다
5	N 은 애초에 꾸밈 축이 아니다. N 의 '최소 필요 조건'은 <code>capability</code> 성립 조건과 같은 대상을 가리키므로, N 을 등급 최하단에 두면 같은 사실이 두 곳에 적힌다

축은 둘이고 각 축이 RLSC role 하나에 대응한다. 축 하나가 role 하나에 붙으면 셋이 정해진다. 추출이 그 role 을 이미 붙이므로 표식을 새로 만들지 않고, 요소 대치가 지을 노드를 role 로 특정할 수 있고, 새 꾸밈 role 이 생기면 축을 하나 더 열면 된다.

축	대응 RLSC role	role 정의
<code>ornament</code>	<code>Element.Decoration</code> , <code>LayoutContainer.Decoration</code>	"시각적 꾸밈 목적, 내용 대치 시 변경 불필요"
<code>background</code>	<code>Element.Background</code> , <code>LayoutContainer.Background</code>	"Parent position 거의 가득 채움, 크기 늘림 선호"

축에서 뺀 것이 셋이다. `description` 과 `highlight` 는 내용이라 `structure` 와 `entity.emphasis` 가 담고, `arrangement` 는 노드에 붙는 성질이 아니라 노드를 놓는 방식이라 role 대응이 없다(배치를 보는 채점 항목은 `areaRatio`, `offCenterPenalty`, `visualSiblingSplit` 로 이미 있다). `Element.Marker` 는 처음에 판정을 미뤘다가 2026-08-04 에 **축이 아니라고 정했다**. 마커는 순서를 담고 순서는 내용이다(마커 보유 540건 중 534건이 기하 순서와 일치). `Element.Separator` 는 순서도 내용도 담지 않아 **세 번째 축 후보로 열려 있다**.

값은 셋이고 **require** 가 없다.

값	뜻	왜 있나
<code>prefer</code>	그 role 노드를 가진 후보를 앞에 둔다. 없어도 후보로 남는다	탈락시키지 않는다
<code>avoid</code>	그 role 노드를 가진 후보를 뒤로 보낸다. 있어도 후보로 남는다	"꾸밈 없는 깨끗한 것"을 요구하는 자리
<code>indifferent</code>	보지 않는다. 기본값	안 적은 것과 같은 뜻이지만, 위 층에서 물려받은 값을 취소하려면 명시할 수 있어야 한다

`require` 를 두지 않은 근거는 검색 폴 분포다. VennDiagram, HierarchicalTree, NestedScope, Pyramid 가 각 164건 이하이고 데이터 차트가 각 51건 이하인데, 여기에 꾸밈까지 필수로 걸면 후보가 0이 되어 전부 `List` 로 강등된다. 수치 점수(`ornament: 0.8`)도 두지 않고 **앞으로도 두지 않는다**(2026-08-29 확정). 채점 코드를 건드리지 않기로 한 결정과 어긋나고 `entity.weight (number)`가 다른 뜻으로 이미 있다.

프리셋 문자열을 스키마 값으로 두지 않는다. N 을 `minimal`, $N+\beta$ 를 `rich` 로 부를 수는 있지만 문서와 회의에서만 쓴다. 이유가 셋이다. 문자열과 객체 두 모양을 허용하면 소비자가 분기해야 하고, 프리셋은 축으로 펼쳐지므로 정보를 더 담지 않으며, 축이 하나 늘 때마다 `rich` 가 가리키는 범위가 달라져 **등급을 기각한 것과 같은 결함이 되살아난다.**

선언 층이 넷이고 축마다 따로, 구체적인 쪽부터 해석한다. 층 전체를 통째로 갈아치우지 않고 축 단위로 병합하는 것이 중요하다. 그래야 "page 는 꾸밈을 다 원하는데 이 컴포넌트만 배경 빼고"를 표현할 수 있다.

순서	어디	누가 채우나
1	<code>component.decoration</code>	생성기. 1단계에서는 안 채운다
2	<code>page.decoration</code>	생성기. 1단계에서는 안 채운다
3	카테고리 기본값	capability 상수. 미정
4	시스템 전역 기본값	검색 설정. 1단계에서는 여기만 채운다
5	<code>indifferent</code>	규약 기본값

1단계는 시스템 전역 기준만 잡고 Summary 는 생성하지 않는다. 스키마에 `optional` 로 자리만 연다. 근거가 다섯이다. 지금 소비할 주체가 없고(요소 대치가 아직 동작하지 않으며 내용 대치는 꾸밈에 관여하지 않는다), 색인이 대조할 role 표식을 받을 경로가 없고(B-22), 지금 필요한 것은 "꾸밈 많은 것을 우선"이라는 전역 정책 하나로 되고, 생성기에 필드를 늘리면 토큰과 생성 실패 자리가 늘고(`entity.id` 를 뺀 근거가 토큰 5.9% 증가였다), `entity.components[]` 와 page 수준 `relations` 가 이미 같은 방식으로 열려 있다는 선례가 있다.

그래서 지금 꾸밈 축이 쓰이는 곳은 검색 순위 하나다. 꾸밈 노드를 지우거나 넣는 것은 요소 대치가 생긴 뒤에 할 수 있다. 색인에는 장식 보존본을 **한 벌만** 둔다. 등급 3단으로 벌을 나누면 comp-sign 변형 3종과 곁해져 컴포넌트당 최대 9벌이 되는데, 보존본 한 벌이면 3벌에서 그친다. 근거는 7/24 비대칭 원칙이다. **장식은 삭제할 수 있어도 없는 것을 추가하기는 어렵다.**

이렇게 쓰지 않는다.

안티패턴	왜
<code>decoration: { ornament: require }</code>	<code>require</code> 는 없는 값이다. 후보가 0이 되어 전부 <code>List</code> 로 강등된다
<code>decoration: { ornament: 0.8 }</code>	수치 점수를 두지 않는다. 앞으로도 두지 않는다
<code>decoration: { description: prefer }</code>	<code>description</code> 은 축이 아니다. 설명 텍스트는 내용이고 <code>structure</code> 가 담는다
<code>decoration: rich</code> , <code>decoration: minimal</code>	프리셋 문자열을 받지 않는다. 축으로 적는다
<code>decoration</code> 과 <code>style</code> 을 섞어 적는 것	키 이름이 다르다. <code>decoration</code> 은 RLSC role 과 겹치고 <code>style</code> 은 스타일 대치가 쓴다
생성기가 <code>decoration</code> 를 채우는 것	1단계에서는 안 채운다

아직 정하지 않은 것이 여덟이다. `Element.Separator` 를 세 번째 축으로 열지, `LayoutContainer.PageHeader` 와 `PageFooter` 를 어떻게 볼지(이 둘은 짝이 되는 `Element` role 이 없어 "축 하나가 `Element.X` 와 `LayoutContainer.X` 쌍에 대응한다"는 전제가 깨진다), DECO 노드 분류가 `Element.Decoration` role 과 같은 집합인지, 카테고리 층을 둘지, 45종의 카테고리

리 기본값, 색인이 role 표식을 받는 경로(B-22, 이것이 먼저 풀려야 검색이 대조할 수 있다), `prefer` 와 `avoid` 를 점수가 아닌 **형태로** 검색이 순위에 어떻게 반영하고 fallback 보다 먼저 보는지, comp-sign 변형 3종을 대치 시점으로 옮길지(B-19)다. 실물 `Decoration` 형 정의와 해석 예시 넷은 5절 부록 K 에 있다.

포맷 실측과 다시 볼 조건

`docs/projects/mordor/visual-summary/spec/format-alternatives.md` 가 YAML 과 KDL 을 비교한 사전 조사 결과를 담는다. 공개 벤치마크에서 확인된 것은 넷이다. JSON 의 파싱 유효율이 100%, S-expression 이 95.8%, YAML 이 91.7% 였고 YAML 은 구조 오류 시 "조용한 손상(silent corruption, 파싱은 성공했지만 값의 의미가 조용히 바뀌는 것)"이 상대적으로 많다는 점, 텍스트에서 YAML 을 만드는 일 자체는 모델이 이미 90% 이상 능숙한 쉬운 작업이라는 점, 11개 모델과 4개 포맷을 비교한 실험에서 포맷 자체가 정확도에 유의한 영향을 주지 않았다는 점(모델별 민감도 -7.7%~+2.7%), KDL 쪽은 언어 스펙과 구현체만 있고 생성 신뢰성 벤치마크가 0건이라는 점이다.

이 조사에서 내린 실무적 결론은 "공개 사례로는 우열을 가릴 수 없으니 사전 확률(YAML 우세)로 초안을 확정하고, 자체 실측은 사후 검증으로 돌린다"이다. YAML 을 선택했으므로 감시해야 할 것은 "파싱은 됐는데 타입이 뒤바뀐" silent corruption 이다. 이 지표는 실패로 보이지 않기 때문에 처음부터 계속에 넣었다(구현은 `parse.ts` 의 `parseYamlWithDiagnostics()` , 4절 참고).

사후 검증 설계(아직 실행 전). 같은 스키마 N개 케이스를 YAML, KDL, JSON 각각으로 생성해 파싱 실패율, 스키마 위반율, silent corruption 율, 재현성(temperature 0 에서 k회 반복했을 때 출력 일치율) 넷을 재는 계획이다. 다른 마감이 걸려 있지 않을 때 돌린다.

자기정합 검사

2026-08-04 에 스키마, capability 테이블, YAML 예시가 서로 어긋나지 않는지 검사 14가지를 기계로 돌렸고, **2026-08-29 에 전량 다시 돌렸다**(`docs/projects/mordor/visual-summary/spec/schema/README.md`). 2026-08-04 실행에서 결함 다섯 건을 찾아 고쳤고 그 과정에서 열린 결정 두 건(B-27, B-28)이 새로 생겼다. 2026-08-29 실행에서는 생성물 둘의 미반영(부록 F 1번과 2번)과 예시 하나를 고쳤고, **검사 정의 자체도 돌 고쳤다**(F2 폐기, F3 을 "끝나는가"에서 "거치거나 자신이 그 셋인가"로). fallback 순서는 이때 그래프 형태로 확정됐다(B-26). 이 검사는 코드에도 그대로 옮겨져 있다. `src/lib/visual-summary/capability.test.ts` 가 `spec/schema/capability.json` 과 45종 전량을 대조해서 문서와 코드가 어긋나지 않는지 확인한다. 다만 `docs/` 폴더가 비공개 submodule 이라 CI 체크아웃에는 없어서, 이 대조는 로컬에서만 돌고 CI 에서는 건너된다.

일정과 작업 상태

지금 한 줄로 말하면

규약(스펙)은 2026-08-02 에 확정됐고, 질의 갈래 PoC(생성 → fallback → 검색 → 대치 → 채점)와 I.I.S.(증분 정보 구조화기) 색인 갈래 코드는 이미 돈다. RAT pod(류민 소관)의 I.I.S. 본편은 기다리지 않고, PoC 가 같은 출력을 계속 대행 생산한다(D-5, 2026-08-22 결정). 그래서 지금 상태를 정확히 읽으면 "스펙과 PoC 골격은 서 있고, 내용 대치 contract, 검수, 통합, 규모 확장이 다음 병목"이다.

로드맵 5단계 진행 상황

단계	무엇이 바뀌는가	목표 시점	지금 상태
1단계 (현행)	Markdown Summary, 임베딩 검색	지금	운영 중, 변동 없음
2단계	컴포넌트에 semantic tag	2026-08-18 로 잡혀 있었으나 폐기	2026-08-07 사실상 스킵 확정. 4단계로 직행
3단계	YAML + structure 검색	없음	실행하지 않음, Jira 상 Cancelled (2026-08-06)
4단계	RLSC node 태그로 정확한 범위 특정	9월 말 dev/stg, 10월 중순 production 판정	PoC 진행 중. 질의 갈래는 Eregion 화면에서 동작, I.I.S. 는 PoC 가 대행 생산
5단계	node 에 semantic category 속성까지	미정	속성 추출(T-7) v1 코드는 있으나 정확도 검수 미완

2단계를 스킵한 경위. 2026-08-03 에 내용시각화 제품의 8/25 production 배포가 연기로 정해지면서, 2단계 배포(MOR-1774)의 8/18 마감에 이 연기에 포함되는지가 미정이었다. 2026-08-07 김수현님이 Confluence 코멘트로 "8/18 마감은 연기에 포함된다. 8/18 은 폐기하고 4단계로 직행한다"고 답하면서 정리됐다. 같은 자리에서 "List 편중 81.1% 를 그대로 두고 2단계를 뱌지"(TODO D-2)도 "2단계를 안 하니 결정할 것이 없다"로 닫혔다. MOR-1774 티켓 자체는 Jira 상 Cancelled 로 전환되지는 않고 Draft 상태를 유지한다.

3단계를 건너뛴 이유. 2026-07-24 회의에서, 3단계에서 개발할 내용 대치 로직이 4단계에서는 버려질 수 있다고 판단해, 정보가 확실한 4단계로 바로 가는 편이 낫다고 봤다. 3단계였던 "Summary 생성기를 YAML 로 바꾸는 것"은 2026-08-02 결정으로 4단계 범위에 편입됐다. 그 결과 4단계는 생성기 교체와 node 태그 도입과 검색 로직 변경이 한 번에 일어난다. 이것이 품질 저하 시 원인을 세가지 중 무엇으로 볼지 가리기 어렵게 만드는 위험이라, 4단계 착수 때 단계별 계측 지점을 설계로 보완하기로 했다(TODO C-3).

Jira 구조

상위 Artifact 는 [MOR-1923](#) 이고 2026-08-06 에 신설됐다. 그 아래 마일스톤 [MOR-1776](#)(4단계)에 Subtask [\[4-1\]](#) 부터 [\[4-15\]](#) 까지가 실행 트랙이다.

구간	Subtask	무엇을 다루나	담당
[4-1] ~ [4-3]	MOR-1909~1911	생성기 프롬프트와 fallback map 정의	허재영, 황현진
[4-4] ~ [4-9]	MOR-1924~1929	검색 질의 변환기 스펙, capability 상수, 대치 PoC, 계측 설계, 평가셋, 판정 지표	정원석
[4-10] ~ [4-15]	MOR-1932~1937	내용 대치 contract 와 열린 결정 해소	정원석, 이석

규약 정의 티켓 [MOR-1773](#)은 Done, 3단계 마일스톤 [MOR-1775](#)는 Cancelled, 2단계 마일스톤 [MOR-1774](#)는 위에서 설명한 대로 사실상 스킵 상태에서 due date 필드만 잠정 유지 중이다.

할 일 SSoT(TODO.md)의 절 구조

`docs/projects/mordor/visual-summary/TOD0.md` 가 할 일을 열 절(A부터 J까지)로 나눈다. 항목 전량은 5절 부록 I에 있다. 열린 항목은 2026-08-28 기준 44건이다.

절	무엇을 다루는가	마감	남은 것
A	규약 확정	2026-08-07	4건
B	규약의 열린 결정 (전체 32건)	A에 종속	16건
C	2단계 배포	사실상 스킵 (2026-08-07)	5건
D	스크드 차원에서 정할 것	각각 다름	3건 (D-4, D-6, D-8)
E	7/24 회의 후속	미정	3건
F	사후 검증 (마감 후로 미룸)	마감 후	2건
G	최근 해소된 것의 기록	—	없음(기록만)
H	문서 정합 백로그	—	3건
I	내용 대치 contract	미정	9건
J	생성 모델 파인튜닝	착수 전	8건

품질을 좌우하는 것은 I절(내용 대치 contract)이다. 실행 1순위는 I-8(평가 pool bake, 여러 모듈이 같은 컴포넌트 부분집합을 기준으로 비교되도록 미리 데이터를 만들어 두는 작업)이다.

확정된 결정과 폐기된 결정

날짜	결정	근거
2026-08-01	Summary 형식을 YAML로 초안 확정	2절 참고
2026-08-02	규약 v0.5 확정 (마감 8/7 보다 닷새 앞섬)	<code>spec/schema-v0.5.md</code>
2026-08-03	꾸밈구조와 계층 결정(B-2, B-3)이 뒤집힘	<code>design/decoration-model.md</code>
2026-08-04	규약 표기에 꾸밈구조 반영, 자기정합 검사 14가지 실행	결함 5건 발견, 열린 결정 2건(B-27, B-28) 신규
2026-08-06	Jira 재편, Artifact MOR-1923 신설, MOR-1775 Cancelled	<code>plan/jira-artifact-blueprint.md</code>
2026-08-07	2단계 배포 마감 폐기, 4단계 직행 확정	김수현 Confluence 코멘트
2026-08-12	Eregion PoC 화면 세워짐, 질의 갈래 end-to-end 동작	<code>plan/eregion-poc.md</code>
2026-08-13	fallback map v2 대부분 반영	<code>plan/fallback-map-review-2026-08-13.md</code>
2026-08-15	I.I.S. 계열 이름을 Semantic Component Boundary(4단계)와 Semantic Component Property(5단계) 둘로 확정	<code>README.md</code> 5절

날짜	결정	근거
2026-08-17	capability 테이블 45종 확정, PageNavigation 4종만 미정 남김	spec/capability-table.md
2026-08-22	D-5 달함. I.I.S. 본편 재개를 4단계 착수 조건으로 두지 않기로 결정	README.md 5절, plan/poc-module-map.md
2026-08-28	노드가 여럿인 질의의 복합 구조 fallback 순서 확정(N1~N5). I-12 달함, B-31·B-32·D-8·I-11 신규	spec/capability-table.md 3.6절, 2절과 6절 부록 7.5

아직 폐기 여부만 확정되고 새 날짜가 안 나온 것. 2단계 배포 마감(8/18)은 폐기됐지만 재조정된 새 마감은 원문에 없다.

가장 급한 병목 셋 (2026-08-22 기준)

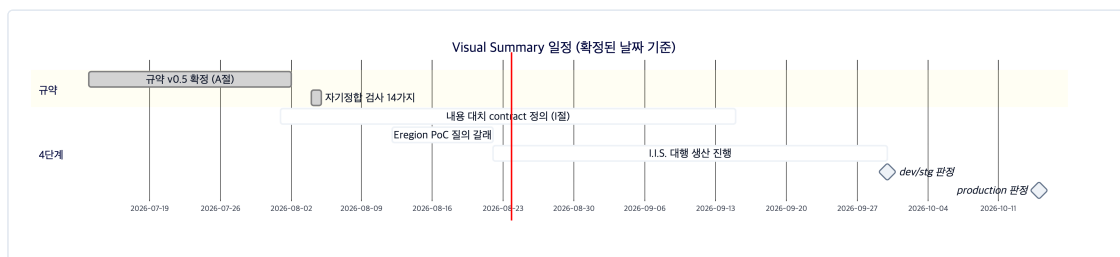
1. **평가 pool과 사전 적재(I-8)**. comp-sign 시그니처, I.I.S. 경계, 5단계 속성을 같은 컴포넌트 부분집합에서 비교하려면 이 작업이 먼저 끝나야 한다. PoC 의 comp-sign 은 아직 서버 메모리에서 즉석으로 만드는 방식이라, 파이프라인이 실제로 쓰는 `comp_signatures` 테이블 적재와 PoC 조회 경로를 맞추는 일이 남아 있다.
2. **T-8 속성 투영(property projection, I-7)과 5단계 대치 PoC(I-9)**. 속성 추출(T-7)만으로는 5단계 화면이 열리지 않는다.
3. **내용 대치 contract(I-2)**. 나쁜 결과의 85% 가 여기서 나온다는 실측(4절 참고)이 있고, 이석님과 류민님 협의가 필요하다. I-8 과 병행 가능하다.

5단계 로드맵 지도의 T-번호 작업 상태

docs/projects/mordor/visual-summary/plan/node-tag-and-property-poc-blueprint.md 가 정의한 작업 단위다.

작업	무엇을 하나	상태
T-4	I.I.S. 경계(Semantic Component Boundary) 추출	코드 있음(<code>extractBoundary.ts</code>), 사람 검수는 아직
T-5	노드 단위 태그로 검색	코드 있음(<code>nodeSearch.ts</code>), 메인 파이프라인 통합은 아직
T-6	노드 단위 태그로 대치	코드 있음(<code>nodeBinding.ts</code>), 평가셋 전량 실행은 아직
T-7	카테고리 고유 속성 추출	v1 코드 있음(<code>extractProperty.ts</code>), 정확도 검수 미완
T-8	property projection(이중 카테고리 간 속성 슬롯 매핑)	미구현

시간순으로 본 일정



2026-07-13 ~ 2026-08-02	규약 확정 (A절)	완료
2026-08-01 ~ 2026-09-15	4단계 내용 대치 contract 정의 (I절)	진행 중
2026-08-12	4단계 PoC: Eregion 질의 갈래 동작	진행 중
2026-08-22	4단계 I.I.S.: T-4/T-5/T-6/T-7 진행	진행 중
2026-09-30	4단계 dev/stg 판정	목표 마일스톤
2026-10-15	4단계 production 판정	목표 마일스톤

위험과 감수하는 것

가장 크게 감수하는 위험은 4단계에서 생성기 교체, node 태그 도입, 검색 로직 변경이 동시에 일어난다는 점이다. 품질이 떨어졌을 때 원인이 이 셋 중 무엇인지 가리기 어렵다. 이를 보완하기 위해 단계별로 계측 지점을 따로 설계하기로 했다(TODO C-3). 또한 데이터 쪽에서는 semantic tag 색인 분포가 List 한 종류에 81.1%(design object 83,771건 기준, README.md 3절 실측) 쏠려 있어, 검색 풀 자체가 얇은 카테고리에서는 fallback 이 자주 발생할 수밖에 없는 구조적 제약이 있다. RAT pod 소관 I.I.S. 본편이 2026년 6월 중순부터 정지된 상태라는 것도 계속 감시해야 할 리스크다. 다만 2026-08-22 결정으로 이 정지가 4단계 진행을 막지는 않는다.

구현물 로직: PoC 파이프라인과 코드 구조

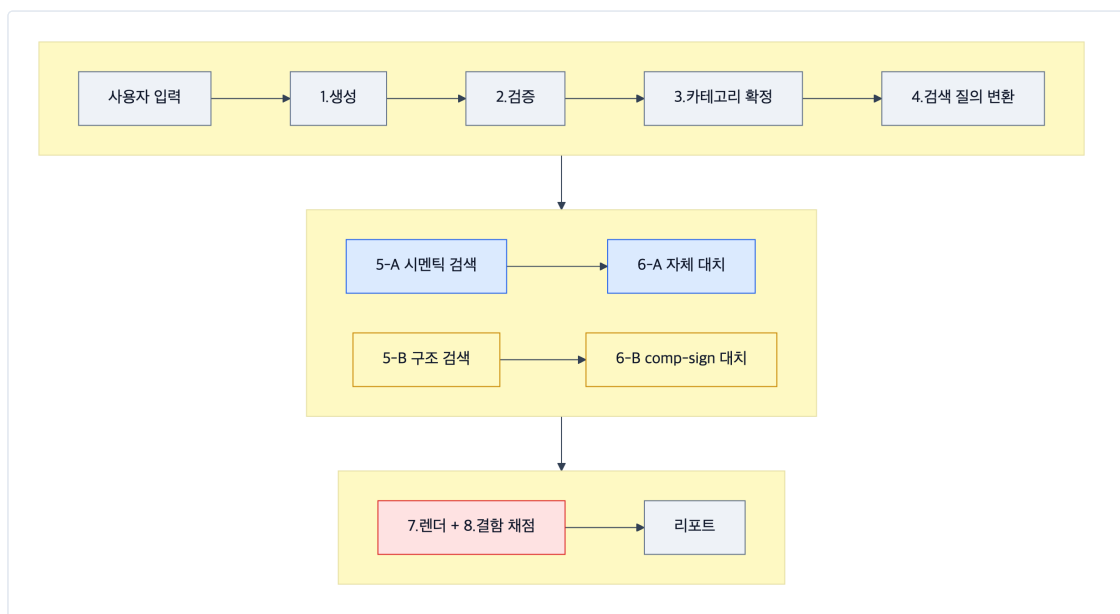
왜 PoC 를 만들었는가

2026-07-26 조사에서, 대치 결과가 나쁘다고 판정된 것 200건을 직접 읽고 자동 채점 16,819행을 함께 집계한 결과, 나쁜 결과의 85% 가 그림이 잘못 그려져서가 아니라 "고른 컴포넌트와 넣으려는 내용이 안 맞아서" 였다. 세부 원인은 넷이다. 글자가 자리에 안 들어가 넘치는 것, 제목에 들어갈 내용이 설명 자리에 들어가는 식으로 위계가 어긋나는 것, 항목 수와 칸 수가 안 맞는 것, 대치가 안 돼 템플릿 원문이 그대로 남는 것이다.

이 구간에는 구현이 없는 것이 아니라 **규약이 없었다**. 이석님의 `packages/comp-sign` 패키지가 시그니처 추출, 검색, 랭킹, 내용 대치 네 단계를 이미 구현하고 있다. 비어 있던 것은 "고른 다음에 무엇을 어디에 어떻게 넣을 것인가"의 규약이다. 그래서 PoC 의 목적은 데모가 아니라 **스쿼드가 못 닫고 있는 결정을 실측으로 닫는 것이다**. 화면이 있는 이유도 보여 주기 위해서가 아니라 판정 이력을 사람이 열어 볼 수 있게 하기 위해서다.

전체 흐름 (여덟 단계)

사용자가 쓴 글 한 덩어리가 들어가서, 슬라이드에 그려질 결과물과 결함 채점표가 나오기까지 여덟 단계를 거친다.



사용자 입력

- 1. Visual Summary 생성 (LLM 이 YAML 을 쓴다)
- 2. 검증 (파싱 → id 부여 → 스키마 검사)
- 3. 카테고리 확정 (성립 조건 판정 + fallback)
- 4. 검색 질의 변환 (structure → 태그와 제약)
- 5-A. 시멘틱 검색 (semantic tag 로 후보 탐색)
- 5-B. 구조 검색 (구조 해시로 후보 탐색)
- 6-A. 자체 대치 (RLSC 슬롯에 직접 삽입)
- 6-B. comp-sign 대치 (comp-sign 이 삽입)
- 7. 실서비스 렌더러로 렌더 (브라우저에서 실제 렌더러로 그림)
- 8. 결함 채점 (같은 기준으로 두 결과를 비교)

1. 생성. 사용자 글을 LLM 에 넣어 YAML 로 구조화한다. 지금 쓰는 모델은 `gpt-5.4-mini` 이고, 스키마를 어기면 오류 메시지를 되먹여 재생성하는 루프가 있다(최대 3회).

- 2. 검증.** YAML 을 파싱하고 각 항목에 배열 순서대로 식별자를 붙인 뒤 스키마로 검사한다. "파싱은 됐는데 값의 타입이 조용히 바뀌는" silent corruption 을 따로 세는 계측이 붙어 있다.
- 3. 카테고리 확정.** 생성기가 지목한 카테고리가 성립 조건을 못 채우면 후보에서 빠지고, 정해진 fallback 순서를 따라 내려간다. 마지막 fallback 후보 셋(List, LongFormTextLayout, HighlightedMessage)은 개수 조건만 보므로 언제나 하나는 성립한다.
- 4. 검색 질의 변환.** 확정된 카테고리나 구조를 검색이 쓸 수 있는 태그와 제약(항목 수, 글자 용량 등)으로 바꾼다.
- 5~6. 검색과 대치.** 여기서 두 갈래로 갈린다. 다음 절에서 자세히 다룬다.
- 7. 렌더.** 대치된 결과를 이미지 합성 서버를 왕복하지 않고, 합성 서버가 쓰는 것과 같은 페이지 렌더러를 브라우저에 직접 띄워 그린다.
- 8. 결함 채점.** 두 갈래의 결과물을 같은 기준으로 재서 결함을 센다.

실서비스 코드와 PoC 코드의 경계

"PoC" 라는 말이 "전부 임시로 만든 것"으로 읽히면 곤란하므로 경계를 분명히 적는다.

구간	상태	설명
Visual Summary 생성	실서비스 코드	실제 OpenAI 호출과 프롬프트, 재생성 루프가 그대로 돈다
검증, 카테고리 확정, 질의 변환	실서비스 코드	규약을 코드로 옮긴 것. 문서와의 정합을 테스트가 기계로 대조한다
시그니처 색인	계산은 실서비스 코드, 저장은 PoC 코드	실제 <code>createCompSignatureBuilder()</code> 로 만들지만, 실서비스는 미리 테이블에 적재하는 반면 PoC 는 서버가 뜰 때 메모리에 즉석으로 만든다
후보 조회	PoC 코드	실서비스는 백엔드가 데이터베이스에서 조회하는데, PoC 는 메모리 안에서 해시를 맞춰 본다
랭킹과 대치 (구조 검색 쪽)	실서비스 코드	이석님 패키지의 <code>rankCompCandidates</code> , <code>buildContentReplacement</code> 를 그대로 호출한다. 내용시각화 BE 가 쓰는 4단계 경로다
랭킹과 대치 (시멘틱 검색 쪽)	PoC 자체 구현	검색은 <code>search.ts</code> , 대치는 <code>binding.ts</code> (규약 RLSC 방식). comp-sign 이나 BE 경로가 아니다
렌더	실서비스 코드	합성 서버가 쓰는 것과 같은 페이지 렌더러
결함 채점	자체 구현	실서비스에 대응물이 없는, 이 PoC 가 새로 만든 계측기

색인 규모는 로컬에 복사해 둔 태깅 컴포넌트 2,468건이다. 여기서 시그니처 3,424벌이 나오고 고유 구조 해시가 668개이며, 빌드에 33초쯤 걸리고 30분 캐시한다. 실서비스 규모는 178,095건이라 이 방식을 그대로 쓸 수 없고, 이것이 아직 스쿼드가 정해야 할 열린 결정이다(아래 참고).

범위 밖으로 둔 것은 차트와 이미지 데이터 대치, 색상 대치, 실제 테이블 적재, 백엔드 검색 실행이다.

왜 검색 경로를 두 갈래로 나란히 두었는가

같은 입력을 두 가지 방식으로 검색하고 대치해서 나란히 비교한다. 어느 쪽을 실서비스에 넣을지는 말로 정할 문제가 아니라 실측으로 정할 문제라고 보기 때문이다.

시멘틱 검색(자체 구현 경로) 은 `design_object_semantic_tags` 에 붙은 의미 태그로 후보를 찾는다. 태그가 맞는 후보를 먼저 채우고, 그 안에서 항목 수와 글자 용량이 얼마나 맞는지로 점수를 매긴다.

구조 검색(comp-sign 실제 코드 경로) 은 의미 태그를 보지 않는다. 사용자 내용의 구조를 markdown 으로 바꿔 comp-sign 에 넣으면 구조 해시와 변형들이 나오고, 색인에서 해시가 정확히 일치하는 컴포넌트를 후보로 삼아 comp-sign 의 채점 항목 20여 종(면적 비율, 줄 수 초과, 스타일 불일치 등)으로 순위를 매긴다.

두 축이 깨끗하게 갈려 있다는 것이 뜻밖의 자산이 됐다. 스쿼드가 합의한 판정 지표가 "시멘틱 검색과 구조 검색의 교집합 상위 10건 품질"인데, 이 PoC 에 그 두 검색이 이미 각각 실서비스 코드로 있어서 교집합을 실제로 계산할 수 있다.

두 경로를 잇는 다리. comp-sign 의 입력은 markdown 인데 Visual Summary 는 항목 트리를 내놓는다. 그 사이를 잇는 것이 `structure` → `markdown` 변환이다. 트리를 직접 조립하지 않고 markdown 을 거치는 이유는, comp-sign 이 트리와 해시를 같은 파싱 과정에서 만들기 때문이다. 트리를 직접 만들면 해시 경로와 어긋날 수 있다고 패키지 소스가 명시한다.

결과물 품질을 재는 방법

이전 라운드에서는 대치 결과 23장을 렌더해 손으로 보고 판정했다. 결함을 실제로 찾아냈지만 코드를 고칠 때마다 23장을 다시 볼 수 없어 "이 변경이 좋아졌나"를 말할 수 없었다. 그래서 같은 판정 기준을 기계로 옮겼다.

결함 16종을 정의했고, 각각은 나쁜 결과의 원인 넷 중 하나에 대응한다.

원인	결함 종류	무슨 일이 벌어진 것인가
텍스트 용량 초과	<code>capacity.lineOverflow</code>	넣은 글이 그 자리의 줄 수를 넘는다
	<code>capacity.charOverflow</code>	줄 수를 모르는 자리에서 글자 수가 용량을 넘는다
매핑과 위계 오류	<code>mapping.categoryDrift</code>	요청한 카테고리에서 규약이 허용하지 않는 카테고리의 컴포넌트를 골랐다
	<code>mapping.roleInversion</code>	세부 내용이 제목 자리로 올라가거나 페이지 제목이 설명 자리로 내려간다
	<code>mapping.markerAbuse</code>	순서 번호가 들어갈 자리에 문장이 들어간다
	<code>mapping.orderInversion</code>	화면에 놓인 순서가 입력 순서와 뒤집힌다
슬롯 수 불일치	<code>slots.unitDeficit</code>	항목이 칸보다 많다
	<code>slots.unitSurplus</code>	칸이 항목보다 많아 빈 칸이 남는다
	<code>slots.lineDeficit</code>	여러 항목이 한 자리에 접히거나 아예 버려진다
	<code>slots.lineSurplus</code>	채우지 못한 텍스트 자리가 남는다
대치 실패	<code>replace.residue</code>	결과에 남은 글이 사용자 입력에서 온 것이 아니다
	<code>replace.placeholder</code>	comp-sign 이 임시 문구로 채웠다
	<code>replace.emptyed</code>	채울 내용이 없어 템플릿 원문을 비웠다
	<code>replace.markupLeak</code>	markdown 표기가 결과 글에 그대로 나갔다
	<code>replace.unmapped</code>	대치할 텍스트를 실제 요소에 못 붙였다
	<code>replace.contentLoss</code>	사용자가 넣은 조각이 결과 어디에도 없다

심각도는 셋이다. `blocker` (가중치 5, 사용자 내용이 사라졌거나 남의 내용이 남은 것), `major` (가중치 2, 위계가 어긋난 것), `minor` (가중치 1, 자리가 비거나 임시 문구가 들어간 것)다. 결함 수를 그냥 더하면 가벼운 결함이 많은 결과가 더 나빠 보이므로, 가중치를 곱해 더한 값을 한 건의 점수로 쓴다.

설계 원칙은 하나다. 런타임 랭킹이 최적화하는 바로 그 양을 쟀다. 넘침 판정을 comp-sign 의 채점과 같은 계산으로 맞췄다(글자 수가 아니라 줄 수 기준, 올림(조각 길이 ÷ 줄당 글자 수) 를 더해 비교). 자리마다의 용량(줄 수, 줄당 글자 수, 역할 이름)도 comp-

sign 시그니처가 담은 값을 두 경로 모두에서 그대로 쓴다. 이게 성립하려면 RLSC 노드 식별자와 sheet 요소 식별자가 같아야 하는데, 색인 2,468건의 텍스트 자리 17,899개 전량에서 확인했고 결합률이 100% 였다.

기계 채점을 그대로 믿지 않고 실제 텍스트를 눈으로 확인해 잘못 잡은 것 셋을 건어냈다. 순서 번호를 "입력에서 오지 않은 글"로 잘못 세던 것, 정상 배치(페이지 제목이 제목 자리로, 항목 이름이 설명 자리로 가는 것)를 위계 어긋남으로 잘못 세던 것, 잔존한 템플릿 원문에 markdown 기호가 있으면 한 결합을 두 번 세던 것이다. 셋 다 회귀 테스트로 고정했다.

결합을 생성, 검색, 대치 중 누구 탓으로 돌리는가

귀속은 단일 결과가 아니라 대안과의 비교에서만 나온다. 두 경로 다 상위 후보 전부에 대해 대치를 돌려 보고 그중 하나만 쓰는데, 채점기가 그 버려지던 것을 주워 담아 다음 세 갈래로 나눈다.

후보 풀 안 어느 것으로도 못 피한 결합	→ 생성 탓
같은 경로의 다른 후보로는 피할 수 있던 것	→ 검색 탓
같은 후보를 다른 경로가 더 잘 채운 것	→ 대치 탓

여기에 정직성 장치가 하나 붙는다. 랭킹이 애초에 볼 수 없었던 결합(시그니처 밖 자리의 넘침 등)은 검색 탓에서 떨어낸다. 랭킹에게 없는 정보로 벌점을 매기면 이 수치로 가중치를 튜닝할 때 잘못된 방향으로 가기 때문이다.

지금까지 나온 기준선 수치 (2026-08-15 갱신, 입력 50건, 컴포넌트 65개, 상위 10건 기준)

지표	시멘틱 검색	구조 검색	순위 융합
심각도 가중 평균 (낮을수록 좋음)	20.0	22.0	20.7
1위 후보가 결합 0 인 비율	0.0%	1.6%	1.5%
무거운 결합(blocker)이 없는 입력 비율	8%	7%	원문에 없음

일여야 할 것 셋. 첫째, 의미 적합성을 재기 시작하자 두 경로의 격차가 사라졌다. 2026-08-15 이전에는 구조 검색(17.5)이 시멘틱 검색(19.9)을 크게 이겼는데, "검색이 허용한 카테고리 밖에서 고른 것"을 결합으로 새로 세자 순위가 뒤집혔다(시멘틱 20.0, 구조 22.0). 구조 검색이 앞서 보이던 것의 대부분은 의미를 재지 않은 덕이었다.

둘째, 텍스트 넘침이 양쪽 모두 1위 결합이고 건수가 거의 같다(시멘틱 176건, 구조 177건). 경로를 바꿔도 줄지 않는다는 것은 대치 방식이 아니라 검색 풀과 용량 자체의 문제라는 뜻이다.

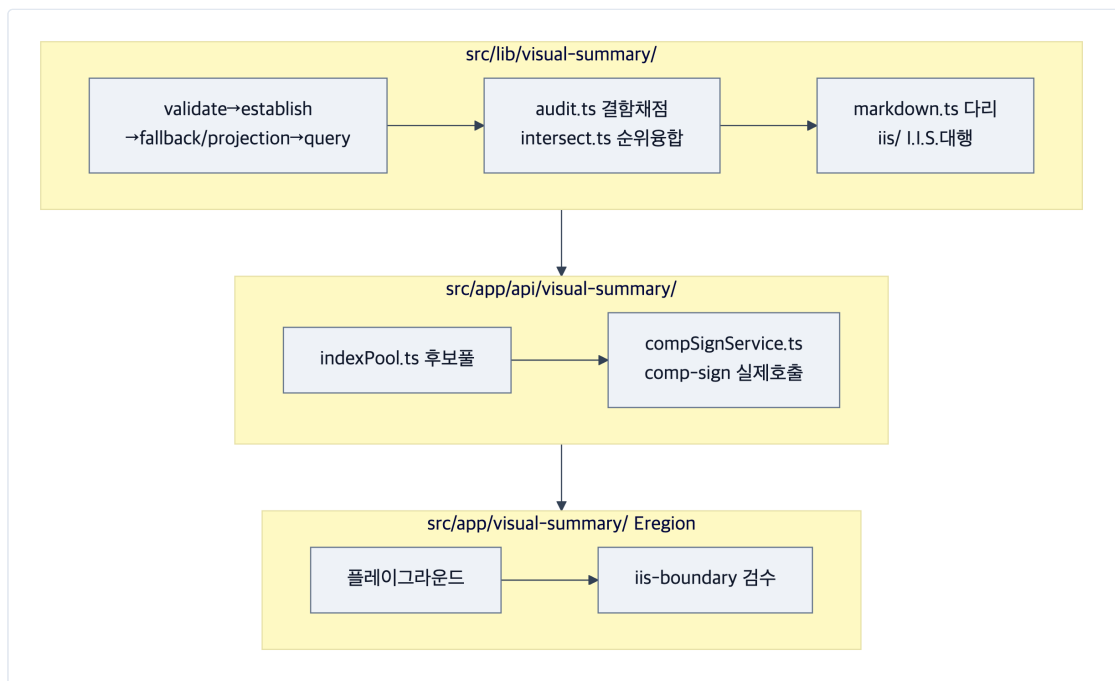
셋째, 결합의 대부분이 검색 탓이다. 귀속을 세면 생성 36, 검색 466, 대치 17, 랭킹이 못 본 것 74 다. 랭킹 가중치 실험(MOR-1936)이 대치 알고리즘을 다듬는 것보다 값이 큰 다음 수라는 근거가 여기서 나온다. 또한 두 검색은 1위 후보가 65개 중 2개(3%)만 겹쳐서, "교집합 상위 10건"을 그대로 지표로 쓰면 표본이 거의 남지 않는다.

정책 결정 중 눈여겨볼 것

두 검색 경로가 처음에는 서로 다른 조건으로 색인을 읽고 있어서(한쪽은 컴포넌트 구조가 없으면 페이지 전체 구조로 대신, 다른 쪽은 sheet 와 구조가 둘 다 있는 것만) 비교가 불공정했다. "sheet 와 컴포넌트 구조가 둘 다 있음"이라는 엄격한 교집합으로 통일해 해결했다.

채점은 무료이고 결정적이어야 한다는 원칙에 따라 생성과 채점을 두 단계로 분리했다. 생성 결과를 저장소에 스냅샷으로 커밋해 두고, 채점은 LLM 을 한 번도 부르지 않고 그 스냅샷 위에서 돈다. 스냅샷 캐시 키에는 프롬프트 본문의 해시가 들어가서, 프롬프트를 고치면 스냅샷이 자동으로 낡은 것이 되고 도구가 멈춘다.

비교 조건(프롬프트, 모델, 색인 풀, 채점 임계값, comp-sign 버전, 평가셋)이 하나라도 다르면 도구가 비교 자체를 거부한다. 이 판단이 실제로 값을 했다. 처음에는 평가셋을 지문에 안 넣어서, 10건짜리 실행과 50건짜리 실행을 비교했더니 통과했고 모든 결함이 100건씩 늘어난 것처럼 보인 적이 있었다.



comp-sign v4.3.8 이 실제로 하는 일 넷

실서비스의 검색과 대치는 이석님의 `miridih/miricanvas-ivi packages/comp-sign` 이 한다. 이 절이 그 실물을 정리한다. 이 구간이 비어 있다고 적힌 문서가 한때 있었는데 사실이 아니다. 비어 있는 것은 구현이 아니라 규약이다. 패키지 README 가 스스로를 "컴포넌트 매칭 파이프라인 전체"로 정의하고, 네 함수로 되어 있다.

#	함수	하는 일
1	<code>buildCompSignatures</code>	컴포넌트를 구조 해시와 <code>text_map</code> , <code>text_node_roles</code> , 채점 feature 로 바꾼다. DB 적재용이다
2	<code>buildCompSearchData</code>	markdown 을 요약 트리(<code>textTree</code>)와 조회용 <code>{hash, variant, demand}[]</code> 로 바꾼다
3	<code>rankCompCandidates</code>	후보를 채점하고 중복을 제거하고 정렬한다
4	<code>buildContentReplacement</code>	대치 텍스트를 만들고 box 를 보정하고 형제 그룹을 만든다

DB 접근과 후보 fetch 는 호출자(BE) 몫이고 이 패키지는 순수 계산만 한다. **입력이 markdown 이라는 점이 중요하다**. Visual Summary YAML 이 이 경로에 직접 들어갈 자리가 아직 없고, 그것을 잇는 것이 3단계 질의 변환기가 하는 일이다.

태그 군 셋이 랭킹을 정한다

"v7 태그를 후보 필터로 쓸지 순위 가산점으로 쓸지"가 열린 결정(D-1)이었는데, **답은 둘 다 아니고 태그 군(tier)이다**. 구현에 이미 들어 있다.

군은 컴포넌트의 성질이 아니라 **그 컴포넌트를 후보로 끌어온 근거의 출처**를 나눈다. 같은 컴포넌트도 검색 조건이 달라지면 다른 군에 들어간다.

군	그 컴포넌트가 후보에 들어온 이유	근거의 출처
0군	사용자가 직접 고른 분류에 속한 태그를 갖고 있다	검색 조건 UI 의 분류와 태그 선택
1군	LLM 이 사용자의 글을 보고 추천한 태그를 갖고 있다. 0군과 겹치는 태그는 뺀다	실사용에서는 LLM 이 글과 함께 내려준다
2군	0군에도 1군에도 해당하는 태그가 하나도 없다	태그와 무관하게 구조 해시만으로 걸린 후보

0군과 1군의 차이는 **사용자가 명시한 의도인가, 기계가 추천한 의도인가** 다. 명시한 쪽을 항상 앞에 둔다.

후보를 채울 때는 군 순서로 채우고 목표치(기본 50)를 넘으면 뒤 군을 조회하지 않는다. 한 군 안에서 여러 태그가 자리를 나눌 때는 max-min fair 분배를 쓴다. 태그마다 $1/M$ 씩 균등하게 가져가되, 그만큼 채우지 못하는 태그의 남은 자리를 여유 있는 태그끼리 다시 나눈다.

정렬할 때 군 순서가 점수보다 절대적으로 우선한다. 1군 후보가 아무리 점수가 높아도 0군 후보를 앞설 수 없고, 같은 군 안에서만 점수로 정렬한다. 그래서 태그 일치에 주는 가점 같은 것이 없다. 0군이 12개뿐이면 1군으로 38개를 더 채우는데, 최종 정렬에서 0군 12개가 점수와 무관하게 1군 38개보다 앞에 놓인다.

규약과 어긋나는 지점이 하나 있다. 규약 5절은 "성립 조건을 못 채우면 후보에서 빠진다"는 이분법인데 구현은 제외하지 않고 뒤 군으로 미룬다. 규약은 컴포넌트가 무엇을 소화하는지를 보고 구현은 그 후보가 어디서 왔는지를 본다. 축이 다르므로 맞춰야 한다(1-5).

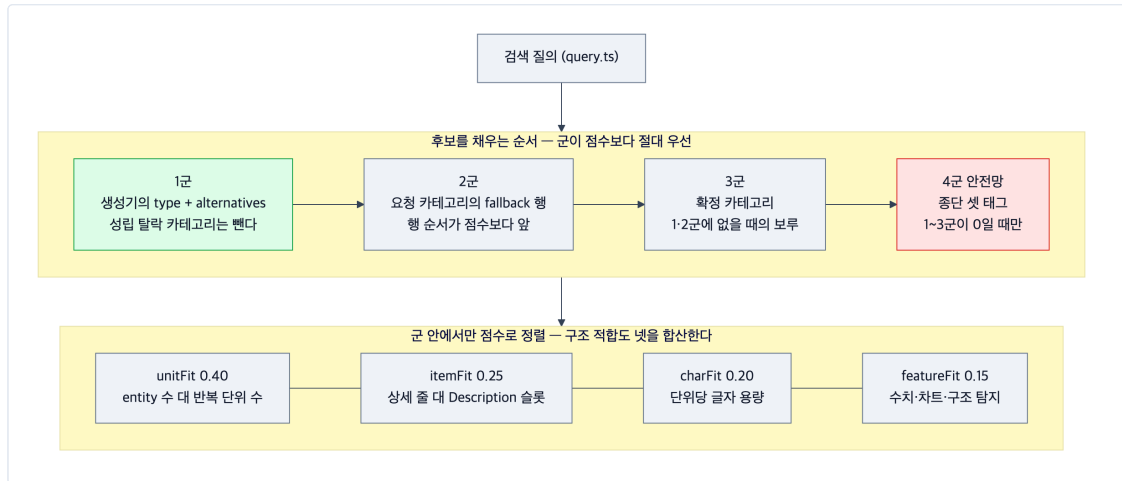
태그 값의 형식도 코드가 SSOT 다. `Enumeration_List` 처럼 `{major}_{minor}` 이고 DB 에 그대로 들어 있다. minor 이름만 담던 옛 어휘(v2.0)는 버렸다. 두 형식을 섞으면 조회가 조용히 0건이 된다. 분류 규모도 코드 쪽이 정확해서 major 23, minor 49 다.

검색 fallback 군 넷: 후보가 어디서 오나

앞 절의 군 셋은 **comp-sign(실서비스)** 이 쓰는 것이고 나누는 축이 "그 후보를 끌어온 근거가 사용자냐 LLM 이냐"다. **PoC 의 실색인 검색(`search.ts`)** 이 쓰는 군 넷은 다른 축이다. 나누는 축이 "그 태그가 카테고리 강등 사다리의 어느 지점에서 나온 것이냐"다. 이름이 같아서 섞이기 쉬우니 항상 어느 쪽 군인지 밝혀 적는다.

검색 군	질의가 그 태그를 어디서 얻나	무엇을 뜻하나
1군	생성기가 적은 <code>type</code> 과 <code>alternatives</code>	사용자 내용을 읽은 쪽이 이걸로 그리자고 했다
2군	요청 카테고리의 fallback 행	요청한 것이 색인에 없을 때 규약이 정한 물러날 곳
3군	강등 사다리가 최종 확정된 카테고리	구조를 실제로 그릴 수 있다고 판정된 것
4군	마지막 fallback 셋	안전망. 1~3군이 하나도 못 찾았을 때만 발동한다

4군의 태그는 `Enumeration.List`, `TextBasedCommunication.LongFormTextLayout`, `TextBasedCommunication.HighlightedMessage` 셋이다.



군이 점수보다 절대 우선한다. 2군 후보의 구조 적합 점수가 아무리 높아도 1군 후보를 앞서지 못한다. 군 안에서도 **태그 순위가 점수보다 앞선다.** 2군은 fallback 행의 순서 그대로라 `Timeline` 이 `List` 보다 항상 먼저다. 이 규칙은 원래 없었고, 실사용 로그 10건 실행에서 "같은 군 안에서 순서가 점수에 밀리는" 문제를 발견해 고정한 것이다.

한 단계씩 따라간다. 앞 절(2절)의 Gauge 예시가 검색 질의로 바뀌는 모습이다. 요청은 `KPIKeyMetric.Gauge`, 구조는 entities 3개에 각각 `value` 가 있고 `dataset` 은 없다.

검색 군	이 질의에서 실제 값	왜 그렇게 되나
1군	비어 있다	<code>Gauge</code> 는 <code>measure</code> 가 없어 성립이 탈락했다. 성립이 탈락한 카테고리는 1군에서 뺀다. 구조가 못그리는 카테고리를 최우선으로 검색하면 안 되기 때문이다. <code>alternatives</code> 가 적혀 있었다면 그것이 1군을 채웠을 것이다
2군	<code>BigNumber</code> , <code>PictogramChart</code> , <code>BarChart</code> , <code>HighlightedMessage</code>	<code>Gauge</code> 행의 fallback 순서 그대로다. 이 순서가 점수보다 앞선다
3군	따로 발동하지 않는다	확정 카테고리인 <code>BigNumber</code> 가 이미 2군에 들어 있기 때문이다. 군 판정은 태그마다 가장 낮은 군 하나만 매긴다
4군	발동 안 함	1:2군에서 후보가 나왔기 때문이다

그러면 3군은 언제 쓰나. 확정 카테고리가 1군에도 2군에도 없을 때의 보루다. 요청 `type` 이 성립 탈락해 1군에서 빠졌는데 확정된 곳이 그 요청의 fallback 행에도 없는 경우가 그것이고, 위 표의 1군 행에 적은 계약("성립 탈락한 요청 카테고리를 1군에 되살리지 않는다")이 성립하려면 이 자리가 반드시 있어야 한다. 2군과 역할이 다르다. 2군은 "물러날 후보들"이고 3군은 "그릴 수 있다고 판정된 하나"다.

1군이 비는 것이 결함이 아니다. 다만 여기에 구제 조항이 하나 있다. 생성기가 `type` 을 아예 안 적었으면 1군이 처음부터 비는데, 그때는 확정 카테고리를 1군에 넣어 준다. 이 분기는 2026-08-29 `type` 필수 개정으로 규약상 도달할 수 없는 경로가 됐다. 스키마 검증이 먼저 막기 때문이다. 코드에는 아직 남아 있고, 뺄지는 개정 반영 작업에서 함께 본다. 반대로 "확정 카테고리를 항상 1군에 넣는" 방식으로 바꾸면 안 된다. 그렇게 고쳐 봤더니 위 표의 첫 행 계약(성립 탈락한 요청 카테고리는 1군에서 빠진다)이 깨졌다. 그 자리는 3군이 맡는다.

4군은 발동했을 때만 질의에 적힌다. 안전망 태그를 늘 질의에 넣어 두면 "쓰지도 않은 안전망"까지 허용 태그가 되어 판정이 무뎀니다. 그래서 랭킹 결과에서 4군 후보가 실제로 나왔을 때만 그 태그를 질의 태그 집합에 되돌려 적는다. 이 안전망 자체가 실사용 10건 실행에서 발견된 구멍의 대응이다. **마지막 fallback 카테고리조차 색인에 없으면 후보가 0이 되던 문제였다.**

1군 판정에는 함정이 하나 있었다. 1군을 채울 때 커널 어휘 41종 상수로 카테고리 여부를 보면 확장 4종(`PageNavigation.*`)이 1군에서 빠진다. 그런데 위의 구제 조항 때문에 `alternatives` 가 없는 컴포넌트는 무사해서 오래 안 보였다. `alternatives` 에 커널 종이 하나라도 있으면 그것이 1군을 채워 구제가 발동하지 않고 **확정 카테고리가 질의에서 통째로 사라진다**. `resolvedId` 는 멀쩡히 `PageNavigation_Opening` 인데 검색은 그 태그를 묻지 않으므로 지표에 흔적이 남지 않는다. 골든셋 실측에서 `PageNavigation` 40행 중 13행이 이 모양이었다(2026-08-26 코드리뷰). 지금은 주입된 45종 사전으로 판별한다.

군 안에서의 채점은 구조 적합도 넷이다. 대치가 하게 될 일을 미리 셈하는 것이라, 태그가 맞는지는 이미 군이 봤고 여기서는 "그 후보에 이 내용이 들어가는가"만 본다.

항목	가중치	무엇을 보나
<code>unitFit</code>	0.40	entity 수 대 RLSC 반복 단위 수. 단위가 모자라면(접힘 = 내용 손실) 남는 것(빈 단위)보다 크게 감점한다
<code>itemFit</code>	0.25	entity 당 상세 줄 수 대 단위당 Description 슬롯 수. 상세가 버려질 자리 부족을 잡는다
<code>charFit</code>	0.20	단위당 글자 용량. 총량이 아니라 단위 기준이라 단위 수가 달라도 왜곡이 없다
<code>featureFit</code>	0.15	수치가 있으면 Subtitle 자리, <code>matrix</code> 면 차트 노드, 구조 탐지 성공 여부

규약과 구현이 어긋나는 지점은 여기에도 남아 있다. 규약 5절은 "성립 조건을 못 채우면 후보에서 빠진다"는 이분법인데, `comp-sign` 은 제외하지 않고 뒤 군으로 미룬다. PoC 는 1군에서만 빼고 2군과 3군으로 내려보내므로 규약 쪽에 가깝지만 완전히 같지는 않다. 축을 맞추는 것이 1-5 다.

대치 입력 용어 세 총

같은 "트리"라도 총이 셋이다. 문서와 코드에서 이름을 섞어 쓰면 fallback 이후 대치 계층을 따라갈 수 없다.

이름	무엇인가	만드는 곳과 쓰는 곳
Visual Summary <code>structure</code>	YAML 의 <code>entities</code> , <code>relations</code> , <code>dataset</code> . 검색 질의와 <code>structure projection(P1~P8)</code> 의 입력	생성기, <code>validate.ts</code> , <code>projection.ts</code>
<code>textTree</code>	<code>buildCompSearchData</code> 가 <code>markdown</code> 을 파싱해 만든 대치용 텍스트 트리 와 변형	<code>compSignService.ts</code> , <code>buildContentReplacement</code> 입력
<code>text_map</code> , <code>text_node_roles</code>	컴포넌트 시그니처의 슬롯 용량과 역할 (<code>chars</code> , <code>lines</code> , <code>role</code>). 런타임 용량의 SSoT	<code>buildCompSignatures</code> 적재, 채점과 대치
RLSC text slots	PoC 자체 대치가 RLSC 서브트리의 Title, Description, Marker 슬롯에 직접 넣는 경로	<code>binding.ts</code> , <code>rlsc.ts</code>

다리는 `structure` → `markdown` → `buildCompSearchData` → `textTree` 다. **comp-sign** 경로는 Visual Summary YAML 을 직접 받지 않는다.

fallback 이후 대치 계층 다섯

카테고리 fallback 으로 어떤 컴포넌트 종류를 고를지 정한 뒤, 무엇을 어디에 넣을지는 아래 계층을 따른다.

계층	무엇을 하나	코드	상태
복합 구조 순서 (N1~N5)	노드가 여럿일 때 어느 노드부터 포기하나	없다	규약 확정(2026-08-28). 코드 미착수(I-11)
exact structure match	해시와 태그로 동일 구조 후보를 찾는다	comp-sign buildCompSearchData , PoC search.ts	동작한다. 풀이 알으면 실패가 잦다
semantic fallback	카테고리 강등과 P1~P8	fallback.ts , projection.ts	규약과 코드 모두 완료
4단계 comp-sign 대치	textTree 와 시그니처로 텍스트를 넣는다	compSignService.ts , comp-sign 패키지	실서비스 경로. 규약 contract 미완
5단계 property 대치	이중 type 사이 카테고리 고유 속성 투영(좁은 뜻)	T-7 추출만 있고 T-8 미착수	부분

PoC 규약 RLSC 대치(binding.ts)는 실서비스 경로가 아니다. Visual Summary structure 를 RLSC 슬롯 규칙으로 직접 넣는 MORDOR PoC 전용 기준선이고, 내용시각화 BE 가 쓰는 대치는 comp-sign buildContentReplacement 하나다. 두 경로를 같은 후보 풀과 같은 용량 근거로 재야 비교가 성립한다.

structure projection 과 property projection 은 이름만 비슷하고 하는 일이 다르다. 앞은 내용(질의 쪽 structure)을 옮기고 뒤는 카테고리 고유 속성(질의 쪽 params , 색인 쪽 properties)을 옮긴다. 여기서 property 는 좁은 뜻이다. 5단계 국면 이름 I.I.S. (Semantic Component Property) 의 'Property'(속성과 구조 대응 둘을 합친 넓은 뜻)와 구분한다. 1절 용어 사전의 표기 규칙을 따른다. P1~P8 만으로는 "List 안의 BigNumber" 처럼 서로 다른 카테고리 사이에서 속성을 옮기는 문제를 풀지 못하고, 그것이 5단계와 T-8 의 영역이다.

평가 pool 을 고정하는 규칙 다섯

전 컴포넌트(로컬 수만 건, 실서비스 17만여 건)에 comp-sign 시그니처와 I.I.S. 속성을 한꺼번에 구울 수는 없다. 세 대치 경로를 공정하게 비교하려면 같은 후보 풀 위에 미리 구워 뒀야 한다.

#	규칙
1	기준 pool 은 indexPool.ts 슬어와 같다. sheet_json 과 structure_json 이 둘 다 있는 것이고, 확장 색인과 같은 집합을 쓴다
2	평가 pool 은 그중 고정 subset 이다. 실사용 50건(컴포넌트 65개)이 검색과 대치의 입력이다
3	후보 pool 크기는 약 5,000건으로 둔다 (2026-08-23 정원석 제안). 지금 로컬 색인이 2,468건이고 실서비스가 178,095건이다. 5,000건이면 카테고리별 후보가 마르지 않아 정책 비교가 pool 크기의 성질에 휘둘리지 않고, 굵은 비용이 하루 안에 끝난다. 아직 실행으로 확인한 값이 아니라 제안이다
4	comp-sign 시그니처, I.I.S. 경계, I.I.S. 속성을 같은 design_object_id 집합에 굶는다. 한쪽만 있으면 5단계 경로 비교가 성립하지 않는다
5	5단계 화면은 속성 bake 와 T-8 규칙이 있어야 연동된다. 지금은 4단계 comp-sign 대 PoC RLSC 2-way 만 가능하다

비용 어림의 근거는 실측이다. 시그니처 빌드가 2만 건에 약 4.4분, 5만 건에 약 9.9분이고 properties 추출이 137건에 4분(입력 23만 토큰)이었다. 5,000건이면 그 추출이 약 2.4시간, 입력 약 840만 토큰으로 어렵다.

코드가 어디 있는가

무엇	어디
규약을 코드로 옮긴 것(어휘, 타입, 스키마, capability 상수, establish/fallback/projection, query, audit, intersect)	src/lib/visual-summary/
항목 트리를 markdown 으로 바꾸는 다리	src/lib/visual-summary/markdown.ts
결함 채점과 귀속	src/lib/visual-summary/audit.ts
교집합과 순위 융합(RRF)	src/lib/visual-summary/intersect.ts
I.I.S. 대행 생산(경계 추출, properties 구조 대응 추출, 노드 검색, 노드 대치)	src/lib/visual-summary/iis/
Summary 생성 프롬프트와 재생성 루프	src/lib/visual-summary/generator/
규약 예시 18종, 안티패턴 5종, 평가셋 50건, 생성 스냅샷	src/lib/visual-summary/fixtures/
두 경로가 공유하는 후보 풀	src/app/api/visual-summary/indexPool.ts
comp-sign 실제 코드 호출	src/app/api/visual-summary/compSignService.ts
Eregion 화면	src/app/visual-summary/ (canvas-frame, iis-boundary, iis-boundary-gold)
평가 도구 CLI	scripts/visual-summary-eval.ts, scripts/visual-summary-poc.ts, scripts/visual-summary-real-poc.ts

src/lib/visual-summary/README.md 가 모듈 구성을 더 세밀하게 정리해 둔다. 핵심 파일은 다음과 같다. taxonomy.ts (어휘. 코드는 아직 62값이고 규약은 2026-08-29 에 결함값 41종으로 좁혔다. l-14. 점과 밑줄 표기 변환), parse.ts (YAML 파싱과 silent corruption 계측), assignIds.ts (id 부여), validate.ts (파싱→id→검증 파이프라인), capability.ts (45종 상수), establish.ts (성립 조건 판정), capabilityCheck.ts (카테고리별 정합성 검사, 성립 조건과 params 키를 error/warning/info 로 나눠 판정), projection.ts (P1~P8 계산과 적용), fallback.ts (카테고리 확정), query.ts (검색 질의 변환기), sheet.ts (sheet JSON 텍스트 대치와 box 보정), pipeline.ts (end-to-end 조립).

Eregion 화면

내부 도구(Eregion)의 사이드바에서 "Visual Summary PoC" 를 연다. 로컬 개발 환경에서는 계정 없이 열리고, 배포 빌드에서만 로그인 게이트가 강제된다. 탭이 다섯 개다. 플레이그라운드(임의 입력이나 YAML 직접 검증), 실사용 샘플(실로그 원문 10건 실행), 색인 현황, 대치 경로 비교(규약 RLSC 대치와 comp-sign textTree 대치를 컴포넌트별로 나란히), 노드 태그 검색(T-5, T-6 미리보기)이다. 결과 화면은 컴포넌트마다 카드로 서고, 카테고리 확정 이력, 검색 질의, 후보 순위와 점수, 대치 전후 텍스트, 원본과 대치 결과를 나란히 그린 캔버스가 함께 표시된다. 대치 결과 캔버스는 이미지 합성 서버를 왕복하지 않고 @miri-unicorn/miricanvas-editor-2 의 PageRootView (합성 서버가 쓰는 페이지 루트 컴포넌트)를 화면에 직접 마운트해서 그린다.

명령어

```
## 1) 생성 - 돈이 들고 드물게 돈다. 결과 스냅샷은 저장소에 커밋한다
DOTENV_CONFIG_PATH=.env.local pnpm tsx scripts/visual-summary-eval.ts --generate --k 1

## 2) 채점 - LLM 호출이 없고 결정적이다
DOTENV_CONFIG_PATH=.env.local NODE_OPTIONS="--max-old-space-size=12288" \
  pnpm tsx scripts/visual-summary-eval.ts --score --top 10 --out eval/baseline --html

## 3) 회귀 비교 - 조건이 다르면 거부한다
pnpm tsx scripts/visual-summary-eval.ts --diff eval/before/report.json eval/after/report.json
```

실사용 데이터 E2E 실측 (2026-08-12, gpt-5.4-mini)

실사용 로그 원문 10건으로 생성부터 실색인 검색, 내용 대치까지 끝까지 돌린 결과, 생성은 10/10 성공(재생성 루프 사용 7회), 검색은 15개 컴포넌트 전부 후보를 확보했으며(앞 절의 검색 군으로 1군 5, 2군 8, 3군 1, 4군 안전망 1. **절반 이상이 2군이라는 것은 생성기가 고른 카테고리를 색인이 그대로 대 주지 못한다는 뜻이다**), 대치는 entity 56개 중 56개를 수용했다(43개 배치, 13개 접기). 이 실행에서 시스템 구멍 두 가지를 찾아 코드에 반영했다. 하나는 같은 fallback 군 안에서 순서가 점수에 밀리는 문제였고(2군 정렬을 태그 순위 우선으로 고정), 다른 하나는 마지막 fallback 카테고리가 색인에 없으면 후보가 0이 되는 문제였다(4군 안전망 검색 추가).

아직 남은 것

PoC 가 직접 할 수 있는 것. 평가 pool 고정과 사전 적재(l-8), 사람 판정과 기계 판정의 일치율 측정, 랭킹 가중치 실험(MOR-1936), fallback 대체 순서의 열린 결정 실측(MOR-1934).

남의 답을 기다리는 것. comp-sign 4.3.8 버전을 사내 저장소에 올려 달라는 요청(이석님, 파이프 표 인식 문제로 markdown 기호가 결과에 그대로 나가는 문제가 컴포넌트당 137건 확인됨), 평가셋을 38개 중분류로 넓히는 작업(MOR-1928, 허재영님).

스쿼드가 정해야 하는 것. 운영 규모(178,095건) 색인을 어떻게 만들 것인가. 선택지는 이 저장소의 시그니처 워커를 comp-sign 4.3.x 로 올려 다시 적재하거나, 이석님 쪽 테이블을 직접 조회하는 것 두 가지다.

부록: 카테고리 와 필드 전량 명세

이 부록이 무엇인가

2절이 규약의 설계 원칙과 fallback 규칙을 설명하되 카테고리 와 필드는 대표 예시만 실었다. 이 부록은 그 자리를 채운다. **semantic category 45종 전량, params 키 27종 전량, structure 아래 필드 전량, 45종의 fallback 순서 전량과 옛지 131개 전량, 열린 결정 47건 전량, 자기정합 검사 14가지, 꾸밈구조 형 정의를** 표로 적는다. 읽고 나서 임의의 입력에 대해 어느 카테고리를 고를지, 그 카테고리에 무엇을 적을 수 있는지 판정할 수 있어야 한다.

찾는 것	절
대분류 21종이 각각 무엇을 담나	A
중분류 45종을 각각 언제 고르나	B
45종의 성립 조건과 실사용 빈도	C
카테고리별 params 키와 기본값	D
structure 아래 필드 전량과 값 범위	E
문서 사이에 아직 어긋나 있는 자리	F
검색 유형별로 어디로 물러나나. 45종 전량	G
fallback 옛지 131개 전량과 옛지마다 발동하는 projection 규칙	H
아직 안 정한 것 전량. 47건	I
문서와 코드가 어긋나지 않는지 보는 검사 14가지	J
꾸밈구조 형 정의와 해석 예시 넷	K

SSoT 는 이 부록이 아니다. 값의 정보는 카테고리 어휘가 spec/schema/visual-summary-0.5.schema.json 의 semanticCategory enum, 성립 조건과 params 가 spec/schema/capability.json, 선택 기준 문장이 reference/semantic-taxonomy-v7.md (원본은 단어장 시트)다. 어긋나면 그쪽을 따른다.

A. 대분류 21종

대분류는 중분류의 이름공간이다(2절 "type 을 문자열 하나로 적는 이유"). 아래는 스키마 semanticCategory enum 에 들어 있는 21종이고, 괄호 안이 단어장이 정한 정의다.

대분류	무엇을 담나
Enumeration	동등한 수준의 정보 단위를 관계나 흐름, 비교보다 목록화와 묶음 중심으로 보여줄 때
AttributeComparison	동일한 기준이나 속성 축을 기준으로 대상 간 차이, 강약점, 우열을 비교할 때
RelationalComparison	항목별 차이를 나열하기보다 대상 간 상대적 위치, 영역, 교집합, 구조적 연관성을 보여줄 때
Transformation	동일 대상의 전후 상태, 개선 방향, 변화 결과를 보여줄 때
ProcessFlow	시간표보다 절차, 순서, 실행 단계가 더 중요할 때
LogicFlow	조건, 판단, 분기, 의사결정 규칙에 따라 흐름이 달라지는 구조를 보여줄 때

대분류	무엇을 담나
Progression	상태, 수준, 성숙도, 성과가 단계적으로 높아지거나 발전하는 과정을 보여줄 때
TimelineSchedule	날짜, 기간, 순서, 마일스톤 등 시간 축을 기준으로 사건이나 작업을 보여줄 때
Hierarchy	상하 관계, 소속 관계, 계층 구조, 레이어 구조가 핵심일 때
ScopeCoverage	중심에서 외부로 확장되는 범위, 포함 영역, 적용 범위를 보여줄 때
Cycle	과정이나 상태가 한 방향으로 끝나지 않고 반복, 순환, 재귀적으로 이어질 때
ReasoningStructure	원인, 문제, 해결책, 조합 등 결과나 주장을 도출하는 논리 구조를 보여줄 때
KPIKeyMetric	핵심 숫자, 대표 성과 지표, 목표 대비 달성 상태를 강조할 때
TextBasedCommunication	별도의 관계 구조보다 문장이나 본문의 의미를 전달하는 것이 핵심일 때
QnACard	질문과 답변을 대응하여 보여줄 때
Utterance	질문, 대화, 증언, 인용문처럼 발화 형식의 콘텐츠를 보여줄 때
PortraitProfile	인물, 역할, 경력, 프로필 정보를 소개할 때
Conversion	단계가 진행될수록 대상이 선별, 축소, 전환되는 흐름과 변화 비율을 보여줄 때
ProportionComposition	전체 대비 구성 비율이나 상대적 규모 차이를 보여줄 때
Trend	시간이나 순서에 따른 수치 변화, 증가와 감소 패턴, 흐름을 보여줄 때
MagnitudeRanking	항목 간 수치 크기, 순위, 상대적 규모 차이를 비교할 때

enum 에 없는 대분류가 둘 있다. PageNavigation (페이지 역할 4종)은 2026-08-06 에 어휘로 편입하기로 정했지만(B-24) 스키마 enum 과 capability.json 에는 아직 반영돼 있지 않다. Geo (지도 전용 4종)는 지도 데이터가 있어야 성립해 structure 가 표현할 대상이 아니라서 뺐다(2026-08-17 스킴드 확인). F절에 다시 적는다.

단어장 쪽 수와 맞춰 읽는 법. 단어장 v7 은 대분류 23종, 중분류 49종이고 일반 카탈로그 29종과 표차트 카탈로그 23종으로 갈라 쓴다. 두 카탈로그에 Timeline, HierarchicalTree, Flow 셋이 겹쳐 들어 있어 52 에서 3 을 빼면 49 다. 여기서 지도 전용 4종을 빼면 이 부록이 다루는 45종이 된다.

B. 중분류 45종: 언제 고르나

번호는 spec/capability-table.md 2절의 행 번호와 같다. 표기는 규약(질의 쪽) 표기인 점 표기로 적었고, 색인과 검색이 읽는 저장 키는 같은 값의 밑줄 표기다(Enumeration.List 와 Enumeration_List 는 같은 카테고리다).

#	카테고리	언제 고르나
1	Enumeration.List	순서나 우열 없이 섹션 형태로 병렬 나열할 때. 시각 배치 형태가 달라도 동등 나열이면 모두 이 라벨 하나로 고른다
2	Enumeration.ListingTable	비교가 아니라 목록 정보를 행과 열로 구조화해 표로 정리할 때. 행이 항목이고 열이 그 항목의 속성이다
3	AttributeComparison.Versus	2개의 대상을 서로 대응시켜 차이점, 장단점, 우열을 직접 대조할 때. 세부 속성 비교보다 대결 구도 자체를 전달한다

#	카테고리	언제 고르나
4	AttributeComparison.ComparisonTable	여러 기준이나 속성을 행과 열로 정리해 여러 대상을 같은 기준으로 나란히 건줄 때. 행이 지표이고 열이 대상이라 ListingTable 과 축이 뒤집혀 있다
5	RelationalComparison.SWOT	내부와 외부, 긍정과 부정의 네 영역으로 전략적 상황 관계를 정리할 때
6	RelationalComparison.VennDiagram	집합 간 공통점과 차이점 관계를 원형 도형으로 교차해 동시에 보여줄 때
7	RelationalComparison.Quadrant	두 축을 기준으로 대상의 상대적 위치나 포지션을 네 영역으로 나누어 보여 줄 때
8	Transformation.Transformation	동일 대상의 Before/After, AS-IS/TO-BE, 개선 전후 상태를 보여줄 때. 전 후의 변화가 핵심이다
9	ProcessFlow.Steps	업무나 절차를 순차 단계로 구분해 진행 순서를 보여줄 때. 단계가 핵심이다
10	TimelineSchedule.Timeline	날짜, 기간, 순서, 마일스톤 등 시간 축을 기준으로 사건이나 작업을 보여줄 때
11	Progression.Staircase	상태, 수준, 성숙도, 성과가 단계적으로 상승하고 성장하는 것을 상향 형태로 보여줄 때
12	Cycle.Cycle	프로세스나 상태가 끝나지 않고 반복, 순환, 피드백되는 구조를 보여줄 때
13	LogicFlow.FlowChart	조건, 판단, 분기, 예외 처리에 따라 흐름이 달라지는 구조를 보여줄 때. 중간 에 분기가 생기는 것이 핵심이다
14	Conversion.Funnel	많은 대상이 단계별로 줄어들거나 선별되고 전환되는 퍼널 전환을 보여줄 때
15	TimelineSchedule.GanttChart	작업별 시작, 종료, 기간, 병렬 진행을 일정표 형태로 보여줄 때. 열에 시간, 행에 작업을 둔다
16	TimelineSchedule.Calendar	월 단위의 일정이나 이벤트를 달력 형태로 보여줄 때
17	Hierarchy.HierarchicalTree	상위와 하위, 부모와 자식, 조직 구조처럼 계층적 연결 관계를 보여줄 때
18	Hierarchy.Pyramid	넓은 기반에서 좁은 정점으로 쌓이는 층 구조. 각 층의 너비와 면적이 중요 도, 우선순위, 위계를 암시한다
19	ScopeCoverage.NestedScope	중심 개념을 기준으로 포함 범위와 적용 영역이 안쪽에서 바깥쪽으로 확장 되는 동심원 구조. TAM/SAM/SOM 이 전형이다
20	ReasoningStructure.Fishbone	문제나 결과의 원인을 여러 갈래로 분해해 분석할 때. 문제를 중심에 두고 복 수 원인이 퍼져나간다
21	ReasoningStructure.CombinedResult	여러 요소의 결합이 하나의 결과나 결론으로 이어지는 구조를 보여줄 때
22	ReasoningStructure.ProblemSolution	문제와 대응 방안, 기대 효과를 짝지어 보여줄 때
23	KPIKeyMetric.BigNumber	가장 중요한 수치나 KPI 를 크게 강조해 한눈에 전달할 때
24	KPIKeyMetric.Gauge	목표 대비 달성률, 진행률, 임계치 대비 상태를 게이지 형태로 보여줄 때
25	MagnitudeRanking.BarChart	항목 간 수치 크기, 순위, 상대적 규모를 막대 길이로 비교할 때
26	ProportionComposition.StackedBarChart	항목별 전체값 안의 구성 비중을 막대 세그먼트로 보여줄 때
27	ProportionComposition.PieChart	전체를 여러 조각으로 나누어 구성 비율을 보여줄 때

#	카테고리	언제 고르나
28	<code>ProportionComposition.DonutChart</code>	중앙 공간을 활용하면서 전체 대비 구성 비율을 보여줄 때
29	<code>ProportionComposition.TreeMap</code>	면적 크기로 전체 대비 구성 비율과 상대적 규모를 보여줄 때
30	<code>ProportionComposition.PictogramChart</code>	아이콘 반복이나 채움 정도로 비율과 수량을 직관적으로 보여줄 때
31	<code>Trend.LineAreaChart</code>	시간이나 순서에 따른 수치 변화와 추세를 선 또는 면적으로 보여줄 때
32	<code>Trend.ComboChart</code>	동일한 시간이나 순서 축 위에서 서로 다른 성격의 지표를 한 차트에 겹쳐 보여줄 때
33	<code>AttributeComparison.RadarChart</code>	동일한 평가 기준 여러 개에서 대상의 강약점이나 프로파일 차이를 수치로 비교할 때
34	<code>RelationalComparison.BubbleChart</code>	두 축의 위치와 버블 크기를 함께 써서 상대적 위치와 규모를 동시에 보여줄 때
35	<code>TextBasedCommunication.HighlightedMessage</code>	반드시 기억해야 할 하나의 핵심 문장이나 하나의 완결된 주장을 화면 중심에 크게 배치할 때
36	<code>TextBasedCommunication.LongFormTextLayout</code>	하나의 주제나 배경, 사례를 여러 문장 이상의 본문 흐름으로 풀어 설명할 때. 장문으로 시각화한 것이 핵심이다
37	<code>QnACard.QnACard</code>	질문과 답변이 쌍을 이루며 반복될 때. FAQ, OX 퀴즈, 골든벨이 전형이다
38	<code>Utterance.DialogueChat</code>	두 화자가 교번으로 메시지를 주고받는 대화를 말풍선이나 채팅 형태로 보여줄 때
39	<code>Utterance.QuoteTestimonial</code>	고객 후기, 인터뷰 인용, 핵심 발언을 강조할 때. 응답이나 교환 없이 단방향으로 제시한다
40	<code>PortraitProfile.ProfileCard</code>	인물의 사진, 이름, 역할, 핵심 정보를 한 카드 안에 압축해 소개할 때
41	<code>PortraitProfile.ResumeCV</code>	경력, 학력, 역량, 프로젝트 이력을 섹션과 구분선으로 문서형 배치할 때
42	<code>PageNavigation.Opening</code>	발표 슬라이드의 맨 앞장을 구성할 때만. 제목, 부제목, 발표자, 날짜를 중앙에 배치한다
43	<code>PageNavigation.TableOfContents</code>	발표의 전체 구성과 섹션, 순서를 번호와 제목으로 안내할 때. 흐름이나 단계 표현은 없다
44	<code>PageNavigation.SectionDivider</code>	발표 중간의 장 전환을 알릴 때. 콘텐츠 없이 단일 섹션의 번호와 제목만 중앙에 둔다
45	<code>PageNavigation.Ending</code>	발표 맨 마지막 장에서 종결 메시지만 크게 배치할 때. 본문 내용의 결론 요약은 해당하지 않는다

헛갈리기 쉬운 경계 넷. 첫째, `ListingTable` (행이 항목, 열이 속성)과 `ComparisonTable` (행이 지표, 열이 대상)은 전치 관계라서 무엇이 첫 열에 오는지로 가른다. 둘째, 발화 계열 셋은 교환이 있는지로 갈린다. `QuoteTestimonial` 은 단방향, `QnACard` 는 질문과 답변의 쌍, `DialogueChat` 은 두 화자의 교번이다. 셋째, `NestedScope` 와 `HierarchicalTree` 는 단일 사슬 포함 관계에서 구조가 같아지므로 동심원으로 그리는지로 가른다. 넷째, `Opening` 과 `Ending` 은 원본 시트 정의문이 서로 겹쳐 있었고, 2026-08-25에 황현진이 `Opening` 을 "맨 앞장만" 으로 좁혀 배타적으로 만들었다.

C. 45종의 성립 조건과 실사용 빈도

순서는 B절과 같다. 개수 조건의 단위는 따로 적지 않으면 `entities` 개수이고, 표에 적힌 계열과 조각, 행과 축은 그 카테고리가 세는 단위다. **실측** 표시는 배치 엔진 `jari 0.12.0` 의 `LAYOUT_CONTENT_CONTRACTS` 에서 그대로 가져온 값이고, 표시가 없는 값은 단어장 정의문에서 유도한 설계 판단이다. 비중은 2026-07-06 부터 07-12 까지 사용자 입력 7,168 block 의 질의 쪽 분포이고, 우선순위는 단어장 시트의 구현 우선순위 열(P0 이 가장 높다)이다.

카테고리	개수	dataset	필수 내용	비중과 우선순위
<code>Enumeration.List</code>	1 이상		없음	31.96% P0
<code>Enumeration.ListingTable</code>	행 2 이상	<code>matrix</code>	<code>matrix</code> 또는 <code>entities + columns</code>	15.08% P0
<code>AttributeComparison.Versus</code>	정확히 2		없음	3.11% P0
<code>AttributeComparison.ComparisonTable</code>	열 2 이상	<code>matrix</code>	<code>matrix</code> 또는 <code>entities + columns</code>	7.24% P0
<code>RelationalComparison.SWOT</code>	정확히 4		없음	0.11% P0
<code>RelationalComparison.VennDiagram</code>	2 이상 3 이하 실측		없음	0.08% P1
<code>RelationalComparison.Quadrant</code>	1 이상	<code>matrix</code>	<code>matrix</code> 필수. 항목마다 x, y	0.14% P3
<code>Transformation.Transformation</code>	정확히 2		없음	0.86% P0
<code>ProcessFlow.Steps</code>	2 이상		없음	10.92% P0
<code>TimelineSchedule.Timeline</code>	2 이상 15 이하 실측		없음	4.27% P0
<code>Progression.Staircase</code>	2 이상 8 이하 실측		없음	0.46% P0
<code>Cycle.Cycle</code>	2 이상 8 이하 실측		없음	0.50% P0
<code>LogicFlow.FlowChart</code>	2 이상		없음	0.39% P3
<code>Conversion.Funnel</code>	2 이상	<code>matrix</code> (선택)	없음	0.07% P2
<code>TimelineSchedule.GanttChart</code>	행 1 이상	<code>matrix</code>	<code>matrix</code> 필수. 작업마다 시작과 끝	0.28% P4

카테고리	개수	dataset	필수 내용	비중과 우선순위
TimelineSchedule.Calendar	1 이상	matrix	matrix 필수. 날짜와 일정	0.64% P4
Hierarchy.HierarchicalTree	1 이상 실측		children 필수	1.27% P1
Hierarchy.Pyramid	3 이상		없음	0.06% P1
ScopeCoverage.NestedScope	2 이상		children 필수	0.08% P1
ReasoningStructure.Fishbone	2 이상		없음	0.14% P3
ReasoningStructure.CombinedResult	2 이상		없음	0.17% P1
ReasoningStructure.ProblemSolution	1 이상		없음	0.85% P1
KPIKeyMetric.BigNumber	1 이상	measure (선택)	value 필수	1.20% P1
KPIKeyMetric.Gauge	1 이상	measure	measure.value 필수	0.42% P0
MagnitudeRanking.BarChart	계열 1 이상	matrix	matrix 필수	1.67% P0
ProportionComposition.StackedBarChart	계열 2 이상	matrix	matrix 필수	0.14% P0
ProportionComposition.PieChart	조각 2 이상	matrix	matrix 필수	0.31% P0
ProportionComposition.DonutChart	조각 2 이상	matrix	matrix 필수	0.20% P0
ProportionComposition.TreeMap	1 이상	matrix	matrix 필수	0건 P4
ProportionComposition.PictogramChart	1 이상	matrix	matrix 필수	0.06% P2
Trend.LineAreaChart	계열 1 이상, 축 2 이상	matrix	matrix 필수	0.70% P0
Trend.ComboChart	계열 2 이상	matrix	matrix 필수, 계열 type 2종 이상 또는 axis 2종 이상	0.25% P0
AttributeComparison.RadarChart	축 3 이상	matrix	matrix 필수	0.03% P2
RelationalComparison.BubbleChart	1 이상	matrix	matrix 필수. 항목마다 x, y, 크기	0건 P4
TextBasedCommunication.HighlightedMessage	정확히 1		없음	5.75% P0

카테고리	개수	dataset	필수 내용	비중과 우선순위
TextBasedCommunication.LongFormTextLayout	1 이상		없음	4.77% P0
QnACard.QnACard	1 이상		없음	0.46% P2
Utterance.DialogueChat	2 이상		없음	0.18% P1
Utterance.QuoteTestimonial	1 이상		없음	0.25% P1
PortraitProfile.ProfileCard	1 이상		없음	1.09% P2
PortraitProfile.ResumeCV	1 이상		없음	0.22% P2
PageNavigation.Opening	정확히 1		제목이 있고 발표자, 날짜, 소속 중 하나 이상이 함께 온다	0.99% P1
PageNavigation.TableOfContents	2 이상		뒤에 올 섹션의 번호와 제목을 나열한다. 흐름이나 단계 표현이 없다	0.88% P1
PageNavigation.SectionDivider	정확히 1		다음 한 섹션의 번호와 제목, 본문 콘텐츠가 없다	0.24% P1
PageNavigation.Ending	정확히 1		종료를 알리는 단일 메시지. 별도 콘텐츠가 없다	0.47% P1

성립 조건이 내용을 요구하는 것은 아홉이다. BigNumber 와 Gauge 의 수치, HierarchicalTree 와 NestedScope 의 children, ComboChart 의 혼합 조건, 그리고 페이지 역할 4종의 내용 조건이다. 나머지 36종은 개수 조건과 matrix 유무만 본다. 옛 테이블에서 필수였던 apex, author, speaker, intersects 는 전부 기본값을 가진 선택 값으로 내려갔다.

상위 5종이 질의의 약 65% 다. List, ListingTable, Steps, ComparisonTable, HighlightedMessage 순이고, 나머지 20여 종은 각 0.5% 미만의 롱테일이다. TreeMap 과 BubbleChart 는 표본 7,168 block 에 한 번도 나오지 않았다.

42번부터 45번은 2026-08-27 에 채웠다. 페이지 역할 4종은 어휘에는 들어왔지만(B-24, 2026-08-06) 성립 조건이 오래 비어 있었다. C-4(검색 풀 정책, 정원석)가 닫혀야 채울 수 있다고 적혀 있었는데, C-4 가 정하는 것은 그 종이 뽑힌 뒤의 템플릿 공급이지 성립 조건이 아니었다. 단어장 원본 시트에 넷의 조건이 이미 있어서 그 값으로 채웠다. dataset 은 넷 다 쓰지 않고 entities 로 적는다. Opening 이면 제목과 발표자, 날짜가 각각 entity 이고 TableOfContents 면 섹션마다 entity 하나다. Opening 을 맨 앞 장 전용으로 붙이는 단어장 오너(김수현) 확인 대기다.

D. params 전량 27종

params 는 표현 선택을 담는다. 스키마는 "객체다" 까지만 검사하고 허용 키와 기본값은 capability.json 의 reads.params 가 카테고리별로 정한다(2절 "params 를 열린 객체로 둔 이유"). 도착 카테고리가 선언하지 않은 키는 fallback 때 버려진다(P2). 성립 조건이 정해진 41종이 선언한 키를 모으면 27종이고, 아래가 전량이다.

키	어느 카테고리가 읽나
apex	Hierarchy.Pyramid

키	어느 카테고리가 읽나
axis	AttributeComparison.ComparisonTable, Trend.ComboChart
axisLabels	RelationalComparison.Quadrant
center	Cycle.Cycle
centerLabel	ProportionComposition.DonutChart
conversionRate	Conversion.Funnel
direction	Progression.Staircase, Transformation.Transformation
fill	Trend.LineAreaChart
header	Enumeration.ListingTable, AttributeComparison.ComparisonTable
icon	ProportionComposition.PictogramChart
labelPosition	ProportionComposition.PieChart, ProportionComposition.DonutChart
marker	ProcessFlow.Steps
maxValue	AttributeComparison.RadarChart
normalize	ProportionComposition.StackedBarChart
order	RelationalComparison.SWOT
orientation	Hierarchy.HierarchicalTree, TimelineSchedule.Timeline, MagnitudeRanking.BarChart, ProportionComposition.StackedBarChart
rotation	Cycle.Cycle
sequence	ProcessFlow.Steps
shape	ScopeCoverage.NestedScope

키	어느 카테고리가 읽나
sizeScale	RelationalComparison.BubbleChart
sort	MagnitudeRanking.BarChart
startAngle	ProportionComposition.PieChart, ProportionComposition.DonutChart
title	TextBasedCommunication.LongFormTextLayout
type	Trend.ComboChart
unit	TimelineSchedule.GanttChart
unitPerIcon	ProportionComposition.PictogramChart
weekStart	TimelineSchedule.Calendar

params 를 하나도 선언하지 않는 카테고리가 16종이다. Enumeration.List, AttributeComparison.Versus, RelationalComparison.VennDiagram, LogicFlow.FlowChart, ReasoningStructure 셋, KPIKeyMetric 둘, ProportionComposition.TreeMap, TextBasedCommunication.HighlightedMessage, QnACard.QnACard, Utterance 둘, PortraitProfile 둘이다. 이 카테고리로 fallback 하면 출발 쪽 params 는 전부 버려진다.

이름이 겹치는 키가 셋 있다. axis (표의 기준 축인가 / 계열이 붙는 축), type (params.type 은 계열 표현, component.type 은 카테고리, column.type 은 열의 차트 표현), unit (params.unit 은 간트의 기간 단위, entity.unit 과 measure.unit 은 수치 단위)이다. P2 는 키 이름만 보고 통과시키므로 뜻이 다른데도 살아남는 자리가 생긴다. ComboChart 에서 ComparisonTable 로 가는 옛지가 그 예이고, 규약은 이 옛지에서 axis 를 살아남는 것으로 적지 않는다.

E. structure 전량

structure 는 내용을 담고, 필드 이름이 모든 카테고리에서 같다(2절 설계 원칙). 카테고리가 못 읽는 필드는 오류가 아니라 무시다. 아래 표의 값 범위는 spec/schema/visual-summary-0.5.schema.json 에서 가져왔다.

E.1 Structure

entities 와 dataset 중 적어도 하나가 있어야 한다.

필드	타입	무엇인가
entities[]	Entity 배열, 1개 이상	항목 목록
relations[]	Relation 배열, 기본값 빈 배열	이 컴포넌트 안 항목 간의 관계
dataset	Measure 또는 Matrix	수치 데이터
columns[]	Column 배열	entities 로 표를 적을 때의 헤더. dataset 없이도 쓴다. ListingTable 과 ComparisonTable 두 카테고리 전용이다

E.2 Entity

`label` 만 필수다. 나머지는 그 카테고리가 읽을 수 있으면 읽는다.

필드	타입	무엇인가	주로 읽는 카테고리
<code>id</code>	문자열, <code>^[a-zA-Z0-9_-]+\$</code>	파싱 뒤 후처리가 부여한다	공통
<code>label</code>	문자열, 빈 값 불허	항목 이름	공통 (필수)
<code>items[]</code>	문자열 배열	하위 설명 목록	공통
<code>value</code>	수	수치	BigNumber, Gauge
<code>unit</code>	문자열	단위	BigNumber, Gauge
<code>period</code>	문자열	기간 표기	Timeline, ResumeCV
<code>emphasis</code>	참거짓	강조 여부	BigNumber
<code>weight</code>	수	상대 비중	Pyramid
<code>speaker</code>	문자열	화자	DialogueChat
<code>author</code>	문자열	발화자	QuoteTestimonial
<code>source</code>	문자열	인용 출처	QuoteTestimonial
<code>avatar</code>	문자열	인물 사진	ProfileCard
<code>organization</code>	문자열	소속	ProfileCard
<code>values[]</code>	문자열, 수, 널의 배열	<code>columns</code> 와 위치로 대응하는 나머지 열 값	ListingTable, ComparisonTable
<code>children[]</code>	Entity 배열	같은 컴포넌트 안의 하위 항목	HierarchicalTree, NestedScope, Fishbone, TreeMap, ResumeCV
<code>components[]</code>	Component 배열	컴포넌트 경계를 넘는 중첩. 1단계 생성기는 채워지지 않는다	없음

어느 카테고리가 어느 내용 필드를 읽는지가 **fallback** 을 좌우한다. 도착이 읽지 않는 내용 필드는 버려지지 않고 그 entity 의 `items` 맨 뒤에 이름: 값 형태로 붙는다(P3). 예를 들어 `ProfileCard` 에서 `List` 로 내려가면 `avatar` 와 `organization` 이 `items` 로 옮겨진다.

E.3 Relation

필드	타입	무엇인가
<code>type</code>	<code>intersects</code> 또는 <code>compares</code>	이 둘만 허용한다
<code>of[]</code>	문자열 배열, 2개 이상	관계로 묶이는 entity 의 <code>label</code> . 파싱 뒤 id 로 바뀐다
<code>desc</code>	문자열	그 관계에만 속하는 설명

관계를 읽는 카테고리는 둘뿐이다. `RelationalComparison.VennDiagram` 이 `intersects` 를, `AttributeComparison.Versus` 가 `compares` 를 읽는다. 성립 조건이 정해진 나머지 39종으로 가면 `relation.desc` 가 of 첫 항목의 `items` 맨 뒤로 간다(P7).

관계 어휘를 새로 만드는 기준은 하나다. 구조로는 적을 수 없는 제3의 내용을 가진 관계만 어휘로 만든다. 배열 순서나 entity 내부 필드, `children` 으로 이미 적히는 것은 별도 관계로 만들지 않는다.

E.4 Dataset : 두 모양

`shape` 값이 어느 모양인지 정한다.

measure (단일 수치). `shape` 와 `value` 가 필수다.

필드	타입	무엇인가
<code>shape</code>	고정값 "measure"	
<code>value</code>	수 (필수)	값
<code>unit</code>	문자열	단위
<code>min, max</code>	수	눈금 범위. Gauge 기본값은 0 과 100
<code>target</code>	수	목표값
<code>delta</code>	수	증감
<code>thresholds[]</code>	객체 배열	임계치. <code>at</code> (수, 필수), <code>label</code> (문자열), <code>state</code> (<code>safe</code> , <code>warn</code> , <code>danger</code>)

matrix (행렬). `shape` 와 `columns`, `rows` 가 필수다.

필드	타입	무엇인가
<code>shape</code>	고정값 "matrix"	
<code>columns[]</code>	<code>Column</code> 배열, 1개 이상	열 정의. 첫 열이 항상 범주 축이다
<code>rows[][]</code>	배열의 배열, 1개 이상	각 칸은 문자열, 수, 널 중 하나다

Column. `label` 만 필수다.

필드	값	무엇인가
<code>label</code>	문자열 (필수)	열 이름
<code>role</code>	<code>category</code> , <code>value</code> , <code>x</code> , <code>y</code> , <code>size</code> , <code>start</code> , <code>end</code> , <code>date</code>	열의 성격. 생략하면 첫 열은 <code>category</code> , 나머지는 <code>value</code> 로 간주한다
<code>unit</code>	문자열	단위
<code>axis</code>	<code>left</code> , <code>right</code>	이중 축에서 어느 축에 붙나. ComboChart 가 읽는다
<code>type</code>	<code>bar</code> , <code>line</code> , <code>area</code>	계열의 표현. ComboChart 가 읽는다

`role` 이 카테고리의 성립 조건과 이어진다. `Quadrant` 는 `x` 와 `y`, `BubbleChart` 는 `x` 와 `y` 와 `size`, `GanttChart` 는 `start` 와 `end`, `Calendar` 는 `date` 를 요구한다.

병합 헤더가 있는 표는 `matrix` 를 쓰지 않는다. `Column` 에 병합을 적을 자리가 없어서, `ListingTable` 과 `ComparisonTable` 두 카테고리만 `columns` 와 `entities` 의 조합을 대안으로 받는다(2026-08-06 추가). 이 경로로 채워진 컴포넌트는 `matrix` 가 필수인 카테고리로 fallback 하지 않는다.

E.5 Decoration

카테고리와 무관한 꾸밈 선택. `structure` 의 형제로 두고, page 수준과 컴포넌트 수준 양쪽에 적을 수 있다.

필드	대응하는 RLSC role	값
<code>ornament</code>	<code>Element.Decoration</code> , <code>LayoutContainer.Decoration</code>	<code>prefer</code> , <code>indifferent</code> (기본값), <code>avoid</code>
<code>background</code>	<code>Element.Background</code> , <code>LayoutContainer.Background</code>	위와 같다

`require` (필수)는 없다. 꾸밈을 필수로 걸면 후보가 0이 되어 전부 `List` 로 강등되기 때문이다. 해석은 축마다 따로 하고 구체적인 쪽을 우선한다. 컴포넌트, page, 카테고리 기본값, 시스템 전역 기본값, `indifferent` 순이다. 1단계 생성기는 이 필드를 채우지 않는다.

F. 알려진 격차 다섯

이 부록과 다음 부록을 쓰면서 문서 사이에 어긋난 자리가 다섯 확인됐다. 파생물 쪽이 규약 본문보다 뒤쳐진 것이 셋이고, 3번은 2026-08-27 에 닫혔다.

#	무엇이 어긋나 있나	어느 쪽이 맞나
1	<code>visual-summary-0.5.schema.json</code> 에 <code>structure.columns</code> 와 <code>entity.values[]</code> 가 없다 2026-08-29 에 닫혔다. 규약 2.2절에서 파일을 다시 내보냈다	없음
2	<code>semanticCategory-enum</code> 과 <code>capability.json</code> 에 <code>PageNavigation</code> 4종이 빠져 있다 2026-08-29 에 전부 닫혔다. 규약 enum 이 결합값 45종(0.5.3), <code>capability.json</code> 도 45종과 옛지 131개가 됐다	없음
3	<code>capability-table.md</code> 2.6절의 페이지 역할 4종 성립 조건은 2026-08-27 에 채워져 이 격차가 닫혔다. 남은 것은 <code>Opening</code> 이 맨 앞장 전용인지 하나다	골든셋은 맨 앞장 전용으로 만들었다 (2026-08-25 황현진). 단어장 오너 확인 대기다
4	fallback 옛지 수가 131개와 128개로 함께 적혀 있다 2026-08-29 에 오تام으로 판정했다. 남아 있는 "128개"는 전부 "125개(3차 반영 뒤 128개)에서 131개로" 같은 이력 서술이고 현재 값을 주장하는 자리가 아니다	현재 값은 131개다
5	<code>capability.json</code> 이 <code>KPIKeyMetric.BigNumber</code> 의 선택 내용에 <code>delta</code> 를 넣어 뒀는데 스키마의 <code>entity</code> 에는 <code>delta</code> 필드가 없다. <code>measure</code> 에만 있다	항목이 여럿인 <code>BigNumber</code> 는 증감을 적을 자리가 없다. 둘 중 어느 쪽을 맞출지 정해야 한다

`spec/schema/README.md` 가 정의한 자기정합 검사 14가지 중 S2(파일이 규약 2.2절과 같은가)와 C1(capability 종수가 스키마 결합값과 같은가)이 1번과 2번을 잡는 검사다. 마지막으로 전량을 돌린 것은 2026-08-04 였고 그 뒤 개정 다섯 번이 검사 없이 들어갔는데, 2026-08-29 에 전량 다시 돌려 14가지를 통과시켰다.

검사 정의 자체도 남아 있었다. F2("어느 행도 대체 후보를 셋 이상 두지 않는가")는 두 칸 상한이 2026-08-17 에 없어져 폐기했고, F3 은 "종단 셋으로 끝나는가"였는데 같은 날 `List` 뒤에 `ListingTable` 을 안전망으로 덧붙인 행이 넷 생겨(`List`, `SWOT`, `ProfileCard`, `TableOfContents`) "거처거나 자신이 그 셋인가" 로 고쳤다. S3, C1, F1, F4, N1 의 수치도 45종과 131개로 맞췄다. 이 실행에서 새로 잡힌 결합은 `yaml-examples.md` 5.2절 예시가 `type: Hierarchy` (대분류 단독)를 쓰고 있어 0.5.2 뒤 검증에 걸린 것 하나다.

G. 45종의 fallback 순서 전량

이 표가 "검색 유형별로 어디로 물러나나"에 답한다. 2절이 대표 체인 여섯 줄만 실었고 여기가 전량이다. 순서는 B절과 C절과 같다. 원문은 spec/capability-table.md 3.2절이고 2026-08-17 에 3차에 걸쳐 45종 전량을 확정했다.

읽는 법. 가운데 열은 그 카테고리 하나의 후속 우선순위 목록이고 왼쪽부터 본다. 다른 카테고리의 행을 이어 붙이지 않는다.

AttributeComparison.Versus 행이 ComparisonTable → List 인 것은 "Versus 가 안 되면 ComparisonTable, 그것도 안 되면 List" 라는 뜻이고, ComparisonTable 의 fallback 이 List 라는 뜻이 아니다. ComparisonTable 자신의 fallback 은 자기 행이 정하고 ListingTable → Versus → List 다. 그래서 Versus 에서 출발한 대치는 ListingTable 을 거치지 않는다(2026-08-04 확정). 목록 길이의 합이 H절 옛지 수 131과 같다.

서로를 첫 자리로 가리키는 쌍이 여덟 있고 순환이 아니다. 검색 모듈이 자기 행만 따라가므로 도착 카테고리의 행으로 넘어가는 일이 없고, 되돌아갈 자리가 애초에 생기지 않는다. 방문 기록도 필요 없다.

카테고리	후속 우선순위	한 번 따라갈 때 잃는 것
Enumeration.List	ListingTable	행과 열의 대응. List 자체는 entities 1개 이상이면 구조적으로 항상 성립하지만(2절 마지막 fallback 셋), 색인에 List 태그 후보가 없으면 ListingTable 로 물러난다
Enumeration.ListingTable	ComparisonTable → List	행과 열의 대응
AttributeComparison.Versus	ComparisonTable → List	좌우 대비의 시각적 강조
AttributeComparison.ComparisonTable	ListingTable → Versus → List	기준 축에 따른 비교 구도. Versus (정확히 2)를 거치면 다시 List로 접힐 수 있어 안전망으로 뒀다
AttributeComparison.RadarChart	BarChart → LineAreaChart → ComparisonTable → List	다축 프로파일의 형태. ComparisonTable 은 같은 AttributeComparison 대분류라 열 2개 이상이면 대부분 여기서 끝난다
RelationalComparison.SWOT	List → ListingTable	내부와 외부, 긍정과 부정의 2축 배치. Quadrant · ComparisonTable 경유 대신 List 를 1순위로 확정했다(원저자 판단 채택)
RelationalComparison.VennDiagram	Versus → ComparisonTable → List	교집합 영역. Versus 는 정확히 2가 조건이라 ComparisonTable 경유를 안전망으로 더했다
RelationalComparison.Quadrant	ComparisonTable → List	2축 위 상대 위치
RelationalComparison.BubbleChart	PieChart → DonutChart → BarChart → List	x축 위치와 버블 크기. 같은 RelationalComparison 대분류인 PieChart · DonutChart 를 먼저 거친다
Transformation.Transformation	Steps → Versus → ProblemSolution → List	전후의 방향성. Steps (ProcessFlow, "단계")가 Versus 보다 의미상 가깝다고 판단해 1순위로 뒀다
ProcessFlow.Steps	Timeline → Staircase → List	절차라는 의미. Timeline 은 시간축, Staircase 는 상승 방향으로 읽힌다

카테고리	후속 우선순위	한 번 따라갈 때 잃는 것
TimelineSchedule.Timeline	Steps → Staircase → List	시간축. Steps 는 순서만, Staircase 는 상승 방향만 남는다
Progression.Staircase	Steps → Timeline → Pyramid → List	상승이라는 방향. Steps (순서), Timeline (시간축), Pyramid (위계)를 차례로 거친다
Cycle.Cycle	Steps → List	순환. 끝이 처음으로 돌아간다는 사실
LogicFlow.FlowChart	Steps → List	분기
Conversion.Funnel	Steps → BarChart → LineAreaChart → ComparisonTable → List	단계마다 줄어든다는 사실. v1의 차트 후보 셋을 안전망으로 전부 살렸다(2026-08-17 사용자 결정)
TimelineSchedule.GanttChart	Timeline → Calendar → ListingTable → List	기간의 길이와 병렬 진행. Calendar (같은 TimelineSchedule)를 안전망으로 더했다
TimelineSchedule.Calendar	Timeline → ListingTable → GanttChart → List	날짜 격자. GanttChart (같은 TimelineSchedule)를 안전망으로 더했다
Hierarchy.HierarchicalTree	Pyramid → List	분기 연결선. Pyramid (개수 3 이상)가 위계를 층으로 대신 표현한다
Hierarchy.Pyramid	NestedScope → Staircase → List	층 넓이가 나타내던 위계. NestedScope 와 상호참조라 각자 안전망이 없으면 둘 다 막힌다
ScopeCoverage.NestedScope	Pyramid → Staircase → List	범위의 크기. Pyramid 와 상호참조
ReasoningStructure.Fishbone	ProblemSolution → List	원인이 한 결과로 모인다는 방향. 같은 ReasoningStructure 대분류인 ProblemSolution 으로 직행한다
ReasoningStructure.CombinedResult	Steps → ProblemSolution → List	결합의 방향. Steps (ProcessFlow, "단계")가 원인 분해보다 의미상 가깝다고 판단했다
ReasoningStructure.ProblemSolution	CombinedResult → Transformation → Steps → List	문제와 해결의 짝. CombinedResult (같은 ReasoningStructure)를 먼저 거친다
KPIKeyMetric.BigNumber	Gauge → HighlightedMessage	수치의 크기 강조. Gauge (같은 KPIKeyMetric)로 목표 대비 위치를 대신 표현한다
KPIKeyMetric.Gauge	BigNumber → PictogramChart → BarChart → HighlightedMessage	목표 대비 위치. PictogramChart · BarChart 는 근접도(원칙 2)로 우선순위에 넣었다. measure → matrix projection 규칙(P9)은 아직 미정이라 B-30 대기. 마지막은 HighlightedMessage (entities 1개 강조형)로 끝난다
MagnitudeRanking.BarChart	LineAreaChart → ComparisonTable → ListingTable → List	수치의 상대 크기. LineAreaChart (상호참조)를 거쳐 ComparisonTable, ListingTable 로 물러난다

카테고리	후속 우선순위	한 번 따라갈 때 잃는 것
ProportionComposition.StackedBarChart	BarChart → LineAreaChart → List	구성 비중. 계열 합계만 남는다. LineAreaChart 를 안전망으로 더했다
ProportionComposition.PieChart	DonutChart → BarChart → ComparisonTable → List	도넛과는 가운데 공간 차이뿐이다. ComparisonTable (열 2개 이상)을 안전망으로 더했다
ProportionComposition.DonutChart	PieChart → BarChart → ComparisonTable → List	가운데 라벨 자리. ComparisonTable 을 안전망으로 더했다
ProportionComposition.TreeMap	PieChart → DonutChart → List	면적으로 나타내던 비율과 중첩. 같은 ProportionComposition 대분류인 PieChart , DonutChart 를 먼저 거친다
ProportionComposition.PictogramChart	Gauge → BigNumber → BarChart → List	아이콘 반복의 직관성. Gauge · BigNumber 는 근접도로 우선순위에 넣었다. matrix → measure projection 규칙(P10)은 아직 미정이라 B-30 대기
Trend.LineAreaChart	BarChart → ComparisonTable → List	추세선의 연속성. BarChart 와 상호참조, ComparisonTable 을 안전망으로 더했다
Trend.ComboChart	ComparisonTable → ListingTable → List	이중 축과 계열 타입 혼합. 차트 계열로는 물러나지 않는다. 계열이 전부 bar / line 인 ComboChart 는 언제나 이중 축이고, 이중 축은 ComboChart 에서만 지원되므로 막대나 선으로 내리면 반드시 축 하나가 사라진다 (2026-08-21 개정)
TextBasedCommunication.HighlightedMessage	QuoteTestimonial → List → LongFormTextLayout	인용 부호와 출처 표기. entities 1개·한 문장이면 구조적으로 항상 성립하지만(2절 마지막 fallback 셋), 색인 공급이 부족하면 List 나 LongFormTextLayout 으로 물러난다
TextBasedCommunication.LongFormTextLayout	List	여러 문장 본문 구조. entities 1개·여러 문장이면 구조적으로 항상 성립하지만, 색인 공급이 부족하면 List 로 물러난다
QnACard.QnACard	DialogueChat → List	카드 형태
Utterance.DialogueChat	QuoteTestimonial → QnACard → List	말풍선과 화자 구분. QuoteTestimonial (같은 Utterance 대분류)을 먼저 거친다
Utterance.QuoteTestimonial	DialogueChat → HighlightedMessage	인용 부호와 출처 표기. DialogueChat 과 상호참조
PortraitProfile.ProfileCard	ResumeCV → List → ListingTable	사진과 카드 배치. ResumeCV (같은 PortraitProfile)를 1순위로 확정했다
PortraitProfile.ResumeCV	ProfileCard → Timeline → ListingTable → List	이력 항목의 기간 정렬. ProfileCard (같은 PortraitProfile)를 먼저 거친다
PageNavigation.Opening	HighlightedMessage → List	없음. 발표 맨 앞장은 전형적으로 제목 한 줄짜리 단일 메시지형이라 HighlightedMessage 를 1순위로 뒀다

카테고리	후속 우선순위	한 번 따라갈 때 읽는 것
PageNavigation.TableOfContents	List → ListingTable	목차라는 구조. 나열형이라 List 가 1순위이고, 항목이 많으면 ListingTable 로 물러난다
PageNavigation.SectionDivider	HighlightedMessage → List	없음. 장 전환은 제목 한 줄짜리 단일 메시지형이라 HighlightedMessage 를 1순위로 뒀다
PageNavigation.Ending	HighlightedMessage → List	없음. 마지막 장의 종결 메시지는 단일 메시지형이라 HighlightedMessage 를 1순위로 뒀다

H. fallback 엣지 131개 전량과 적용 규칙

G절의 각 행이 자기 카테고리에서 나가는 엣지를 정한다. 행마다의 목록 길이를 합하면 131개이고 중복이 없다. **규칙 열은 그 엣지에서 실제로 발동하는 projection 규칙**이고 규칙 자체의 정의는 2절 'structure projection' 절에 있다. 원문은 spec/capability-table.md 3.5절이다.

엣지 수의 변천은 F절 4번에 적었다. 51개에서 75개(2026-08-04), 111개와 125개(2026-08-17), 131개(2026-08-21 ComboChart 개정)로 다시 계산됐고 131개가 현재 값이다.

출발	도착	규칙	무엇이 옮겨지나
BarChart	ComparisonTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
BarChart	LineAreaChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
BarChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit, value 가 items 로 간다, orientation, sort 는 표현 선택이라 버린다
BarChart	ListingTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
BigNumber	Gauge	P1 P2 P3	unit, delta, emphasis
BigNumber	HighlightedMessage	P1 P2 P3 P5 P8	value 와 unit 이 entity로 간다, unit, delta, emphasis, 1개를 넘으면 나머지를 마지막 항목에 접는다
BubbleChart	BarChart	P1 P2	sizeScale 는 표현 선택이라 버린다
BubbleChart	DonutChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
BubbleChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit, value 가 items 로 간다, sizeScale 는 표현 선택이라 버린다
BubbleChart	PieChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Calendar	GanttChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Calendar	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit, value 가 items 로 간다, weekStart 는 표현 선택이라 버린다
Calendar	ListingTable	P1 P2	weekStart 는 표현 선택이라 버린다
Calendar	Timeline	P1 P2 P4 P8	matrix 의 행이 entities 가 된다, 15개를 넘으면 나머지를 마지막 항목에 접는다
CombinedResult	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다

출발	도착	규칙	무엇이 옮겨지나
CombinedResult	ProblemSolution	P1 P2	옮길 것이 없다. 같은 것을 읽는다
CombinedResult	Steps	P1 P2	옮길 것이 없다. 같은 것을 읽는다
ComboChart	ComparisonTable	P1 P2	type 은 표현 선택이라 버린다. 표는 축을 그리지 않아 이중 축 표현이 사라진다
ComboChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, axis , type 은 표현 선택이라 버린다
ComboChart	ListingTable	P1 P2	axis , type 은 표현 선택이라 버린다
ComparisonTable	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, axis , header 는 표현 선택이라 버린다
ComparisonTable	ListingTable	P1 P2	axis 는 표현 선택이라 버린다
ComparisonTable	Versus	P1 P2 P4 P8	matrix 의 행이 entities 가 된다, 2개를 넘으면 나머지를 마지막 항목에 접는다
Cycle	List	P1 P2	center , rotation 는 표현 선택이라 버린다
Cycle	Steps	P1 P2	center , rotation 는 표현 선택이라 버린다
DialogueChat	List	P1 P2 P3	speaker 가 items 로 간다
DialogueChat	QnACard	P1 P2 P3	speaker 가 items 로 간다
DialogueChat	QuoteTestimonial	P1 P2 P3	speaker
DonutChart	BarChart	P1 P2	centerLabel 는 표현 선택이라 버린다
DonutChart	ComparisonTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
DonutChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, centerLabel , labelPosition , startAngle 는 표현 선택이라 버린다
DonutChart	PieChart	P1 P2	centerLabel 는 표현 선택이라 버린다
Ending	HighlightedMessage	P1 P2 P8	1개를 넘으면 나머지를 마지막 항목에 접는다
Ending	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Fishbone	List	P1 P2 P3	children 가 items 로 간다
Fishbone	ProblemSolution	P1 P2 P3	children
FlowChart	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
FlowChart	Steps	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Funnel	BarChart	P1 P2	conversionRate 는 표현 선택이라 버린다
Funnel	ComparisonTable	P1 P2	conversionRate 는 표현 선택이라 버린다
Funnel	LineAreaChart	P1 P2	conversionRate 는 표현 선택이라 버린다

출발	도착	규칙	무엇이 옮겨지나
Funnel	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, conversionRate 는 표현 선택이라 버린다
Funnel	Steps	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, conversionRate 는 표현 선택이라 버린다
GanttChart	Calendar	P1 P2	옮길 것이 없다. 같은 것을 읽는다
GanttChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, unit (기간 단위) 는 표현 선택이라 버린다
GanttChart	ListingTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
GanttChart	Timeline	P1 P2 P3 P4 P8	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, 15개를 넘으면 나머지를 마지막 항목에 접는다
Gauge	BarChart	P1 P2 P3 P9	value 가 1행짜리 matrix 로 접힌다 (신규 규칙, 확정 아님. TODO), target , max , min , thresholds
Gauge	BigNumber	P1 P2 P3	target , max , min , thresholds
Gauge	HighlightedMessage	P1 P2 P3 P5 P8	value 와 unit 이 entity로 간다, target , max , min , thresholds , 1개를 넘으면 나머지를 마지막 항목에 접는다
Gauge	PictogramChart	P1 P2 P3 P9	value 가 1행짜리 matrix 로 접힌다 (신규 규칙, 확정 아님. TODO), target , max , min , thresholds
HierarchicalTree	List	P1 P2 P3	children 가 items 로 간다, orientation 는 표현 선택이라 버린다
HierarchicalTree	Pyramid	P1 P2 P3	children
HighlightedMessage	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
HighlightedMessage	LongFormTextLayout	P1 P2	옮길 것이 없다. 같은 것을 읽는다
HighlightedMessage	QuoteTestimonial	P1 P2	옮길 것이 없다. 같은 것을 읽는다
LineAreaChart	BarChart	P1 P2	fill 는 표현 선택이라 버린다
LineAreaChart	ComparisonTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
LineAreaChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, fill 는 표현 선택이라 버린다
List	ListingTable	P1 P2 P6	entities 가 matrix 의 행이 된다
ListingTable	ComparisonTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
ListingTable	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, header 는 표현 선택이라 버린다
LongFormTextLayout	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다

출발	도착	규칙	무엇이 옮겨지나
NestedScope	List	P1 P2 P3	children 가 items 로 간다, shape 는 표현 선택이라 버린다
NestedScope	Pyramid	P1 P2 P3	children
NestedScope	Staircase	P1 P2 P3 P8	children , 8개를 넘으면 나머지를 마지막 항목에 접는다
Opening	HighlightedMessage	P1 P2 P8	1개를 넘으면 나머지를 마지막 항목에 접는다
Opening	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
PictogramChart	BarChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
PictogramChart	BigNumber	P1 P2 P4	matrix 의 행이 entities 가 된다
PictogramChart	Gauge	P1 P2 P10	대표값(1행)이 value 로 간다, 나머지 행은 items 로 흘러보낸다 (신규 규칙, 확정 아님. TODO)
PictogramChart	List	P1 P2 P4	matrix 의 행이 entities 가 된다
PieChart	BarChart	P1 P2	labelPosition , startAngle 는 표현 선택이라 버린다
PieChart	ComparisonTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
PieChart	DonutChart	P1 P2	labelPosition , startAngle 는 표현 선택이라 버린다
PieChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, labelPosition , startAngle 는 표현 선택이라 버린다
ProblemSolution	CombinedResult	P1 P2	옮길 것이 없다. 같은 것을 읽는다
ProblemSolution	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
ProblemSolution	Steps	P1 P2	옮길 것이 없다. 같은 것을 읽는다
ProblemSolution	Transformation	P1 P2 P8	2개를 넘으면 나머지를 마지막 항목에 접는다
ProfileCard	List	P1 P2 P3	avatar , organization
ProfileCard	ListingTable	P1 P2 P3 P6	entities 가 matrix 의 행이 된다, avatar , organization
ProfileCard	ResumeCV	P1 P2 P3	avatar , organization
Pyramid	List	P1 P2 P3	weight 가 items 로 간다, apex 는 표현 선택이라 버린다
Pyramid	NestedScope	P1 P2 P3	weight
Pyramid	Staircase	P1 P2 P3 P8	weight 가 items 로 간다, apex 는 표현 선택이라 버린다, 8개를 넘으면 나머지를 마지막 항목에 접는다
QnACard	DialogueChat	P1 P2	옮길 것이 없다. 같은 것을 읽는다
QnACard	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Quadrant	ComparisonTable	P1 P2	axisLabels 는 표현 선택이라 버린다

출발	도착	규칙	무엇이 옮겨지나
Quadrant	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, axisLabels 는 표현 선택이라 버린다
QuoteTestimonial	DialogueChat	P1 P2 P3	author , source
QuoteTestimonial	HighlightedMessage	P1 P2 P3 P8	author , source 가 items 로 간다
RadarChart	BarChart	P1 P2	maxValue 는 표현 선택이라 버린다
RadarChart	ComparisonTable	P1 P2	옮길 것이 없다. 같은 것을 읽는다
RadarChart	LineAreaChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
RadarChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, maxValue 는 표현 선택이라 버린다
ResumeCV	List	P1 P2 P3	children , period 가 items 로 간다
ResumeCV	ListingTable	P1 P2 P3 P6	children , period 가 items 로 간다, entities 가 matrix 의 행이 된다
ResumeCV	ProfileCard	P1 P2 P3	period , children
ResumeCV	Timeline	P1 P2 P3 P8	children , 15개를 넘으면 나머지를 마지막 항목에 접는다
SWOT	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
SWOT	ListingTable	P1 P2 P6	entities 가 matrix 의 행이 된다
SectionDivider	HighlightedMessage	P1 P2 P8	1개를 넘으면 나머지를 마지막 항목에 접는다
SectionDivider	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
StackedBarChart	BarChart	P1 P2	normalize 는 표현 선택이라 버린다
StackedBarChart	LineAreaChart	P1 P2	옮길 것이 없다. 같은 것을 읽는다
StackedBarChart	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다, normalize , orientation 는 표현 선택이라 버린다
Staircase	List	P1 P2	direction 는 표현 선택이라 버린다
Staircase	Pyramid	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Staircase	Steps	P1 P2	direction 는 표현 선택이라 버린다
Staircase	Timeline	P1 P2 P8	15개를 넘으면 나머지를 마지막 항목에 접는다
Steps	List	P1 P2	marker , sequence 는 표현 선택이라 버린다
Steps	Staircase	P1 P2 P8	8개를 넘으면 나머지를 마지막 항목에 접는다
Steps	Timeline	P1 P2 P8	marker , sequence 는 표현 선택이라 버린다, 15개를 넘으면 나머지를 마지막 항목에 접는다
TableOfContents	List	P1 P2	옮길 것이 없다. 같은 것을 읽는다
TableOfContents	ListingTable	P1 P2 P6	entities 가 matrix 의 행이 된다

출발	도착	규칙	무엇이 옮겨지나
Timeline	List	P1 P2 P3	period 가 items 로 간다, orientation 는 표현 선택이라 버린다
Timeline	Staircase	P1 P2 P3 P8	period , 8개를 넘으면 나머지를 마지막 항목에 접는다
Timeline	Steps	P1 P2 P3	period 가 items 로 간다, orientation 는 표현 선택이라 버린다
Transformation	List	P1 P2	direction 는 표현 선택이라 버린다
Transformation	ProblemSolution	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Transformation	Steps	P1 P2	옮길 것이 없다. 같은 것을 읽는다
Transformation	Versus	P1 P2	direction 는 표현 선택이라 버린다
TreeMap	DonutChart	P1 P2 P3	children
TreeMap	List	P1 P2 P3 P4	matrix 의 행이 entities 가 된다, unit , value 가 items 로 간다
TreeMap	PieChart	P1 P2 P3	children
VennDiagram	ComparisonTable	P1 P2 P6 P7	entities 가 matrix 의 행이 된다, 관계의 desc 가 items 로 간다
VennDiagram	List	P1 P2 P7	관계의 desc 가 items 로 간다
VennDiagram	Versus	P1 P2 P7 P8	관계의 desc 가 items 로 간다, 2개를 넘으면 나머지를 마지막 항목에 접는다
Versus	ComparisonTable	P1 P2 P6 P7	entities 가 matrix 의 행이 된다, 관계의 desc 가 items 로 간다
Versus	List	P1 P2 P7	관계의 desc 가 items 로 간다

I. 열린 결정 전량 (2026-08-28 기준)

본문 3절은 할 일 SSoT 의 절 구조와 절마다의 남은 건수만 실었다. 이 부록이 그 항목 전량이다. 원문은 TODO.md 이고 각 항목의 근거 문단도 거기 있다. 상태가 [x] (단함)인 항목은 뺐다. 문서 정합 백로그(H절) 4건은 저장소 전체의 표기 통일 작업이라 이 부록에서 다루지 않는다.

열린 항목은 모두 47건이다(2026-08-29 에 B-33, I-13, I-14 가 늘었다).

A절. 규약 확정 (1건)

#	무엇을 정해야 하나	누가, 어떤 근거로
A-6	UNICORN-77437 깨진 링크 정리	정원석

B절. 규약 열린 결정 (16건)

#	무엇을 정해야 하나	누가, 어떤 근거로
B-6	<code>component.confidence</code> 를 스키마에서도 제거할지	지금은 예시에서만 제거(생성기가 안 채움)
B-7	갭 3종(big_number, quote, dialogue)을 투영으로만 돌지, RLSC 에 전용 Role 조합을 표준화할지	capability 테이블 상태 '갭'
B-9	<code>tree</code> 의 children → Graph relations 변환 규칙, <code>nested_scope</code> 와 DisjointEuler 의미 정합	I.I.S. 와 협의 필요
B-12	<code>stacked_bar</code> 의 stacking 을 RLSC ChartType 어디에 표식할지	현재 ChartType 에 stacking 플러그 없음
B-13	<code>dataset</code> 을 DataViz Matrix 와 encoding 으로 바로 넘길 수 있는지 어댑터로 실측	<code>T0D0.md</code> 의 근거 문단 참고
B-19	comp-sign 변형 3종을 대치 시점 처리로 옮길지	옮기면 색인이 한 벌로 줄어 컴포넌트당 최대 9벌이 되지 않는다. 이석님 확인 전에 는 확정 이 아니다. 1-2 의 남은 절반이다 → MOR-1933
B-20	<code>entity.components[]</code> 를 생성기가 언제부터 채울지	B-2 와 B-3 양쪽의 전제다. 계층(B-3)과 타이틀·디스크립션(B-2 의 DESC 판정)이 같은 수단을 쓴다. 스키마는 열려 있으나 생성기가 안 채워서 지금은 둘 다 <code>items</code> 로 평탄화된 채 읽힌다. 허재영님 → MOR-1911
B-22	장식 role 표식을 색인으로 넘기는 경로	이것이 먼저 풀려야 검색이 대조할 수 있다. 추출은 role 을 이미 붙이는데 색인에 노드 분류를 담는 자리가 없다. 1-4 와 같은 건이다 (<code>design/decoration-model.md</code> 4.8절) → MOR-1935
B-23	꾸밈구조 용어 확정	내용구조/ <code>structure</code> 와 꾸밈구조/ <code>decoration</code> , 꾸밈 축/ <code>decoration facet</code> , 선호값/ <code>preference</code> . '선호 축'과 '장식 축'은 쓰지 않는다. 제안 상태 다 (<code>design/decoration-model.md</code> 0절)
B-25	대분류(major)를 제거하고 중분류(minor)만 남겨도 되는지	허재영님이 2026-08-04 회의 안건으로 제기했다 (Slack). B-24 는 49종으로 닫혔고 이 항목은 별개로 남는다
B-27	도착의 개수 상한을 넘을 때 P8(접기)인지 규칙 1(후보 탈락)인지	<code>T0D0.md</code> 의 근거 문단 참고. 규약 5절 1번과 <code>capability-table.md</code> 3.4 절 P8 이 같은 상황에 서로 다른 답을 준다 → MOR-1934
B-28	처음 고른 카테고리가 못 읽는 <code>relations</code> 와 필드를 어디로 보낼지	<code>T0D0.md</code> 의 근거 문단 참고. 3.4절 규칙 여덟은 fallback 옛지에만 걸려 있어서, fallback 이 일어나지 않으면 적용되지 않는다. 3항 이상 <code>compares</code> 가 이 구명에 그대로 빠진다 → MOR-1934
B-29	축이 갈라지는 독립적 <code>rowspan</code> (주축과 무관하게 특정 컬럼 하나만 여러 행에 걸쳐 값이 반복되는 병합)을 어떻게 담을지	<code>T0D0.md</code> 의 근거 문단 참고. <code>entities + children</code> 트리 하나로는 안 담긴다
B-30	<code>measure</code> 와 <code>matrix</code> 사이의 projection 규칙 (P9, P10) 정의	<code>T0D0.md</code> 의 근거 문단 참고. Gauge 에서 PictogramChart / BarChart 로 가는 fallback 이 이 규칙에 걸려 있다
B-31	부모의 대분류가 Hierarchy 일 때 N3(최상위 범기)을 건너뛰고 N5(평탄화)로 갈지	<code>T0D0.md</code> 의 근거 문단 참고. <code>capability-table.md</code> 3.6.4절 N-R5 가 제안 상태다
B-32	N3 과 N4 를 반복해 최상위 컴포넌트 수가 늘어날 때 상한을 둘지	<code>T0D0.md</code> 의 근거 문단 참고. 규약 8.6절이 최상위 원소 수에 상한을 두지 않았다
B-33	그 카테고리가 읽지 않는 내용 필드 (<code>List</code> 의 <code>avatar</code> 등)를 <code>info</code> 로 셀지 (2026-08-29 신설)	지금은 아무 등급도 안 붙는다. 같은 상황의 <code>params</code> 는 <code>warning</code> 이다. 갈리는 근거는 "사라지느냐"인데(P2 는 버리고 P3 은 보존), 사라지지 않는다고 결함이 아닌 것은 아니다

C절. 2단계 배포 (3건)

#	무엇을 정해야 하나	누가, 어떤 근거로
C-1	모듈 버전 2.0(semantic) BE 정책 협의	정원석 · 박지우
C-2	semantic tag 데이터 흐름 다시 확인하고 Slack 공유	정원석
C-7	증량분 반영 후 README.md 3절 컴포넌트 분포 재측정	정원석

D절. 스퀘드 차원에서 정할 것 (3건)

#	무엇을 정해야 하나	누가, 어떤 근거로
D-4	structure → 검색 질의 변환기를 누가 만들지	정원석, 이석 / 미정
D-6	용어를 semantic tag 에서 Component Visual Category 로 바꿀지	정원석 / 미정
D-8	이중중첩 템플릿을 검색 폴에 되돌릴지 (D-7 부분 철회)	이석, 김수현, 정원석 / TODO.md 의 근거 문단 참고. 복합 구조 순서의 N1 과 N2 가 이 결정에 걸려 있다

E절. 7/24 회의에서 새로 받은 것 (2건)

#	무엇을 정해야 하나	누가, 어떤 근거로
E-1	시멘틱 컴포넌트 정의(규약)와 '문제 → 인과 분해 → 해결 방법 → 목표'의 링크를 문서로 정리해 공유	정원석
E-3	유효 컴포넌트 응집성 게이트와 node 건을 별도 트랙으로 잡기	정원석

F절. 마감 후로 미룬 것 (사후 검증) (2건)

#	무엇을 정해야 하나	누가, 어떤 근거로
F-1	YAML 과 KDL 과 JSON 의 생성 안정성 실측	미정
F-2	생성기에 silent corruption 지표를 처음부터 내장	미정

I절. 내용 대치 contract (9건)

#	무엇을 정해야 하나	누가, 어떤 근거로
I-2	세 가지를 하나로 정렬 → MOR-1933	류민, 신영빈, 이석
I-4	구간 ①의 속성을 comp_signs 로 넘기는 컬럼과 경로 정의 → MOR-1935	정원석, 이석, 이강윤
I-5	comp-sign 실물을 capability-table.md 에 반영 (MOR-1925 와 순서 조율)	정원석
I-6	rankCompCandidates 위계 가중치 A/B → MOR-1936	정원석, 이석
I-7	cross-type property projection (T-8) 규칙 초안. structure P1~P8과 구분	정원석, blueprint T-8

#	무엇을 정해야 하나	누가, 어떤 근거로
I-8	PoC 평가 pool 고정 및 사전 적재: 같은 DO subset에 comp-sign signature(T-2) + I.I.S. 경계(T-4) + 속성(T-7). pool 정의 스크립트·적재 상태 문서화	정원석
I-9	T-8 property 기반 대치 PoC: I-8·I-7 선행. 5단계 경로를 service.ts / 대치 비교 UI에 3번째 열로 연동	정원석
I-10	속성 골든셋 사람 확정 UI: 경계 골든셋과 같은 모양으로 properties / preannotation_properties 컬럼을 더 하고 편집기에 패널을 엮는다 → plan/iis-property-annotation-ui-blueprint.md	오너 미정
I-11	복합 구조 fallback 오케스트레이션: 질의가 중첩·병치인 것을 인식하고 N1~N5 를 돌리며 조각마다 검색을 호출하고 결과를 다시 조립하는 코드	정원석
I-14	type 필수 개정 코드 반영(2026-08-29 신설): zod 스키마와 JSON Schema 의 required , 생성기 프롬프트의 필수 명사와 "못 고르면 중단 셋 중 하나" 지시, query.ts 의 type 없을 때 구제 분기 정리	정원석
I-13	복잡도 레벨 토글(2026-08-29 신설): 생성기 프롬프트의 위상 지시를 레벨 단위 블록으로 쪼개고(병치=L2, 포함=L3), 프롬프트 조립이 복잡도 레벨을 인자로 받게 하고, 산출물의 실제 레벨(entity.components[] 사용 여부)이 선언한 레벨과 어긋나지 않는지 검사한다	정원석

J절. 생성 모델 파인튜닝 (8건)

#	무엇을 정해야 하나	누가, 어떤 근거로
J-1	평가셋을 38 중분류로 확장 → MOR-1928	없음. 분할 찌꺼기를 먼저 견어내야 한다 (아래)
J-2	few-shot 기준선 측정 (튜닝 없이 어디까지 되나)	J-1, 스키마 확정
J-3	Steps 역생성 파일럿	스키마 확정, CORE/DESC/DECO/OUT 정의 확인 (I절)
J-4	round-trip 검증 파이프라인	J-3
J-5	카테고리 확대 (ListingTable, ComparisonTable)	RLSC 대응 확인
J-6	라벨링 항목과 지침 확정	B-1, B-2 등급 논의, importance 필드 결정
J-7	SFT	J-3, J-4
J-8	선호 학습	J-6, J-7

J. 자기정합 검사 14가지 전량

자동 생성 도구를 저장소에 두지 않는다. 규약 문서, capability.json , 스키마 파일, YAML 예시가 서로 어긋나지 않는지는 작업할 때 스크립트를 짜서 돌린다. 2026-08-04 에 아래 14가지를 전부 돌려 통과시킨 판이 지금 파일이다. 원문은 spec/schema/README.md 다.

#	무엇을 보나
S1	규약 2.2절 JSON Schema 가 draft 2020-12 로 유효한가
S2	visual-summary-0.5.schema.json 이 그 2.2절과 같은 내용인가
S3	semanticCategory enum 이 대분류 21 + 결합 41 = 62 이고 중복이 없는가
E1	문서의 전체 문서 YAML 예시가 전량 스키마를 통과하는가 (2026-08-04 기준 37건)

#	무엇을 보나
E2	안티패턴 절의 예시는 스키마가 거부하는가
C1	capability 2절 41종이 스키마 결합값 41종과 같고 번호가 1부터 41까지 연속인가
C2	<code>capability.json</code> 이 capability 2절과 3.2절과 일치하는가
C3	예시가 자기가 선언한 카테고리의 성립 조건을 채우는가
F1	3.2절 후속 목록의 도착이 41종 안에 있는가
F2	어느 행도 대체 후보를 셋 이상 두지 않는가 (4절 두 칸 상한)
F3	모든 행이 마지막 fallback 셋 중 하나로 끝나는가
F4	3.5절 엣지 표가 3.2절에서 유도한 그래프 엣지와 같고 중복이 없는가
F5	3.5절의 규칙이 <code>capability.json</code> 에서 계산한 것과 같은가
N1	본문이 말하는 수(41, 62, 75, 아홉)가 표의 실제 수와 맞는가

F5 가 이 검사를 살아 있게 한다. `capability.json` 의 `reads` 로 엣지마다의 규칙(P1~P8)을 다시 계산해 표와 대조한다. 선택 내용과 속성의 구분이 틀리면 이 검사가 걸린다.

이 검사에는 남은 값이 섞여 있다. S3, C1, F1 이 41종을 세고 F2 가 폐지된 두 칸 상한을 검사하고 F4 와 N1 이 엣지 75개를 센다. 어휘가 45종으로 늘고 상한이 폐지되고 엣지가 131개가 된 것이 그 뒤의 일이다. **마지막으로 전량을 돌린 것이 2026-08-04 이고 2026-08-06, 2026-08-21, 2026-08-28 개정이 검사 없이 들어갔다.** 이것이 부록 F 1번과 2번이 가리키는 격차의 원인이다.

코드 쪽에는 같은 대조가 살아 있다. `src/lib/visual-summary/capability.test.ts` 가 `spec/schema/capability.json` 과 45종 전량을 대조한다. 다만 `docs/` 가 비공개 submodule 이라 CI 체크아웃에 없어서 로컬에서만 돌고 CI 에서는 건너뛰다.

K. 꾸밈구조 형 정의와 해석 예시

2절이 꾸밈구조의 설계 근거와 축, 값, 층을 설명한다. 이 부록은 실물 형과 해석 예시를 적는다. 원문은 `design/decoration-model.md` 5절이고 확정 표기는 규약 2.1절, 2.2절, 2.7절이 갖는다.

붙는 자리는 `structure` 의 형제다. `params` 는 카테고리별 표현 선택이고 꾸밈구조는 카테고리과 무관하다. `structure` 는 내용이고 꾸밈구조는 내용이 아니다. `page` 수준에도 같은 키를 연다.

```
// 꾸밈 속에 담기는 값. 이 셋이 전부다
export type Preference = 'prefer' | 'indifferent' | 'avoid';

// 꾸밈구조. 모양은 하나다. 축 하나가 RLSC role 하나에 대응한다
// minimal 이나 rich 같은 프리셋 문자열은 받지 않는다
export interface Decoration {
  ornament?: Preference; // Element.Decoration, LayoutContainer.Decoration
  background?: Preference; // Element.Background, LayoutContainer.Background
}

export interface VisualSummary {
  version: '0.5';
  title: string;
  components: Component[];
  relations?: Relation[];
  decoration?: Decoration; // page 수준. 아래 컴포넌트의 기본값이 된다
}

export interface Component {
  id?: string;
  type?: SemanticCategory;
  alternatives?: SemanticCategory[];
  params?: Record<string, unknown>;
  structure: Structure;
  decoration?: Decoration; // 이 컴포넌트에만 적용된다
}
```

JSON Schema 조각은 `additionalProperties: false` 를 쓴다. **없는 축 이름을 적으면 거부된다.** 축을 늘릴 때 스키마를 함께 고쳐야 한다는 뜻이고 의도한 것이다. 오타로 적은 축이 조용히 무시되면 적었는데 적용되지 않는 사고가 난다.

```
{
  "decoration": {
    "type": "object",
    "additionalProperties": false,
    "minProperties": 1,
    "properties": {
      "ornament": { "$ref": "#/$defs/preference" },
      "background": { "$ref": "#/$defs/preference" }
    }
  },
  "preference": {
    "type": "string",
    "enum": ["prefer", "indifferent", "avoid"],
    "default": "indifferent"
  }
}
```

예시 1. 아무것도 안 적은 경우다. 1단계의 기본 동작이다.

```
version: '0.5'
title: 분기 실적
components:
  - type: Enumeration.List
    structure:
      entities: [ { label: 신규 계약, items: [12건] } ]
```

decoration 이 없으므로 시스템 전역 기본값이 그대로 적용된다. 전역이 '꾸밈 많은 것 우선'이면 두 축이 prefer 로 해석된다.

예시 2. page 값을 컴포넌트가 축 단위로 덮는다.

```
decoration:                                # page 수준. 회의 용어로는 rich 에 해당한다
  ornament: prefer
  background: prefer
components:
  - type: Enumeration.List                # 물려받는다. 둘 다 prefer
    structure: { entities: [ ] }
  - type: ProcessFlow.Steps
    decoration:
      background: avoid                  # 이 축만 덮는다
    structure: { entities: [ ] }
```

두 번째 컴포넌트의 해석 결과는 ornament: prefer (page 에서 물려받음)와 background: avoid (자기 값)다. ornament 를 다시 적지 않아도 유지된다. 병합이 축 단위인 것이 이 예시의 요점이다. 층 전체를 통째로 갈아치우면 이 모양을 표현할 수 없다.

예시 3. 물려받은 값을 취소한다.

```
decoration:                                # 회의 용어로는 minimal
  ornament: avoid
  background: avoid
components:
  - type: Enumeration.List
    decoration:
      background: indifferent            # 배경만 중립으로 되돌린다
    structure: { entities: [ ] }
```

결과는 ornament: avoid, background: indifferent 다. indifferent 를 명시적 값으로 둔 이유가 이 자리다. 축을 빼면 page 의 avoid 를 그대로 물려받는다.

예시 4. 카테고리 기본값과 만난다.

```
components:
  - type: Enumeration.List                # 카테고리 기본값이 background: prefer 라고 가정
    structure: { entities: [ ] }
  - type: Enumeration.List
    decoration: { background: avoid }
    structure: { entities: [ ] }
```

첫째는 카테고리 기본값을 받아 background: prefer 이고, 둘째는 자기 값이 있으므로 avoid 를 우선한다. 45종의 카테고리 기본값은 아직 안 채웠다.

위 예시는 "적을 수 있다"를 보인 것이고 "생성기가 이렇게 쓴다"가 아니다. 1단계에서 이 필드는 optional 이고 생성기가 채우지 않는다. 값은 시스템 전역 기본값에서 온다. 선례가 둘 있다. entity.components[] 와 page 수준 relations 가 같은 방식으로 열려 있다.

부록: 카테고리 45종 데이터 구조 예시

이 부록을 읽는 법

앞 부록이 필드 목록을 표로 적었다면 이 부록은 같은 것을 실물 YAML 로 적는다. 45종 각각에 예시를 하나씩 두고, 그 카테고리가 선언한 `params` 를 전부 채우고 `structure` 도 선택 내용까지 채운 모양으로 적었다. 최소 예시가 아니라 최대 예시다. 실제 생성물은 이보다 훨씬 짧고, 선택 값을 생략하면 규약이 정한 기본값으로 그려진다.

번호는 앞 부록 B절, C절과 같고 `spec/capability-table.md` 2절의 행 번호와도 같다.

`params` 를 못 채운 자리가 20종이다. 45종 중 25종만 `params` 키를 선언한다. 나머지 20종(`params` 를 선언하지 않는 16종과 페이지 역할 4종)은 채울 키가 없어서 예시에도 `params` 가 없다. 선언하지 않은 키를 적으면 그 카테고리에서는 뜻이 없고 fallback 때 P2 로 버려진다.

모든 예시는 검증을 통과한 값이다. `spec/schema-v0.5.md` 2.2절의 JSON Schema 로 스키마 검증을 돌렸고, `spec/schema/capability.json` 과 대조해 개수 조건, dataset 모양, 선언한 `params` 키 전량 포함 셋을 함께 확인했다.

찾는 것	절
나열과 비교 (1번부터 8번)	1
순서와 흐름 (9번부터 16번)	2
계층과 범위 (17번부터 22번)	3
수치 (23번부터 34번)	4
텍스트와 사람 (35번부터 41번)	5
페이지 역할 (42번부터 45번)	6
컴포넌트가 둘 이상일 때: 병치와 포함	7

예시는 `components[]` 안의 컴포넌트 한 벌만 적는다. 실제 문서는 `version`, `title` 이 최상위에 붙고 `components[]` 가 배열이다. 7절이 그 온전한 모양을 보여준다.

1. 나열과 비교

1. `Enumeration.List` 선언한 `params` 가 없다. 선택 내용은 `items` 하나다.

```
- type: Enumeration.List
  structure:
    entities:
      - label: 콘텐츠 제작
        items: ["템플릿 8,200종", 월 신규 300종]
      - label: 협업
        items: [실시간 공동 편집, 댓글과 승인]
      - label: 배포
        items: [웹 링크, PDF, 이미지]
```

2. `Enumeration.ListingTable` 행이 항목이고 열이 그 항목의 속성이다.

```

- type: Enumeration.ListingTable
  params:
    header: true
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 플랜, role: category }
        - { label: 월 요금, role: value, unit: 원 }
        - { label: 저장 용량, role: value, unit: GB }
      rows:
        - [무료, 0, 2]
        - [프로, 12900, 100]
        - [팀, 24900, 1000]

```

3. **AttributeComparison.Versus** 정확히 2개다. `compares` 관계가 가운데 대비 문구를 담는다.

```

- type: AttributeComparison.Versus
  structure:
    entities:
      - label: 사내 구축
        items: [초기 비용 높음, 데이터 통제권 확보, 운영 인력 필요]
      - label: 클라우드 구독
        items: [초기 비용 낮음, 확장이 즉시, 월 비용 지속]
    relations:
      - type: compares
        of: [사내 구축, 클라우드 구독]
        desc: 3년 총소유비용은 사용 규모 200석을 넘어서면 역전된다

```

4. **AttributeComparison.ComparisonTable** 행이 지표이고 열이 대상이다.

```

- type: AttributeComparison.ComparisonTable
  params:
    header: true
    axis: true
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 평가 항목, role: category }
        - { label: A 사, role: value }
        - { label: B 사, role: value }
        - { label: C 사, role: value }
      rows:
        - [도입 기간, 2주, 6주, 4주]
        - [연 비용, 1200, 2400, 1800]
        - [한국어 지원, 있음, 없음, 있음]

```

4-1. 병합 헤더가 있는 표는 `matrix` 대신 `entities` 와 `columns` 를 쓴다. `ListingTable` 과 `ComparisonTable` 두 카테고리 만 이 경로를 받는다(2026-08-06 추가). `entity.label` 이 첫 열이고 `entity.values[]` 가 나머지 열이며, 상위 범주가 하위 행을 거느리는 병합은 `entity.children[]` 로 적는다.

```
- type: AttributeComparison.ComparisonTable
params:
  header: true
  axis: true
structure:
  columns:
    - { label: 항목 }
    - { label: 1Q }
    - { label: 2Q }
    - { label: 3Q }
  entities:
    - label: PROJECT 01
      children:
        - { label: 성장률, values: ["35%", "67%", "55%"] }
        - { label: 매출, values: [14000, 25000, 18000] }
    - label: PROJECT 02
      children:
        - { label: 성장률, values: ["12%", "20%", "31%"] }
        - { label: 매출, values: [8000, 9500, 12400] }
```

5. RelationalComparison.SWOT 정확히 4개다.

```
- type: RelationalComparison.SWOT
params:
  order: [strength, weakness, opportunity, threat]
structure:
  entities:
    - label: 강점
      items: [국내 점유율 1위, 템플릿 자산 8,200종]
    - label: 약점
      items: [해외 인지도 낮음, 영어 콘텐츠 부족]
    - label: 기회
      items: [생성 AI 수요 확대, 원격 근무 정착]
    - label: 위협
      items: [글로벌 경쟁사 가격 인하, 오픈소스 대안 등장]
```

6. RelationalComparison.VennDiagram 2개 이상 3개 이하다(실측). intersects 관계가 교집합 영역의 문구를 담는다.

```
- type: RelationalComparison.VennDiagram
structure:
  entities:
    - label: 전통
      items: [유산, 정체성, 신뢰]
    - label: 변화
      items: [혁신, 적응, 미래]
  relations:
    - type: intersects
      of: [전통, 변화]
      desc: 두 요소를 조화롭게 관리하는 것이 장수기업 HR 의 핵심이다
```

7. RelationalComparison.Quadrant matrix 가 필수이고 항목마다 x 와 y 가 있어야 한다.

```
- type: RelationalComparison.Quadrant
  params:
    axisLabels: { x: 시장 성장률, y: 상대적 점유율 }
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 제품, role: category }
        - { label: 성장률, role: x, unit: "%" }
        - { label: 점유율, role: y, unit: "%" }
      rows:
        - [디자인 편집기, 24, 41]
        - [프레젠테이션, 31, 18]
        - [인쇄 주문, 4, 52]
        - [영상 편집, 38, 6]
```

8. **Transformation.Transformation** 정확히 2개이고 앞이 전, 뒤가 후다.

```
- type: Transformation.Transformation
  params:
    direction: forward
  structure:
    entities:
      - label: AS-IS
        items: [수작업 배치, 페이지당 12분, 오타자 검수 별도]
      - label: TO-BE
        items: [자동 배치, 페이지당 40초, 검수 규칙 내장]
```

2. 순서와 흐름

9. **ProcessFlow.Steps** 2개 이상이다.

```
- type: ProcessFlow.Steps
  params:
    sequence: [1, 2, 3, 4]
    marker: number
  structure:
    entities:
      - label: 요구 정리
        items: [담당자 인터뷰, 기존 문서 수집]
      - label: 초안 생성
        items: [자동 배치, 이미지 후보 제시]
      - label: 검수
        items: [사내 리뷰, 수정 반영]
      - label: 배포
        items: [링크 공유, PDF 내보내기]
```

10. **TimelineSchedule.Timeline** 2개 이상 15개 이하다(실측). **period** 가 선택 내용이다.

```
- type: TimelineSchedule.Timeline
  params:
    orientation: horizontal
  structure:
    entities:
      - { label: 창업, period: "2010", items: [사무실 1곳, 구성원 4명] }
      - { label: 첫 유료 고객, period: "2013", items: [연 매출 3억원] }
      - { label: 해외 진출, period: "2018", items: [일본, 대만 동시 출시] }
      - { label: 사용자 1천만, period: "2024", items: [누적 가입 기준] }
```

11. Progression.Staircase 2개 이상 8개 이하다(실측).

```
- type: Progression.Staircase
  params:
    direction: ascending
  structure:
    entities:
      - label: 인지
        items: [광고 노출, 검색 유입]
      - label: 체험
        items: [무료 가입, 템플릿 1건 사용]
      - label: 구독
        items: [프로 전환]
      - label: 확산
        items: [팀 초대, 사내 표준 채택]
```

12. Cycle.Cycle 2개 이상 8개 이하다(실측).

```
- type: Cycle.Cycle
  params:
    center: 품질 개선 루프
    rotation: clockwise
  structure:
    entities:
      - label: 측정
        items: [지표 수집]
      - label: 분석
        items: [원인 후보 도출]
      - label: 개선
        items: [배포]
      - label: 검증
        items: [재측정]
```

13. LogicFlow.FlowChart 선언한 params 가 없다. 분기 라벨이 선택 내용이다.

```
- type: LogicFlow.FlowChart
  structure:
    entities:
      - label: 결제 요청 수신
        items: [카드 정보 검증]
      - label: 한도 확인
        items: ["초과: 대체 수단 안내", "정상: 승인 진행"]
      - label: 승인
        items: [영수증 발행]
      - label: 대체 수단 안내
        items: [계좌 이체, 다른 카드]
```

14. **Conversion.Funnel** `matrix` 는 선택이고, 넣으면 단계마다 수치가 붙는다.

```
- type: Conversion.Funnel
  params:
    conversionRate: true
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 단계, role: category }
        - { label: 인원, role: value, unit: 명 }
      rows:
        - [방문, 48000]
        - [가입, 12400]
        - [무료 체험, 5100]
        - [유료 전환, 860]
```

15. **TimelineSchedule.GanttChart** `matrix` 가 필수이고 작업마다 시작과 끝이 있어야 한다.

```
- type: TimelineSchedule.GanttChart
  params:
    unit: month
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 작업, role: category }
        - { label: 시작, role: start }
        - { label: 종료, role: end }
      rows:
        - [요구 정리, "2026-09-01", "2026-09-20"]
        - [설계, "2026-09-15", "2026-10-31"]
        - [구현, "2026-10-20", "2027-01-15"]
        - [검증, "2027-01-05", "2027-02-10"]
```

16. **TimelineSchedule.Calendar** `matrix` 가 필수이고 날짜와 일정이 있어야 한다.

```
- type: TimelineSchedule.Calendar
  params:
    weekStart: Mon
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 일정, role: category }
        - { label: 날짜, role: date }
      rows:
        - [스프린트 시작, "2026-09-01"]
        - [중간 점검, "2026-09-10"]
        - [코드 동결, "2026-09-24"]
        - [배포, "2026-09-28"]
```

3. 계층과 범위

17. **Hierarchy.HierarchicalTree** `children` 이 필수다. 깊이 상한은 없다.

```
- type: Hierarchy.HierarchicalTree
  params:
    orientation: vertical
  structure:
    entities:
      - label: 대표이사
        children:
          - label: 제품본부
            children:
              - { label: 디자인팀 }
              - { label: 개발팀 }
          - label: 사업본부
            children:
              - { label: 영업팀 }
              - { label: 마케팅팀 }
```

18. **Hierarchy.Pyramid** 3개 이상이다. **weight** 가 층 높이를 정하는 선택 내용이다.

```
- type: Hierarchy.Pyramid
  params:
    apex: last
  structure:
    entities:
      - { label: 생리적 욕구, weight: 5, items: [수면, 식사] }
      - { label: 안전 욕구, weight: 4, items: [주거, 고용] }
      - { label: 사회적 욕구, weight: 3, items: [소속, 우정] }
      - { label: 존경 욕구, weight: 2, items: [인정, 지위] }
      - { label: 자아실현, weight: 1, items: [성장, 창조] }
```

19. **ScopeCoverage.NestedScope** **children** 이 필수이고 안쪽이 좁은 범위다.

```
- type: ScopeCoverage.NestedScope
  params:
    shape: circle
  structure:
    entities:
      - label: 국내 시장
        items: [TAM 연 12조원]
        children:
          - label: 유효 시장
            items: [SAM 연 2.4조원]
            children:
              - { label: 수익 시장, items: ["SOM 연 3,100억원"] }
      - label: 해외 시장
        items: [TAM 연 41조원]
        children:
          - label: 유효 시장
            items: [SAM 연 5.2조원]
            children:
              - { label: 수익 시장, items: ["SOM 연 2,900억원"] }
```

20. **ReasoningStructure.Fishbone** 선언한 **params** 가 없다. **children** 이原因的 하위 갈래다.

```
- type: ReasoningStructure.Fishbone
  structure:
    entities:
      - label: 사람
        children:
          - { label: 신규 인력 교육 부족 }
          - { label: 검수 담당 부재 }
      - label: 공정
        children:
          - { label: 배치 규칙이 문서로만 존재 }
          - { label: 승인 단계 누락 }
      - label: 도구
        children:
          - { label: 폰트 동기화 실패 }
```

21. **ReasoningStructure.CombinedResult** 결과 라벨이 선택 내용이다.

```
- type: ReasoningStructure.CombinedResult
  structure:
    entities:
      - label: 자체 템플릿 자산
        items: ["8,200종"]
      - label: 생성 모델
        items: [사내 파인튜닝]
      - label: 배치 엔진
        items: [규칙 기반 검증]
      - label: 결과
        items: [입력 한 줄에서 완성 발표자료 10쪽]
```

22. **ReasoningStructure.ProblemSolution** label 이 문제, items 가 해결이다.

```
- type: ReasoningStructure.ProblemSolution
  structure:
    entities:
      - label: 초안 작성에 시간이 오래 걸린다
        items: ["해결: 자동 생성", "기대: 페이지당 12분에서 40초"]
      - label: 사내 표기 규칙이 지켜지지 않는다
        items: ["해결: 규칙을 템플릿에 내장", "기대: 검수 반려 60% 감소"]
```

4. 수치

23. **KPIKeyMetric.BigNumber** value 가 필수다. dataset: measure 도 선택으로 쓸 수 있지만, 항목이 여럿이면 entity 쪽에 적는다. delta (증감)는 entity 에 적을 수 없다. capability.json 은 BigNumber 의 선택 내용에 delta 를 넣어 뒀는데 스키마의 entity 에는 그 필드가 없고 measure 에만 있다. 그래서 항목이 여럿이면 증감을 적을 자리가 없고, 증감을 적으려면 항목 하나짜리 measure 로 쓴다. 앞 부록 F절 5번이다.

```
- type: KPIKeyMetric.BigNumber
  structure:
    entities:
      - { label: 연 매출, value: 152, unit: 억원, emphasis: true }
      - { label: 신규 고객, value: 320, unit: 건 }
      - { label: 재구매율, value: 68, unit: "%" }
```

24. **KPIKeyMetric.Gauge** measure.value 가 필수다. 선언한 params 는 없고 선택 값은 전부 measure 안에 있다.

```
- type: KPIKeyMetric.Gauge
structure:
  entities:
    - { label: 연간 목표 달성률 }
  dataset:
    shape: measure
    value: 78
    unit: "%"
    min: 0
    max: 100
    target: 90
    delta: 12
    thresholds:
      - { at: 50, label: 주의, state: warn }
      - { at: 80, label: 안정, state: safe }
```

25. **MagnitudeRanking.BarChart** `matrix` 가 필수다.

```
- type: MagnitudeRanking.BarChart
params:
  orientation: vertical
  sort: desc
structure:
  dataset:
    shape: matrix
    columns:
      - { label: 채널, role: category }
      - { label: 매출, role: value, unit: 억원 }
    rows:
      - [직영몰, 62]
      - [오픈마켓, 41]
      - [대리점, 28]
      - [해외, 21]
```

26. **ProportionComposition.StackedBarChart** 계열이 2개 이상이어야 한다. `normalize` 는 `false` 고정이다.

```
- type: ProportionComposition.StackedBarChart
params:
  orientation: vertical
  normalize: false
structure:
  dataset:
    shape: matrix
    columns:
      - { label: 분기, role: category }
      - { label: 구독, role: value, unit: 억원 }
      - { label: 광고, role: value, unit: 억원 }
      - { label: 기타, role: value, unit: 억원 }
    rows:
      - [1Q, 28, 9, 3]
      - [2Q, 31, 11, 4]
      - [3Q, 35, 10, 4]
      - [4Q, 40, 14, 5]
```

27. **ProportionComposition.PieChart** 조각이 2개 이상이어야 한다.

```
- type: ProportionComposition.PieChart
  params:
    startAngle: 0
    labelPosition: outside
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 기기, role: category }
        - { label: 비중, role: value, unit: "%" }
      rows:
        - [데스크톱, 54]
        - [모바일, 38]
        - [태블릿, 8]
```

28. **ProportionComposition.DonutChart** `centerLabel` 이 가운데 공간을 쓴다.

```
- type: ProportionComposition.DonutChart
  params:
    centerLabel: 총 12,400명
    startAngle: 90
    labelPosition: outside
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 플랜, role: category }
        - { label: 인원, role: value, unit: 명 }
      rows:
        - [무료, 8200]
        - [프로, 3100]
        - [팀, 1100]
```

29. **ProportionComposition.TreeMap** 선언한 `params` 가 없다. `matrix` 가 필수이고, 2단으로 그리려면 `entities` 쪽에 `children` 을 함께 적는다.

```
- type: ProportionComposition.TreeMap
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 사업부, role: category }
        - { label: 매출, role: value, unit: 억원 }
      rows:
        - [디자인, 62]
        - [프레젠테이션, 41]
        - [인쇄, 28]
        - [영상, 21]
```

30. **ProportionComposition.PictogramChart** 아이콘 하나가 나타내는 수량을 `unitPerIcon` 이 정한다.

```
- type: ProportionComposition.PictogramChart
params:
  icon: person
  unitPerIcon: 100
structure:
  dataset:
    shape: matrix
    columns:
      - { label: 부서, role: category }
      - { label: 인원, role: value, unit: 명 }
    rows:
      - [개발, 800]
      - [디자인, 400]
      - [영업, 300]
```

31. **Trend.LineAreaChart** 계열 1개 이상, 축 2개 이상이다. **fill** 이 참이면 면적으로 그린다.

```
- type: Trend.LineAreaChart
params:
  fill: true
structure:
  dataset:
    shape: matrix
    columns:
      - { label: 월, role: category }
      - { label: 방문자, role: value, unit: 명 }
    rows:
      - [1월, 42000]
      - [2월, 45800]
      - [3월, 51200]
      - [4월, 49700]
      - [5월, 58300]
```

32. **Trend.ComboChart** 계열 **type** 이 2종 이상이거나 **axis** 가 2종 이상이어야 성립한다. 이 둘은 **params** 가 아니라 **dataset.columns[]** 에 적는다.

```
- type: Trend.ComboChart
structure:
  dataset:
    shape: matrix
    columns:
      - { label: 분기, role: category }
      - { label: 매출, role: value, unit: 억원, type: bar, axis: left }
      - { label: 영업이익률, role: value, unit: "%", type: line, axis: right }
    rows:
      - [1Q, 40, 12]
      - [2Q, 46, 14]
      - [3Q, 49, 13]
      - [4Q, 59, 17]
```

33. **AttributeComparison.RadarChart** 축이 3개 이상이어야 한다.

```

- type: AttributeComparison.RadarChart
  params:
    maxValue: 100
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 평가 축, role: category }
        - { label: 자사, role: value }
        - { label: 경쟁사, role: value }
      rows:
        - [속도, 82, 64]
        - [정확도, 76, 81]
        - [사용성, 91, 70]
        - [가격, 68, 88]
        - [지원, 74, 59]

```

34. **RelationalComparison.BubbleChart** 항목마다 x, y, 크기가 있어야 한다.

```

- type: RelationalComparison.BubbleChart
  params:
    sizeScale: linear
  structure:
    dataset:
      shape: matrix
      columns:
        - { label: 시장, role: category }
        - { label: 성장률, role: x, unit: "%" }
        - { label: 수익률, role: y, unit: "%" }
        - { label: 규모, role: size, unit: 억원 }
      rows:
        - [국내, 12, 21, 620]
        - [일본, 24, 14, 310]
        - [동남아, 38, 6, 140]

```

5. 텍스트와 사람

35. **TextBasedCommunication.HighlightedMessage** 정확히 1개다. **items** 가 선택이다.

```

- type: TextBasedCommunication.HighlightedMessage
  structure:
    entities:
      - label: 초안 작성 시간을 페이지당 12분에서 40초로 줄인다
        items: [2026년 3분기 사내 실측 기준]

```

36. **TextBasedCommunication.LongFormTextLayout** **title** 이 유일한 **params** 다.

```
- type: TextBasedCommunication.LongFormTextLayout
  params:
    title: 우리가 이 문제를 푸는 이유
  structure:
    entities:
      - label: 발표자로 제작은 여전히 손으로 하는 일이다
        items:
          - 사내 설문에서 응답자 412명 중 68%가 초안 배치에 가장 오래 걸린다고 답했다.
          - 같은 응답자의 74%는 사내 표기 규칙을 매번 다시 확인한다고 답했다.
          - 규칙을 템플릿에 내장하면 이 두 시간이 함께 줄어든다.
```

37. `QnACard.QnACard` `label` 이 질문, `items` 가 답변이다.

```
- type: QnACard.QnACard
  structure:
    entities:
      - label: 무료 플랜에서도 PDF 로 내보낼 수 있나
        items: [워터마크가 붙은 상태로 가능하다]
      - label: 팀 플랜은 몇 명부터 쓸 수 있나
        items: [3명부터이고 좌석 단위로 늘린다]
```

38. `Utterance.DialogueChat` 2개 이상이다. `speaker` 가 선택 내용이고, 없으면 번갈아 배치한다.

```
- type: Utterance.DialogueChat
  structure:
    entities:
      - { label: 이번 분기 자료 초안 언제 되나요, speaker: 팀장 }
      - { label: 오늘 오후에 자동 생성으로 1차 뽑아 두겠습니다, speaker: 담당자 }
      - { label: 표기 규칙은 반영되나요, speaker: 팀장 }
      - { label: 템플릿에 들어 있어서 자동으로 맞춰집니다, speaker: 담당자 }
```

39. `Utterance.QuoteTestimonial` `author` 와 `source` 가 선택 내용이다.

```
- type: Utterance.QuoteTestimonial
  structure:
    entities:
      - label: 예전에는 만나질 걸리던 월간 보고서를 이제 회의 직전에 만든다
        author: 김00
        source: 제조업 기획팀, 2026-07 인터뷰
      - label: 표기 규칙을 외우지 않아도 되는 것이 가장 크다
        author: 이00
        source: 금융업 마케팅팀, 2026-07 인터뷰
```

40. `PortraitProfile.ProfileCard` `avatar` 와 `organization` 이 선택 내용이다.

```
- type: PortraitProfile.ProfileCard
  structure:
    entities:
      - label: 정원석
        items: [Visual Summary 요약, 검색 대치]
        avatar: https://cdn.example.com/avatar/wsjeong.png
        organization: MORDOR 스쿼드
      - label: 황현진
        items: [의미 추출 모듈]
        avatar: https://cdn.example.com/avatar/hjhwang.png
        organization: MORDOR 스쿼드
```

41. `PortraitProfile.ResumeCV` `period` 와 `children` 이 선택 내용이다.

```
- type: PortraitProfile.ResumeCV
  structure:
    entities:
      - label: 경력
        children:
          - { label: 미리디 제품개발, period: "2021-2026", items: [발표자료 자동 생성] }
          - { label: 이전 직장 플랫폼팀, period: "2017-2021", items: [검색 색인 운영] }
      - label: 학력
        children:
          - { label: 00대학교 컴퓨터공학, period: "2013-2017" }
```

6. 페이지 역할

네 종 모두 선언한 `params` 가 없고 `dataset` 도 쓰지 않는다. 규약이 `entities` 나 `dataset` 중 하나를 요구하므로 `entities` 로 적는다. 성립 조건은 2026-08-27 에 채워졌다.

아래 넷은 2026-08-29 부터 스키마 검증을 통과한다. 어휘 편입은 2026-08-06 에 정해졌는데 규약 `semanticCategory` enum 만 안 따라오고 있었고(앞 부록 F절 2번), 0.5.3 에서 결합값 45종으로 맞추며 닫혔다.

42. `PageNavigation.Opening` 정확히 1개이고, 제목과 함께 발표자, 날짜, 소속 중 하나 이상이 온다.

```
- type: PageNavigation.Opening
  structure:
    entities:
      - label: 2026년 하반기 제품 로드맵
        items: [부제 - 생성에서 배치까지, 정원석, "2026-09-01", MORDOR 스쿼드]
```

43. `PageNavigation.TableOfContents` 2개 이상이다. 흐름이나 단계 표현은 쓰지 않는다.

```
- type: PageNavigation.TableOfContents
  structure:
    entities:
      - { label: "01 시장 현황", items: [경쟁 구도와 수요 변화] }
      - { label: "02 제품 전략", items: [올해 집중할 셋] }
      - { label: "03 실행 계획", items: [분기별 마일스톤] }
      - { label: "04 예산", items: [항목별 배분] }
```

44. `PageNavigation.SectionDivider` 정확히 1개이고 본문 콘텐츠가 없다.

```
- type: PageNavigation.SectionDivider
  structure:
    entities:
      - label: "02 제품 전략"
        items: [올해 집중할 셋]
```

45. `PageNavigation.Ending` 정확히 1개이고 종료를 알리는 단일 메시지만 담는다.

```
- type: PageNavigation.Ending
  structure:
    entities:
      - label: 감사합니다
```

7. 병치와 포함: 이종중첩 컴포넌트

이종중첩 컴포넌트(heterogeneous nested component)는 하나의 컴포넌트 경계 안에 서로 다른 semantic category 가 함께 있는 경우다. 위상 관계(topological relation)로 셋을 구분하고, 그중 둘만 실제로 나타난다.

위상	정의	실측
포함 (Containment)	한 카테고리의 시각 영역이 다른 카테고리의 영역 안에 완전히 들어간다	예: List 항목마다 안에 BigNumber 가 들어간 모양
병치 (Juxtaposition)	서로 다른 카테고리의 영역이 겹치지 않고 나란히 놓인다	2026-08-13 표본에서 관측된 11건 전부가 이 경우였다
중첩 (Overlap)	영역이 부분적으로 겹친다	관측되지 않는다. RLSC 트리에서 한 노드는 부모 하나에만 속하므로 구조적으로 생기지 않는다

규모는 실측돼 있다. 전체 183,052개 컴포넌트에서 1,000개를 뽑아 판정한 결과 이종중첩은 11건(1.10%, 95% 신뢰구간 0.62%부터 1.96%, 전체 환산 약 2,014개)이었다. 조합은 `BigNumber + List` 4건, `HighlightedMessage + List` 2건, 나머지 다섯 조합이 각 1건이었다.

아래 예시는 전부 지금 생성되지 않는 값이다. 스키마는 `entity.components[]` 를 v0.4 때부터 열어 뒀지만 1단계 생성기는 이 필드를 채우지 않는다. 표현 가능성만 열어 두고 켜는 시점은 생성기 정책이 정한다.

아래 그림 셋을 읽는 법. 굵은 검정 테두리가 컴포넌트 경계 하나다. 파란 테두리는 그 안에서 한 카테고리가 차지하는 영역, 주황 테두리는 그와 겹치지 않고 놓인 다른 카테고리의 영역, 회색 테두리는 개별 요소다. 그림 2와 그림 3은 실측 컴포넌트의 썸네일 위에 덧그렸고, 번호는 바로 아래 표의 `structure` 자리와 대응한다.

7.1 포함 위상

부모 컴포넌트의 항목 하나하나가 자기 안에 다른 카테고리를 품는다. `entity.components[]` 는 항목 하나에 정확히 1개만 매단다. `entity` 는 자기 `label` 과 `items` 를 그대로 갖고, 그 밑에 하위 컴포넌트를 하나 엮는다.

```

version: "0.5.3"
title: 분기별 사업 현황
components:
  - type: Enumeration.List
    structure:
      entities:
        - label: 1분기
          items: [신규 계약 12건]
          components:
            - type: KPIKeyMetric.BigNumber
              structure:
                entities:
                  - { label: 매출, value: 12, unit: 억원 }
        - label: 2분기
          items: [신규 계약 15건]
          components:
            - type: KPIKeyMetric.BigNumber
              structure:
                entities:
                  - { label: 매출, value: 15, unit: 억원 }

```

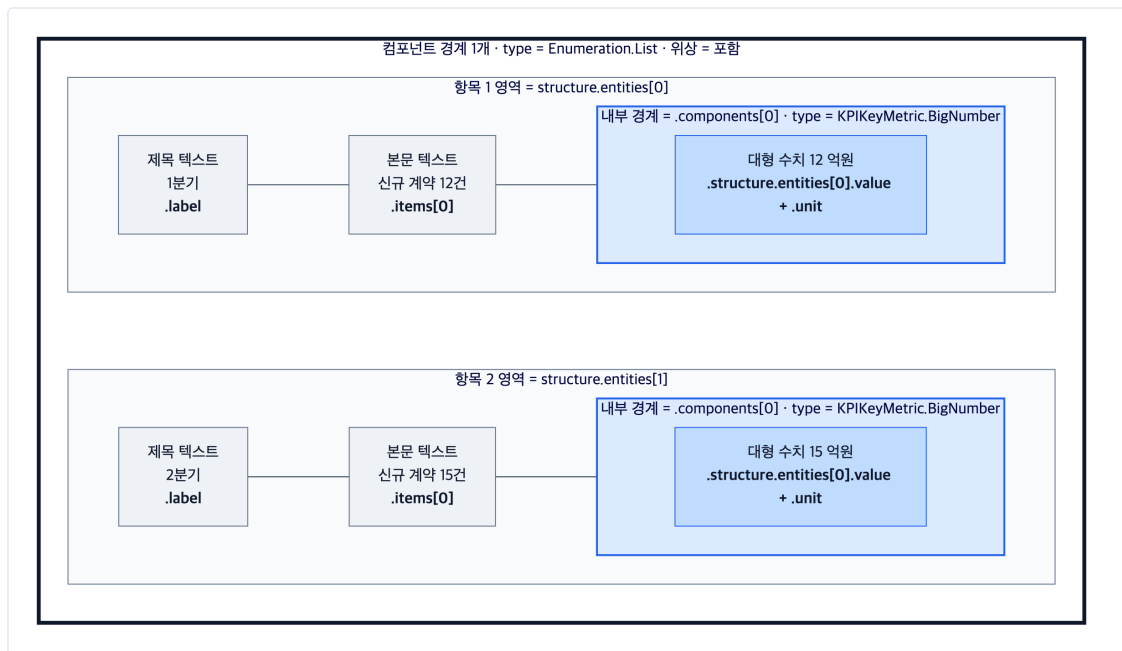


그림 1. 포함 위상. 위 YAML 을 화면 영역으로 옮긴 것이다. 바깥 경계 하나가 Enumeration.List 이고, 항목 영역 둘이 각각 structure.entities[0] 과 [1] 이다. 항목마다 제목과 본문이 label 과 items[0] 으로 들어가고, 그 안쪽에 파란 경계로 KPIKeyMetric.BigNumber 가 하나씩 들어갔어 entity.components[0] 자리를 차지한다. 이 모양은 2026-08-13 표본 11건에는 없었다. 11건이 전부 병치였기 때문이다. 그림은 규약 8.7절이 정보으로 든 예시를 옮긴 것이다.

중첩이 매달리는 자리는 entity 전체이지 entity 안의 값 필드가 아니다. label 은 항상 문자열이고 items 는 항상 문자열 배열이라서, 그 값 자리에 컴포넌트를 직접 넣으면 스키마가 거부한다. 유일한 중첩 통로는 components 라는 별도 배열 필드다.

포함이 실패했을 때 갈래가 둘이다. 이 구조를 그대로 담은 후보를 색인에서 못 찾으면 평탄화하거나 떼어낸다.

평탄화(구조 강등). 규약 1절 "중첩은 평탄화된다"가 정한 유일한 확정 기법이다. 자식의 entities 를 부모의 items 로 편다.

```

version: "0.5.3"
title: 분기별 사업 현황
components:
  - type: Enumeration.List
    structure:
      entities:
        - { label: 1분기, items: [신규 계약 12건, 매출 12억원] }
        - { label: 2분기, items: [신규 계약 15건, 매출 15억원] }

```

매출 12억원 이 큰 숫자로 강조되던 BigNumber 의 정체성을 잃고 신규 계약 12건 과 같은 글자 한 줄이 됐다. 구조는 안 잃지만 표현을 잃는 강등이다.

떼어내기(승격). 자식을 부모에서 분리해 최상위 컴포넌트로 올린다. 검색 쪽 재료(T-5 노드 태그 검색)는 이미 있고 오케스트레이션 코드가 아직 없다.

```

version: "0.5.3"
title: 분기별 사업 현황
components:
  - type: Enumeration.List
    structure:
      entities:
        - { label: 1분기, items: [신규 계약 12건] }
        - { label: 2분기, items: [신규 계약 15건] }
  - type: KPIKeyMetric.BigNumber
    structure:
      entities:
        - { label: 1분기 매출, value: 12, unit: 억원 }
        - { label: 2분기 매출, value: 15, unit: 억원 }

```

BigNumber 의 강조 표현은 살지만 "이 매출이 1분기 항목 바로 옆에 붙어 있다"는 공간적 연결이 사라진다. **둘의 순서는 2026-08-28에 정해졌다.** 둘 다 텍스트를 하나도 버리지 않는데 떼어내기가 카테고리 표현을 더 남기므로 떼어내기를 먼저 시도하고 평탄화는 마지막에만 한다(7.5절).

7.2 병치 위상

병치에는 자연스러운 모양이 둘이다. 새 필드나 wrapper 는 필요 없다.

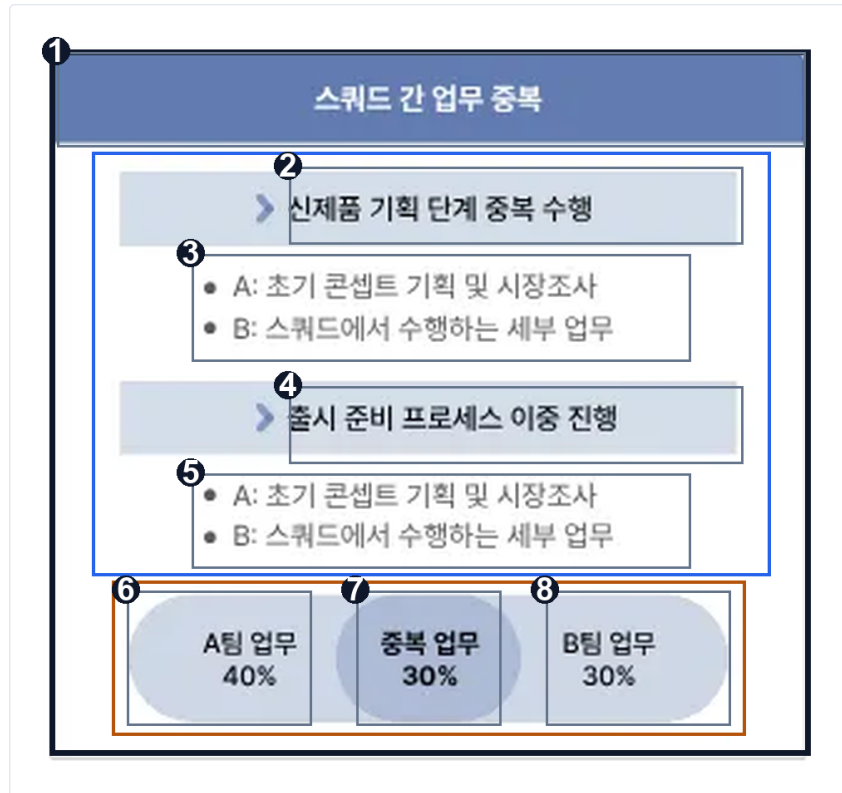


그림 2. 병치 위상. 2026-08-13 표본에서 실제로 걸린 컴포넌트 8244468(레이아웃 626391)의 썸네일 위에 경계를 덧그린 것이다. 검은 테두리가 컴포넌트 경계 하나이고, 그 안에서 파란 테두리(Enumeration.List)와 주황 테두리 (RelationalComparison.VennDiagram)가 겹치지 않고 위아래로 놓여 있다. 이것이 병치다.

#	화면 요소	들어가는 자리
1	제목 띠 "스쿼드 간 업무 중복"	적을 자리가 없다. title 은 page 수준에만 있고 컴포넌트 수준 제목 필드가 규약에 없다
2	소제목 "신제품 기획 단계 중복 수행"	components[0].structure.entities[0].label
3	볼릿 2줄	components[0].structure.entities[0].items[]
4	소제목 "출시 준비 프로세스 이중 진행"	components[0].structure.entities[1].label
5	볼릿 2줄	components[0].structure.entities[1].items[]
6	왼쪽 타원 "A팀 업무 40%"	components[1].structure.entities[0].label
7	가운데 겹친 영역 "중복 업무 30%"	components[1].structure.relations[0].desc
8	오른쪽 타원 "B팀 업무 30%"	components[1].structure.entities[1].label

7번이 관계가 왜 필요한지를 보여준다. 겹친 영역의 문구는 A팀에도 B팀에도 속하지 않는다. 그래서 entity 가 아니라 relations[0].desc 로 간다. 이 컴포넌트는 경계 하나에 카테고리가 둘이므로 색인 태깅에서 뿌리가 Unclassified 가 되고, D-7 에 따라 검색 풀에서 빠진다.

1번은 규약의 빈자리다. 컴포넌트마다 제목 띠가 붙는 디자인이 흔한데 Component 에는 제목 필드가 없다. 지금은 첫 entity 의 label 로 접거나 버리는 수밖에 없고, 어느 쪽인지 규약이 정한 바 없다.

뿌리가 없는 병치는 처음부터 다중 컴포넌트다. 서로 다른 카테고리가 다른 무엇에도 속하지 않고 나란히 있으면 최상위 components[] 를 늘리면 된다. 생성기가 이미 쓰는 모양이다.

```

version: "0.5.3"
title: 이번 분기 하이라이트
components:
  - type: KPIKeyMetric.BigNumber
    structure:
      entities:
        - { label: 총매출, value: 152, unit: 억원 }
  - type: Enumeration.List
    structure:
      entities:
        - { label: 신규 고객 320건 }
        - { label: 재구매율 68% }
        - { label: 해외 매출 비중 22% }

```

뿌리가 있는 병치는 항목마다 컴포넌트를 하나씩 매단다. 부모의 `entities[]` 를 늘려서 항목마다 자기 컴포넌트를 하나씩 갖게 한다. entity 하나에 서로 다른 카테고리 둘을 몰아넣는 전례는 없다.

```

version: "0.5.3"
title: 분기별 핵심 지표
components:
  - type: Enumeration.List
    structure:
      entities:
        - label: 총매출
          components:
            - type: KPIKeyMetric.BigNumber
              structure:
                entities:
                  - { label: 총매출, value: 152, unit: 억원 }
        - label: 핵심 성과
          components:
            - type: Enumeration.List
              structure:
                entities:
                  - { label: 신규 고객 320건 }
                  - { label: 재구매율 68% }
                  - { label: 해외 매출 비중 22% }

```

뿌리가 있는 병치가 실패하면 조각마다 따로 검색해 최상위로 편다. 부모가 두 항목 말고 자기 고유 항목이 없으면 부모 자체를 버린다. 결과는 뿌리 없는 병치와 똑같은 모양이 된다. 계기만 다르다. 뿌리 없는 병치는 처음부터 이 모양이 정답이었고, 뿌리가 있던 병치는 묶어서 찾다 실패해 이 모양으로 물러난 것이다.

7.3 둘이 함께 있는 경우

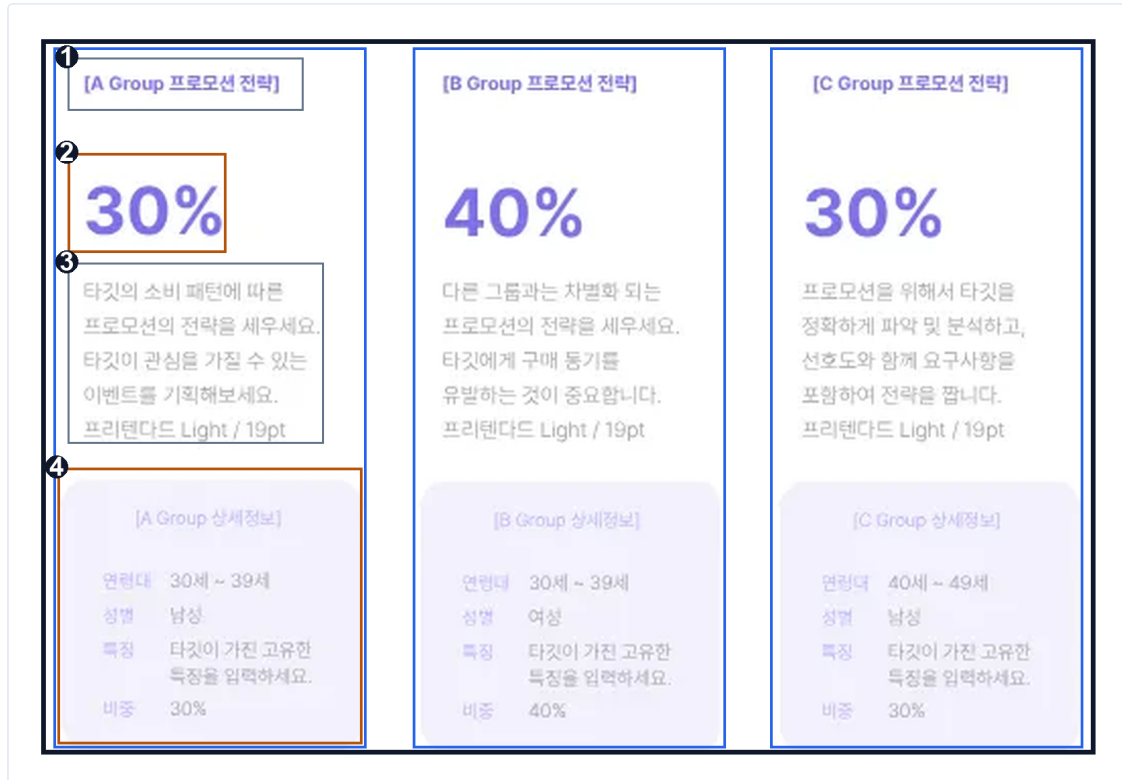


그림 3. 병치와 포함이 한 경계 안에 함께 있는 모양. 표본에서 걸린 컴포넌트 8183278(레이아웃 731419)이다. 판정 근거는 "A/B/C Group 별로 30%, 40%, 30% 수치가 큰 폰트로 강조되고, 그 아래에는 연령대, 성별, 특징, 비중 항목이 나열된 상세정보 박스가 별도로 있다" 였고, 태그는 `BigNumber` 와 `List` 둘이 붙었다.

검은 테두리가 컴포넌트 경계이고, 파란 테두리 셋이 A, B, C 그룹 영역이다. 그룹 영역 안에 주황 테두리 둘이 들어 있다. 아래는 A 그룹만 번호를 붙였고 B, C 는 같은 모양이 반복된다.

#	화면 요소	들어가는 자리
1	그룹 제목 "[A Group 프로모션 전략]"	<code>structure.entities[0].label</code>
2	대형 수치 "30%"	<code>entities[0].components[]</code> 의 <code>KPIKeyMetric.BigNumber</code>
3	설명 문단 4줄	<code>structure.entities[0].items[]</code>
4	상세정보 박스 (연령대, 성별, 특징, 비중)	<code>entities[0].components[]</code> 의 <code>Enumeration.ListingTable</code>

두 위상이 층을 달리해 함께 있다. 그룹 영역이 대형 수치와 상세정보 박스를 자기 안에 담고 있으므로 그 둘은 그룹에 대해 포함이다. 그리고 대형 수치와 상세정보 박스는 서로 겹치지 않고 위아래로 놓여 있으므로 둘 사이는 병치다. 위상은 부모자식 한 쌍마다 따로 정해지는 것이라 한 컴포넌트 안에서 이렇게 섞인다.

이 컴포넌트는 지금 표기로 깔끔하게 적히지 않는다. entity 하나(A 그룹)가 자기 안에 서로 다른 카테고리 둘을 담고 있는데, 규약 8.7 절의 전례는 entity 하나에 컴포넌트를 정확히 1개 매다는 모양뿐이다. 스키마의 `entity.components[]` 는 배열이라 둘을 넣어도 검증은 통과하지만, 전례가 없어서 검색이 이것을 어떻게 다룰지 정해진 바가 없다. 아래는 검증만 통과시킨 모양이고, 표기로 확정된 것이 아니다.

```

version: "0.5.3"
title: 그룹별 프로모션 전략
components:
  - type: Enumeration.List
    structure:
      entities:
        - label: "[A Group 프로모션 전략]"
          items:
            - 타깃의 소비 패턴에 따른 프로모션의 전략을 세우세요.
            - 타깃이 관심을 가질 수 있는 이벤트를 기획해보세요.
          components:
            - type: KPIKeyMetric.BigNumber
              structure:
                entities:
                  - { label: 비중, value: 30, unit: "%" }
            - type: Enumeration.ListingTable
              params:
                header: false
              structure:
                dataset:
                  shape: matrix
                  columns:
                    - { label: 항목, role: category }
                    - { label: 내용, role: value }
                  rows:
                    - [연령대, 30세 ~ 39세]
                    - [성별, 남성]
                    - [특징, 타깃이 가진 고유한 특징을 입력하세요.]
                    - [비중, "30%"]
# B Group, C Group 도 같은 모양이 반복된다

```

대치가 실패하면 총마다 다른 갈래를 탄다. 그룹 안의 포함은 7.1절대로 평탄화하거나 떼어내고, 떼어낸 조각 둘 사이의 병치는 7.2절대로 각자 따로 검색해 대치한다. 두 갈래가 한 컴포넌트 안에서 함께 필요하다는 것이 이 경우의 특징이고, 그 오케스트레이션이 아직 없다는 것이 7.4절 셋째 줄이다.

7.5 복합 구조 fallback 을 한 단계씩 따라간다

앞의 7.1부터 7.3 은 모양을 보여줬고, 이 절은 그 모양이 색인에서 안 걸릴 때 무엇이 순서대로 벗겨지는지를 끝까지 따라간다. 순서의 SSoT 는 spec/capability-table.md 3.6절이고 2절의 '복합 구조 fallback' 절이 규칙 전량을 요약한다. 이 절의는 실측이 아니라 규약이 표현할 수 있는 모양으로 만든 것이다. 1단계 생성기가 entity.components[] 를 아직 채우지 않으므로 지금 나올 수 있는 값이 아니다. 후보의 유무도 실측이 아니라 D-7 정책에서 따라 나오는 것이다.

출발 질의. 최상위에 병치가 있고(A 와 B), A 의 항목마다 안에 포함이 있고, 그 포함이 서로 다른 카테고리 둘을 담는다. 7.3절 실측 사례(컴포넌트 8183278)와 같은 모양이다.

```

version: "0.5.3"
title: 2026 상반기 사업 리뷰
components:
  # A. 분기 4개. 항목마다 대형 수치와 서술 요약이 함께 들어간다
  - type: Enumeration.List
    structure:
      entities:
        - label: 1분기
          items: [신규 계약 12건]
          components:
            - type: KPIKeyMetric.BigNumber
              structure:
                entities:
                  - { label: 매출, value: 12, unit: 억원 }
            - type: TextBasedCommunication.LongFormTextLayout
              structure:
                entities:
                  - label: 요약
                    items:
                      - 신규 시장 세 곳에 진입해 계약 12건을 확보했다.
                      - 단가 인하 없이 확보한 계약이라 마진이 유지됐다.
        - label: 2분기
          items: [신규 계약 15건]
          components:
            - type: KPIKeyMetric.BigNumber
              structure:
                entities:
                  - { label: 매출, value: 15, unit: 억원 }
            - type: TextBasedCommunication.LongFormTextLayout
              structure:
                entities:
                  - label: 요약
                    items:
                      - 기존 고객 재계약이 늘어 계약 건수가 증가했다.
        # 3분기, 4분기도 같은 모양이 반복된다
  # B. 하반기 실행 계획 3단계
  - type: ProcessFlow.Steps
    structure:
      entities:
        - { label: 채널 정비 }
        - { label: 가격 개편 }
        - { label: 해외 확장 }

```

노드는 이렇게 센다. 최상위 2개, A의 항목마다의 묶음 4개, 그 묶음 안의 단일 카테고리 8개다. **항목마다의 묶음에는 붙일 카테고리 이름이 없다.** 항목 하나가 서로 다른 카테고리 둘을 담고 있어서다. 색인 태깅 어휘에는 이 자리를 가리키는 `Unclassified`가 있지만 질의 `type` enum에는 대응 값이 없고 검증이 거부한다. 그래서 N-R3이 이 자리를 카테고리로 검색하지 않고 부모의 슬롯 경계로만 쓰기로 정했다.

최상위 둘은 서로 독립으로 진행한다. B는 `ProcessFlow.Steps` 단일 카테고리이고 성립 조건이 entities 2개 이상이라 N1에서 바로 후보가 걸린다. 아래는 A만 따라간다.

단계	무엇을 시도했나	결과	근거
N1	A의 트리 전체를 그대로 담은 후보	후보 0	뿌리가 <code>Unclassified</code> 인 템플릿이 D-7로 검색 풀에서 빠져 있다

단계	무엇을 시도했나	결과	근거
N2	모양을 고정한 채 노드마다 카테고리를 내린다. N-R6 순서로 합 1인 조합 셋(<code>BigNumber</code> → <code>Gauge</code> , <code>LongFormTextLayout</code> → <code>List</code> , <code>List</code> → <code>ListingTable</code>), 그다음 합 2 이상	후보 0	모양이 이중중첩인 채로 남으므로 어느 조합도 같은 이유로 후보가 없다
N3	A 의 뿌리 <code>Enumeration.List</code> 를 벗겨 항목 4개를 최상위로 올린다. N-R2 로 <code>[4]</code> , <code>[3,1]</code> , <code>[2,2]</code> , <code>[2,1,1]</code> , <code>[1,1,1,1]</code>	후보 0	조각 하나하나가 여전히 <code>BigNumber</code> 와 <code>LongFormTextLayout</code> 의 이중 묶음이다
N4	조각마다 <code>Unclassified</code> 경계를 벗겨 단일 카테고리 8개로 만든다. N-R2 로 <code>[8]</code> 부터 줄여 간다	<code>[1,1,1,1,1,1,1,1]</code> 에서 후보가 걸린다	아래
N5	여기까지 전부 실패했을 때만 평탄화한다	반드시 성립	마지막 fallback 셋

N4 에서 큰 묶음이 왜 전부 실패하나. 벗긴 뒤 형제 순서는 `BigNumber`, `LongFormTextLayout`, `BigNumber`, `LongFormTextLayout` 이 반복되는 모양이다. N-R2 는 연속한 형제만 묶고 순서를 바꾸지 않으므로, 크기 2 이상의 묶음은 어떻게 잘라도 이중 묶음이 된다. 그래서 크기 1까지 내려간다.

순서를 바꿔 같은 카테고리끼리 모으는 것은 허용하지 않는다. `BigNumber` 4개를 한 컴포넌트로, `LongFormTextLayout` 4개를 다른 컴포넌트로 모으면 각각 동종이라 N4 가 `[4,4]` 에서 끝난다. 그런데 읽는 순서가 1분기 매출, 1분기 요약, 2분기 매출 순에서 매출 넷, 요약 넷 순으로 바뀌고, 그러면 어느 요약이 어느 분기의 것인지가 배치로 드러나지 않는다. 이 제약을 풀어야 한다는 실측이 나오면 N-R2 를 고친다.

N4 통과 시점의 결과. 최상위 컴포넌트가 9개가 된다. A 에서 나온 8개와 B 1개다. 아래는 1분기 조각 둘만 적었다.

```
version: "0.5.3"
title: 2026 상반기 사업 리뷰
components:
  - type: KPIKeyMetric.BigNumber
    structure:
      entities:
        - { label: 1분기 매출, value: 12, unit: 억원 }
  - type: TextBasedCommunication.LongFormTextLayout
    structure:
      entities:
        - label: 1분기 요약
        items:
          - 신규 시장 세 곳에 진입해 계약 12건을 확보했다.
          - 단가 인하 없이 확보한 계약이라 마진이 유지됐다.
          - 신규 계약 12건
# 2분기, 3분기, 4분기 조각 여섯이 같은 모양으로 이어진다
  - type: ProcessFlow.Steps
    structure:
      entities:
        - { label: 채널 정비 }
        - { label: 가격 개편 }
        - { label: 해외 확장 }
```

N-R1 이 벗긴 부모의 내용을 어디로 보냈는지 보라. 뿌리 `Enumeration.List` 가 갖고 있던 항목 `label` ("1분기")과 `items` ("신규 계약 12건")는 버려지지 않았다. `label` 은 승격된 조각의 `label` 앞에 붙어 "1분기 매출", "1분기 요약"이 됐고, `items` 는 승격된 첫 조각의 첫 entity `items` 맨 뒤로 갔다.

N5 까지 갔을 때의 결과. 비교를 위해 적어 둔다.

```

version: "0.5.3"
title: 2026 상반기 사업 리뷰
components:
  - type: Enumeration.List
    structure:
      entities:
        - label: 1분기
          items:
            - 신규 계약 12건
            - "매출: 12억원"
            - 신규 시장 세 곳에 진입해 계약 12건을 확보했다.
            - 단가 인하 없이 확보한 계약이라 마진이 유지됐다.
        - label: 2분기
          items:
            - 신규 계약 15건
            - "매출: 15억원"
            - 기존 고객 재계약이 늘어 계약 건수가 증가했다.
  - type: ProcessFlow.Steps
    structure:
      entities:
        - { label: 채널 정비 }
        - { label: 가격 개편 }
        - { label: 해외 확장 }

```

두 결과를 나란히 놓고 보면 무엇을 걸고 무엇을 얻는지가 드러난다.

	N4 통과 (떼어내기)	N5 (평탄화)
텍스트	전량 보존	전량 보존
BigNumber 의 대형 수치 강조	살아 있다	사라진다. 글머리표 한 줄이 된다
분기와 매출, 요약의 공간적 짝	사라진다	살아 있다
분기 넷을 하나의 나열로 읽게 하던 배치	사라진다	살아 있다
최상위 컴포넌트 수	9개	2개

그래서 N4 가 N5 보다 앞이다. 둘 다 텍스트를 안 버리는데 N4 쪽이 카테고리 표현을 더 남긴다. 표현을 더 많이 남기는 쪽을 먼저 두는 것이 확정된 순서다. 다만 N4 가 언제나 더 나은 결과라는 뜻은 아니다. 위 표에서 보듯 공간적 짝이 내용의 핵심인 질의에서는 N5 가 나올 수 있고, 그 판정 기준은 아직 없다. N-R5 가 계층 대분류에 대해서만 그 예외를 제안하고 있고 미확정이다(B-31).

이 예시가 드러낸 것 셋. 첫째, D-7 이 유지되는 동안 이중 질의는 실질적으로 N3 에서 시작한다. N1 과 N2 가 둘 다 이중중첩인 모양 그대로 후보를 찾기 때문이다. 둘째, 이중 질의에서 N3 한 번으로는 끝나지 않는 경우가 흔하다. 뿌리 아래에 Unclassified 경계가 한 층 더 있으면 N4 가 필요하고, 7.3절 실측 사례가 그 모양이다. 셋째, 반복하면 최상위 컴포넌트 수가 늘어난다. 1개가 9개가 됐고, 규약 8.6절이 최상위 원소 수에 상한을 두지 않았으므로 규약 위반은 아니지만 한 페이지에 9개를 배치할 수 있는지는 확인한 바 없다(B-32).

7.4 아직 정해지지 않은 것

무엇	상태
질의 쪽 포함 위상을 단일 type 으로 평탄화한 뒤 fallback 할지, entity.components[] 를 유지한 채 노드 태그로 대체 범위를 자를지	단했다 (2026-08-28, I-12). 유지한다. 구조를 그대로 두고 노드마다 카테고리를 내려 보고(N2), 실패하면 최상위 부모부터 벗기고(N3, N4), 평탄화(N5)는 마지막에만 한다. 순서는 7.5절과 2절의 '복합 구조 fallback' 절이다

무엇	상태
색인 태깅 어휘에는 이중 병치 뿌리를 가리키는 <code>Unclassified</code> 가 있는데, 질의 스키마 <code>type enum</code> (결합 값 45종)에는 대응 값이 없다	열려 있다. <code>type: Unclassified</code> 는 지금 검증에서 거부된다
병치를 쪼개야 한다고 인식하고, 조각마다 검색을 호출하고, 결과를 다시 조립하는 코드	없다. 검색(T-5)만 있고 메인 파이프라인에도 아직 안 엮혔다. 무엇을 어떤 순서로 호출해야 하는지는 2026-08-28에 정해졌다(I-11)
entity 하나가 서로 다른 카테고리 둘 이상을 담는 표기	스키마는 통과하지만 전례가 없다. 규약 8.7절은 entity 하나에 정확히 1개를 매다는 모양만 적어 뒀다(7.3절 실측 사례가 이 경우다)
컴포넌트 수준 제목 필드	없다. <code>title</code> 은 page 수준에만 있다(7.2절 1번)
이중중첩으로 태깅된 색인 템플릿	검색 풀에서 뺐다(D-7, 2026-08-11). 뿌리가 <code>Unclassified</code> 인 템플릿은 어느 질의에도 후보로 안 걸린다
이중중첩 템플릿을 검색 풀에 되돌릴지	열려 있다(D-8, 2026-08-28 신설). D-7 이 유지되는 동안 복합 구조 순서의 N1 과 N2 는 이중 질의에서 언제나 후보가 0이다
떼어내기(N3)와 평탄화(N5) 중 어느 쪽이 내용을 더 잘 보존하는지	닫혔다 (2026-08-28) . 둘 다 텍스트를 안 버리지만 떼어내기가 카테고리 표현을 더 남기므로 떼어내기가 먼저다. 다만 공간적 짝이 핵심인 질의의 예외 기준은 없다(B-31)