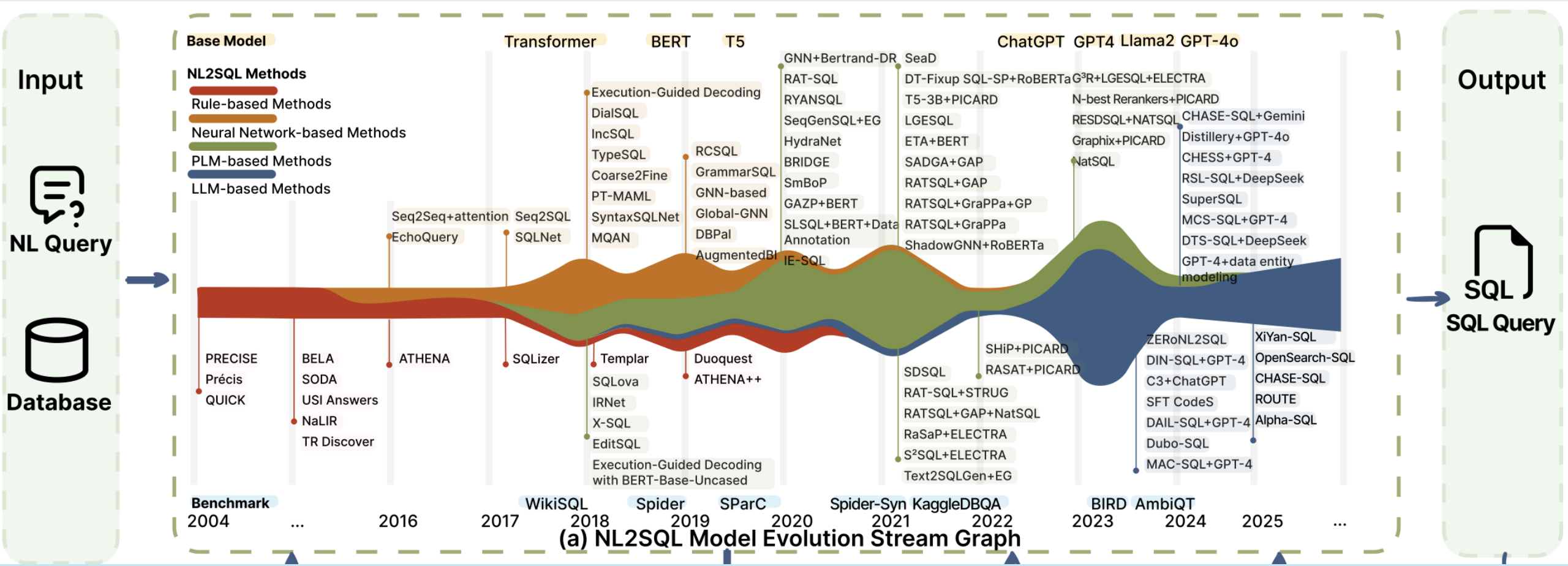
Same Data, Different Schemas: Robustness of LLM-based Text-to-SQL

Nitin Kanchinadam, Aditya Menachery, Amol Deshpande
University of Maryland at College Park

NL2SQL

- Major focus of research in AI+DM in the last decade

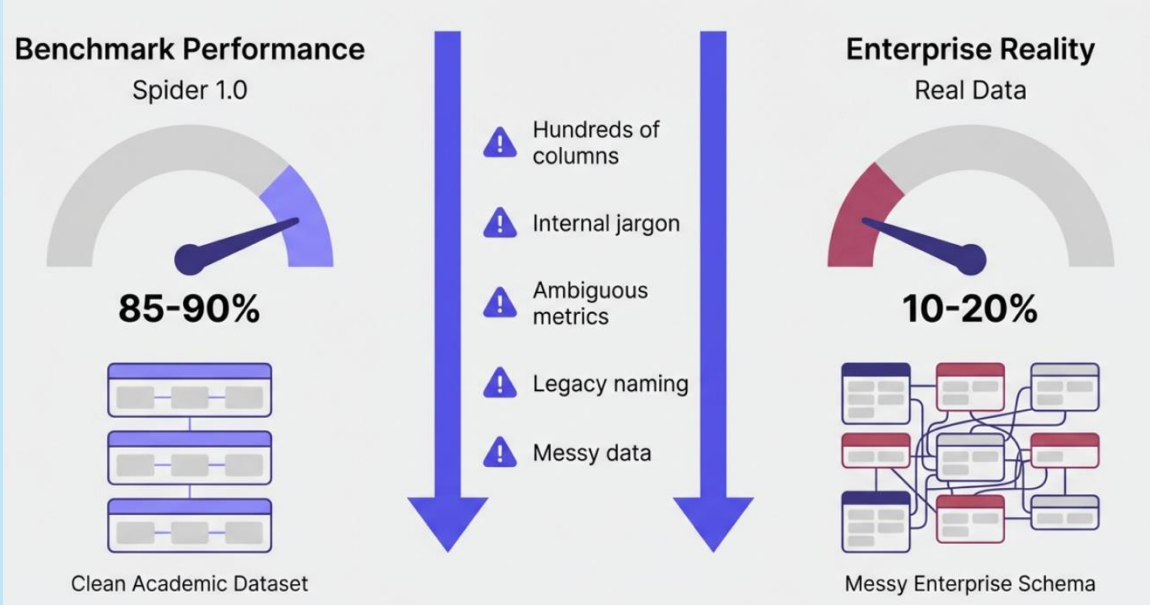


Natural Language to SQL: State of the Art and Open Problems; Luo et al.; VLDB 2025

NL2SQL

- Very good performance on benchmarks
- But high failure rates in practice
- Our focus: how well NL2SQL tools ”understand” schemas and handle schema variations?

Rank	Method	Score
1 Mar 1, 2026	Genloop's Sentinel Agent v2 Pro *Genloop*	96.70
2 Feb 18, 2026	Native mini *usenative.ai*	96.53
3 Feb 9, 2026	QUVI-3 + Gemini-3-pro-preview *DAQUV*	94.15
4 Feb 3, 2026	TCDataAgent-SQL with Contextual Scaling Engine *Tencent Cloud Big Data*	93.97
5 Jan 27, 2026	Prism Swarm with Deepthink + Claude-Sonnet-4.5 *Paytm*	90.49
6 Feb 17, 2026	Genloop's Sentinel Agent v2 *Genloop*	88.48



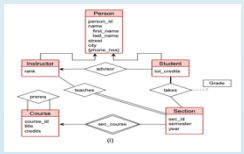
https://www.blazesql.com/blog/natural-language-to-sql

Recent Prior Work on Schema Impact

- Exploring Schema Generalizability [Li et al., ACL 2023]
  - Constructed Spider variants using local schema changes
  - Significant decline in performance of RAT-SQL and similar techniques
- FootballDB [Furst et al., EDBT 2025]
  - Three hand-curated schemas for the same FIFA dataset
  - No major differences across schemas for LLMs (more for specialized models)
- Effect of Normalization [Kohita, CIKM 2025]
  - Notes significant impact of normalization choices on performance
  - Custom benchmark w/ conflicting results
- EVOSCHEMA [Zhang et al., PVLDB 2025]
  - Ten types of local schema perturbations on BIRD dataset
  - Same natural language queries (?)
  - GPT models exhibit stable performance, unlike fine-tuned open-weights models

Our Approach

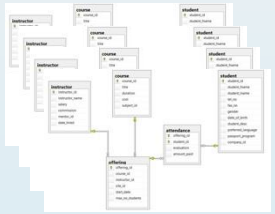
Multiple dataset instances can be created to increase coverage



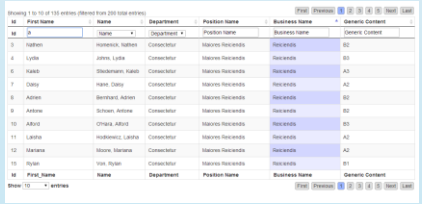
E/R Diagram

+

Data Generation Parameters



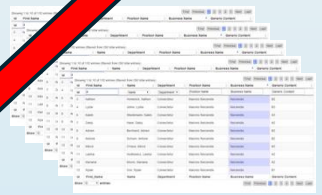
Relational Schemas



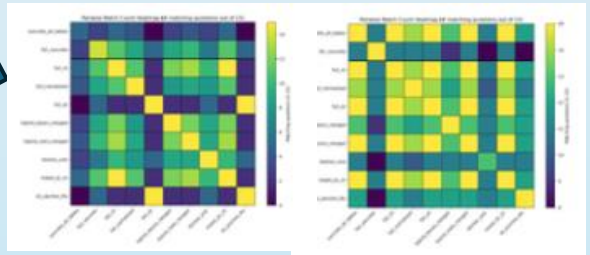
E/R Data

(1) Who are the 50 creators that own the largest number of channels?
(2) For each creator, how many total pieces of content have they produced?
(3) Who are the 50 streamers that have hosted the most live streams?
(4) How many clips were derived from each live stream?
(5) Who are the top 50 creators that have the largest number of subscribers?

NL Questions



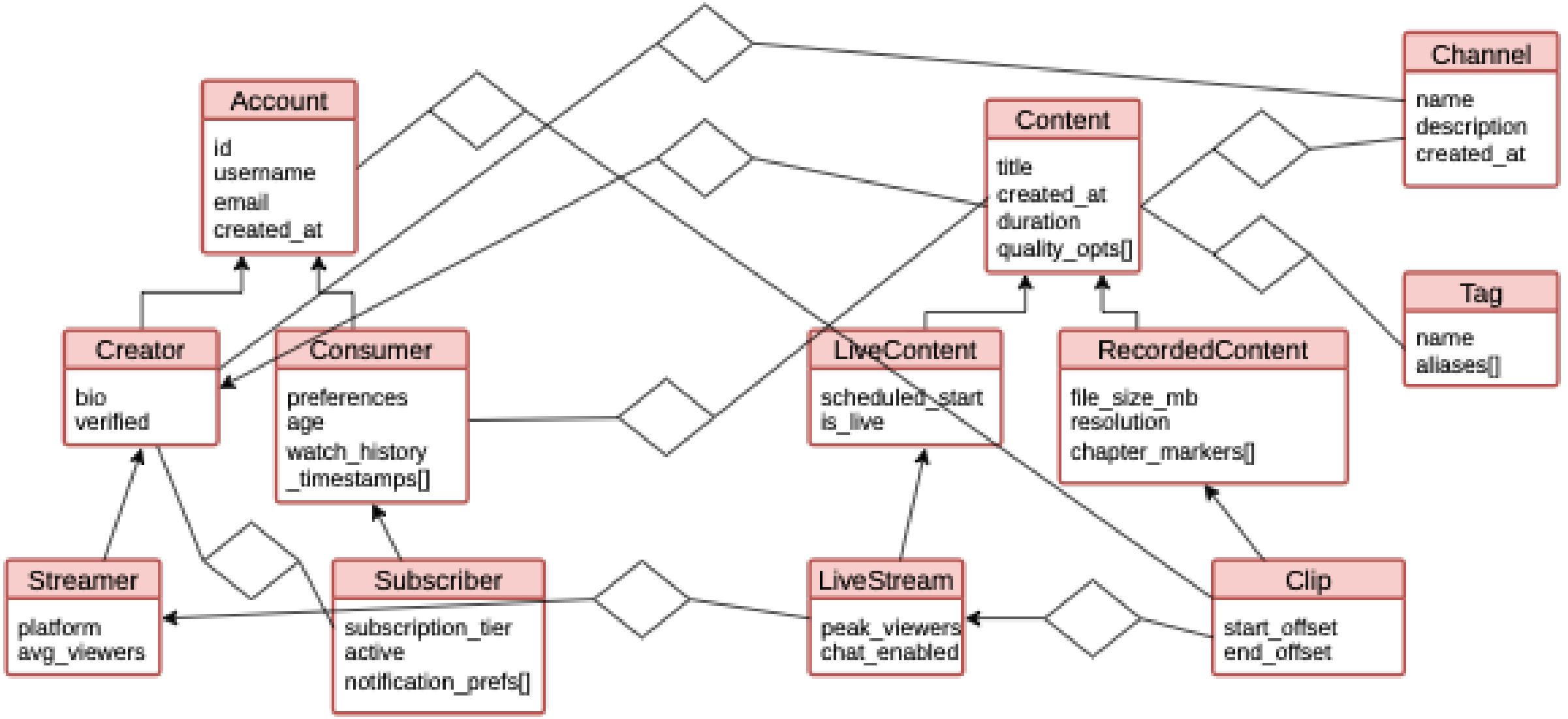
E/R- Equivalent RDBMS Instances



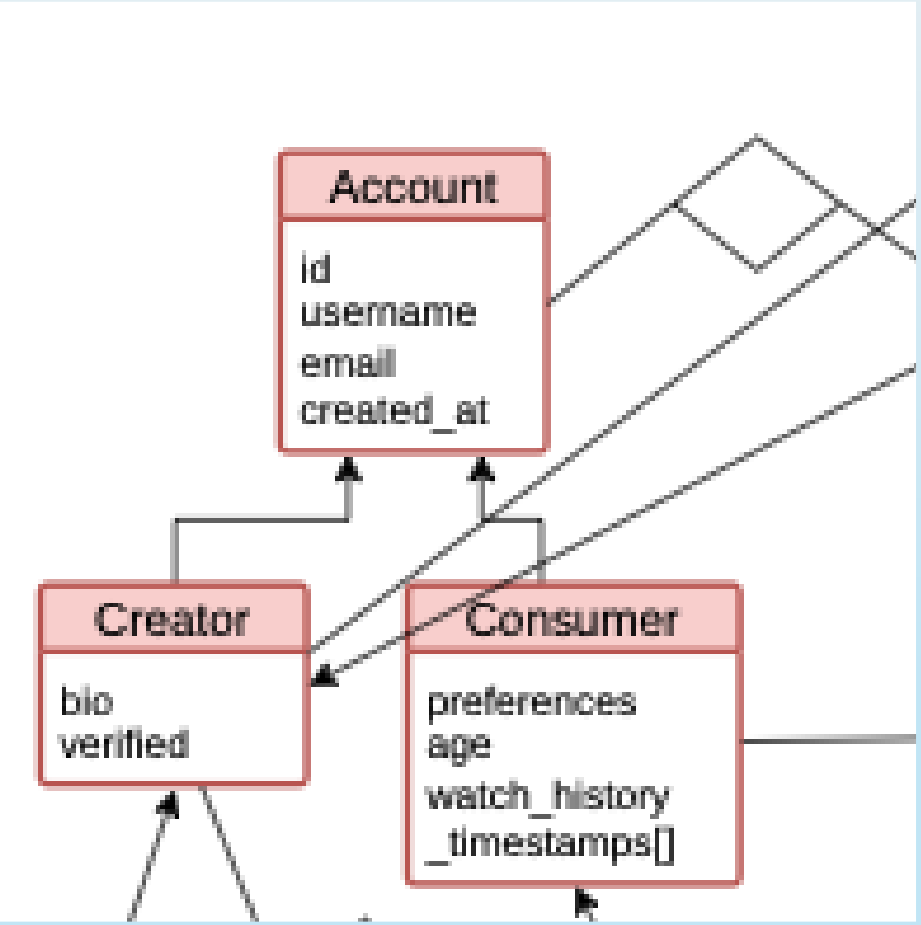
Cross-Schema Consistency Analysis

The entire process can be fully automated

Example E/R Schema: Social Media



Example E/R Schema: Social Media



Concrete Tables per Entity (accounts may not be created)

accounts(id, username, email, created_at)
creator_accounts(id, username, email, created_at, bio, verified)
consumer_accounts(id, username, email, created_at, preferences, age)

Class Table Inheritance

accounts(id, _type, username, email, created_at)
creator_accounts(id, bio, verified)
consumer_accounts(id, preferences, age, watch_history_timestamps)

Single Table Inheritance

accounts(id, _type, username, email, created_at, bio, verified, platform, avg_viewers, preferences, age, watch_history_timestamps, subscription_tier, active, notification_preferences)

And many other combinations…

Example E/R Schema: Social Media

Concrete Tables per Entity (accounts may not be created)

accounts(id, username, email, created_at)
creator_accounts(id, username, email, created_at, bio, verified)
consumer_accounts(id, username, email, created_at, preferences, age)

Class Table Inheritance

accounts(id, _type, username, email, created_at)
creator_accounts(id, bio, verified)
consumer_accounts(id, preferences, age, watch_history_timestamps)

Single Table Inheritance

accounts(id, _type, username, email, created_at, bio, verified, platform, avg_viewers, preferences, age, watch_history_timestamps, subscription_tier, active, notification_preferences)

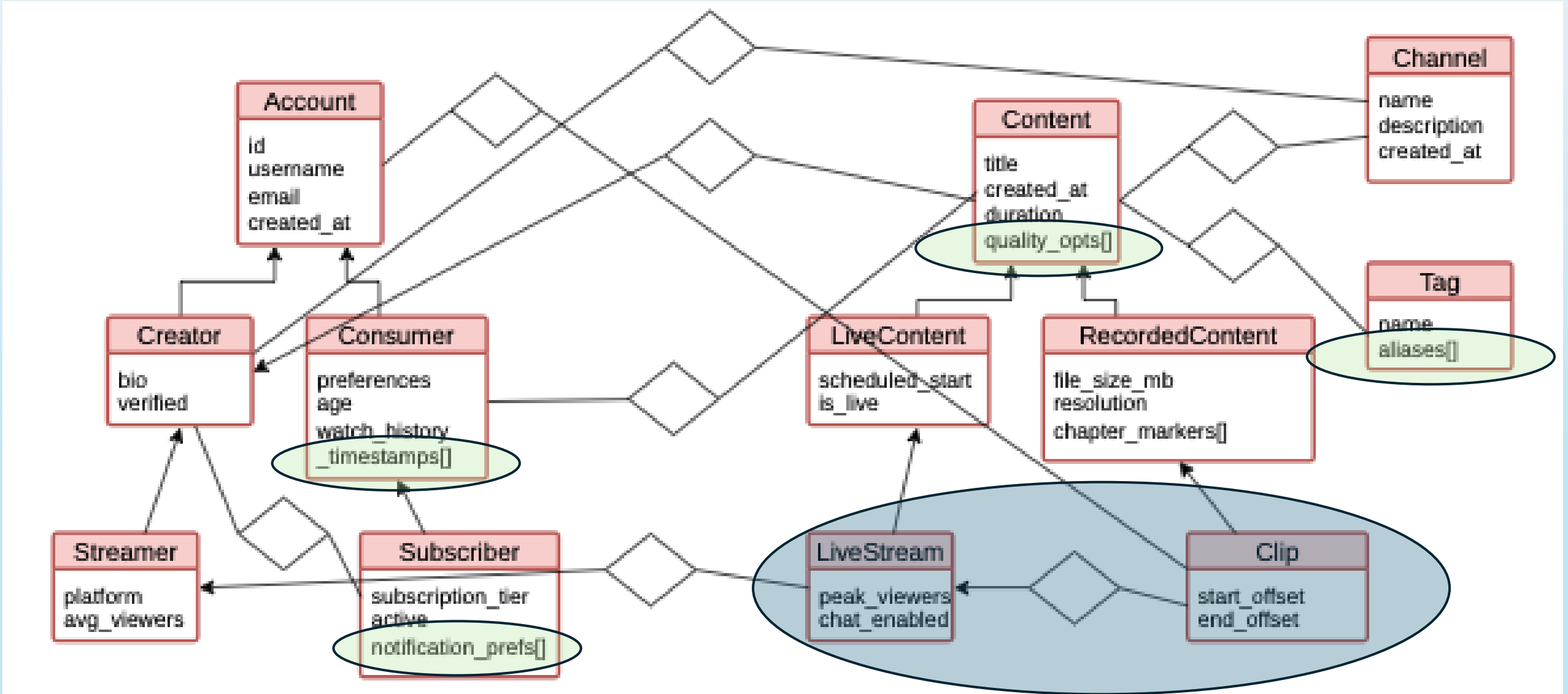
Natural Language Query: How many clips has each account shared?

```
select id, count(*)
from (select id from accounts
        union
      select id from creator_accounts
      union
      select id from consumer_accounts) T,
      join account_shares on id = account_id
group by id
```

vs

```
select id, count(*)
from accounts join account_shares
      on id = account_id
group by id
```

Example E/R Schema: Social Media



Example Relational Schemas

Schema Variant	Inheritance Strategy	Multi-valued Attributes	Relationship Encoding
`full_sti`	Single-table inheritance for both `Account` and `Content` hierarchies; subtype membership represented using discriminator columns	Prefer nested representation (arrays / JSONB) for attributes such as `aliases`, `quality_options`, and `chapter_markers`	Many-to-one relationships represented as foreign keys; many-to-many relationships via junction tables
`full_cti`	Class-table inheritance throughout; each superclass/subclass gets its own relation with keys	Fully normalized into separate child tables	Many-to-one relationships represented as foreign keys; many-to-many relationships via junction tables
`full_concrete`	Concrete-table inheritance for leaf classes; inherited attributes duplicated into concrete subtype tables	Fully normalized into separate child tables	Relationship endpoints attached to concrete tables when possible; many-to-many relationships via junction tables
`full_normalized`	Class-table inheritance throughout	Fully normalized into separate relations for every multi-valued attribute	Explicit relationship tables used aggressively, including some many-to-one relationships
`concrete_all_tables`	Concrete-table inheritance for concrete subclasses, with root entities retained when needed	Fully normalized into separate relations	Explicit tables for all many-to-many relationships and selected many-to-one relationships to maximize uniformity
`hybrid_roots_merged`	Root and first-level hierarchy nodes merged; leaf subtypes such as `Streamer`, `Subscriber`, `LiveStream`, and `Clip` remain separate	Mixed: frequently queried attributes normalized; smaller collections kept nested	Foreign keys for many-to-one relationships; standard junction tables for many-to-many relationships
`hybrid_leaves_merged`	Root entities represented separately, but leaf classes merged into parent subtype tables where possible	Mixed: arrays / JSONB for compact attributes, normalized tables for larger collections	Mostly foreign-key based, with junction tables for many-to-many relationships
`mixed_sti_cti`	`Account` hierarchy uses STI, while `Content` hierarchy uses CTI	Mixed normalized and nested representation	Standard foreign keys for many-to-one relationships and junction tables for many-to-many relationships
`sti_junction_fks`	STI for both major hierarchies	Fully normalized for all multi-valued attributes	Explicit foreign keys and junction tables, minimizing nested columns despite STI
`kitchen_sink`	Mix of STI, CTI, and concrete-style encodings across the two hierarchies	Mixed representation chosen attribute-by-attribute	Mixed encoding, including both foreign-key style and explicit tables

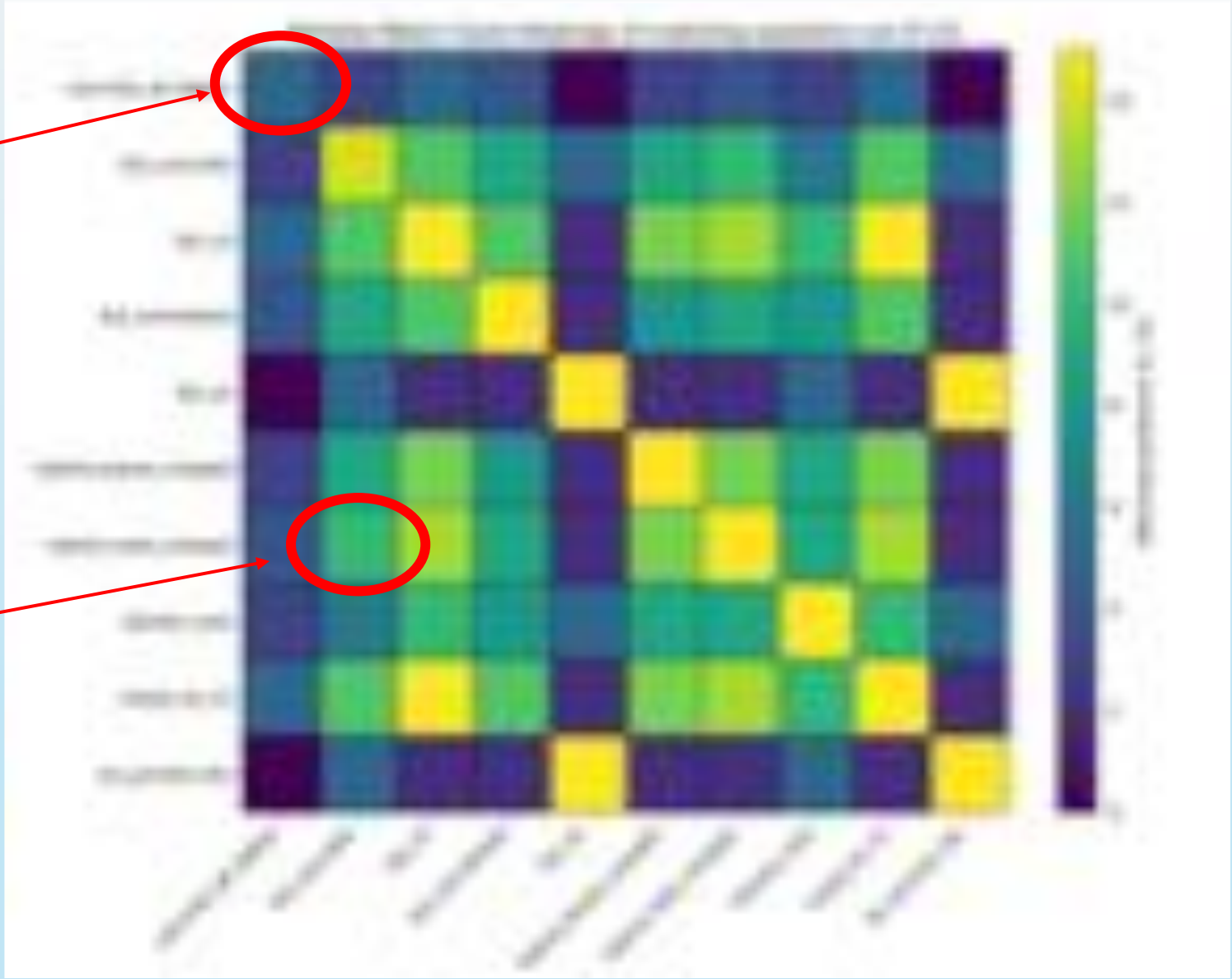
Example Queries

1. Who are the 50 creators that own the largest number of channels?
2. For each creator, how many total pieces of content have they produced?
3. Who are the 50 streamers that have hosted the most live streams?
4. How many clips were derived from each live stream?
5. Who are the top 50 creators that have the largest number of subscribers?
6. For each subscriber, how many different creators are they subscribed to?
7. Which 50 channels have published the most content items?
8. How many clips has each account shared?
9. What are the 50 tags that are used on the largest number of content items?
10. For each creator, how many times has content they produced been viewed by consumer accounts
11. Who are the top 50 creators that produce content that is published on the greatest number of distinct channels?
12. How many live streams hosted by each streamer have at least one derived clip?
13. Which subscribers follow creators who own more than one channel?
14. For each channel, how many live streams and how many clips have been published on it?
15. Who are the top 50 creators that have produced content tagged with 3 or more tags?

Cross-schema Heatmaps

Some queries had syntax errors

For a schema pair, how many of the 15 queries had the same result on the database?



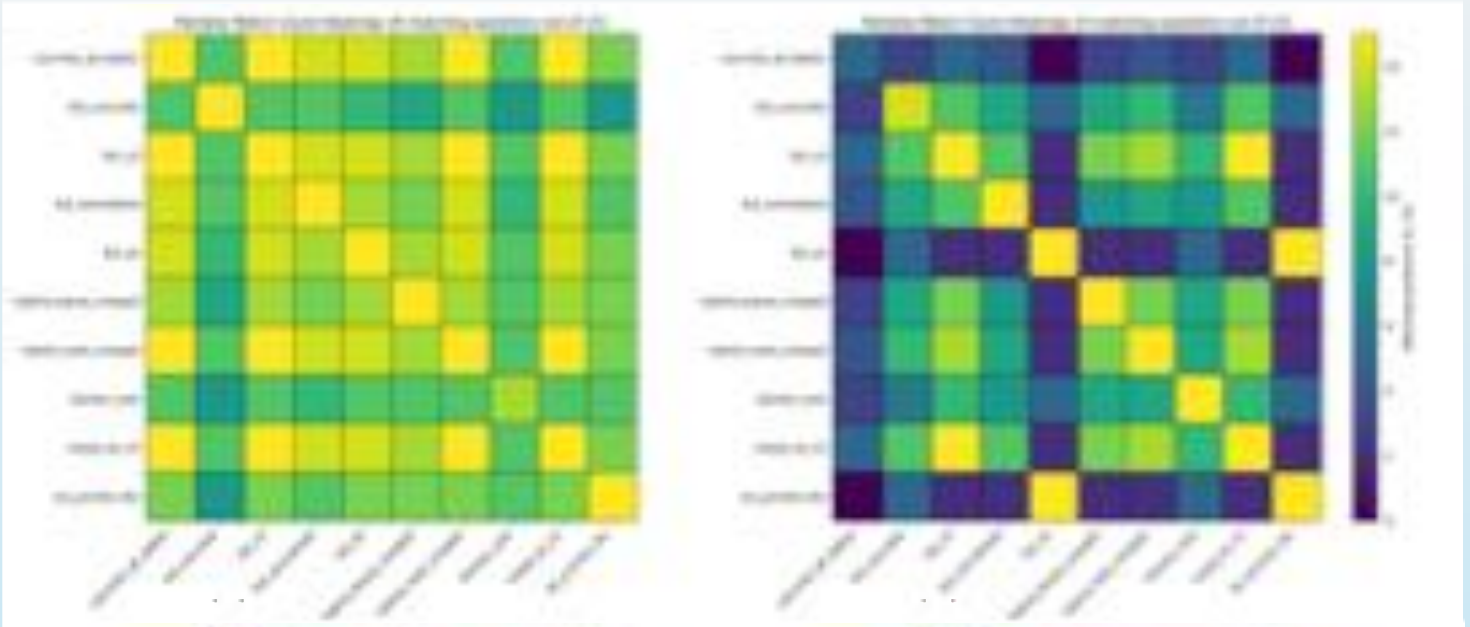
ChatGPT 5.2

43.62% Consistency

Cross-schema Heatmaps

Gemini 3 Flash

81.6% Consistency



ChatGPT 5.2

43.62% Consistency

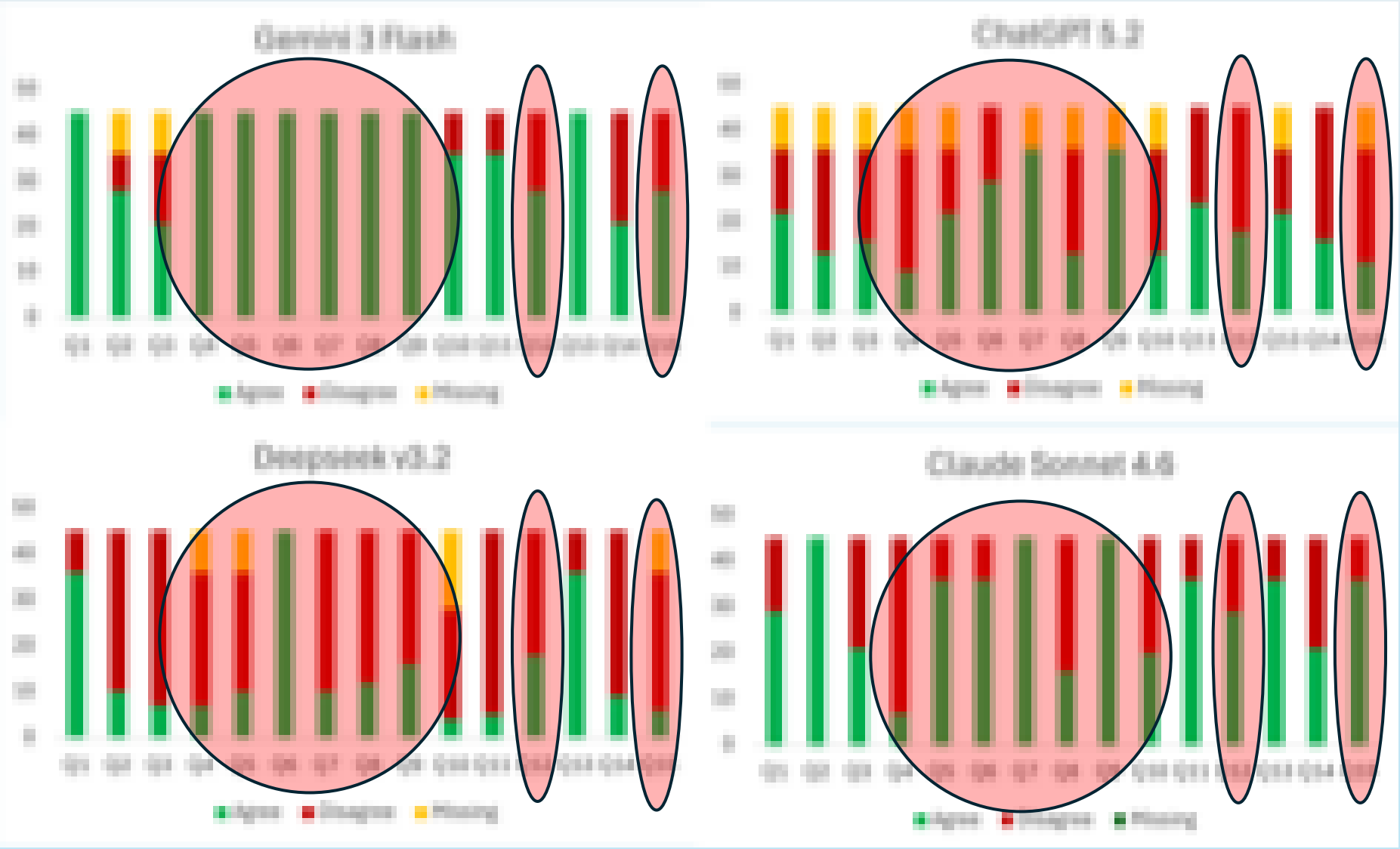
DeepSeek V3.2

33.85% Consistency

Claude Sonnet 4.6

67.8% Consistency

Agreement by Question

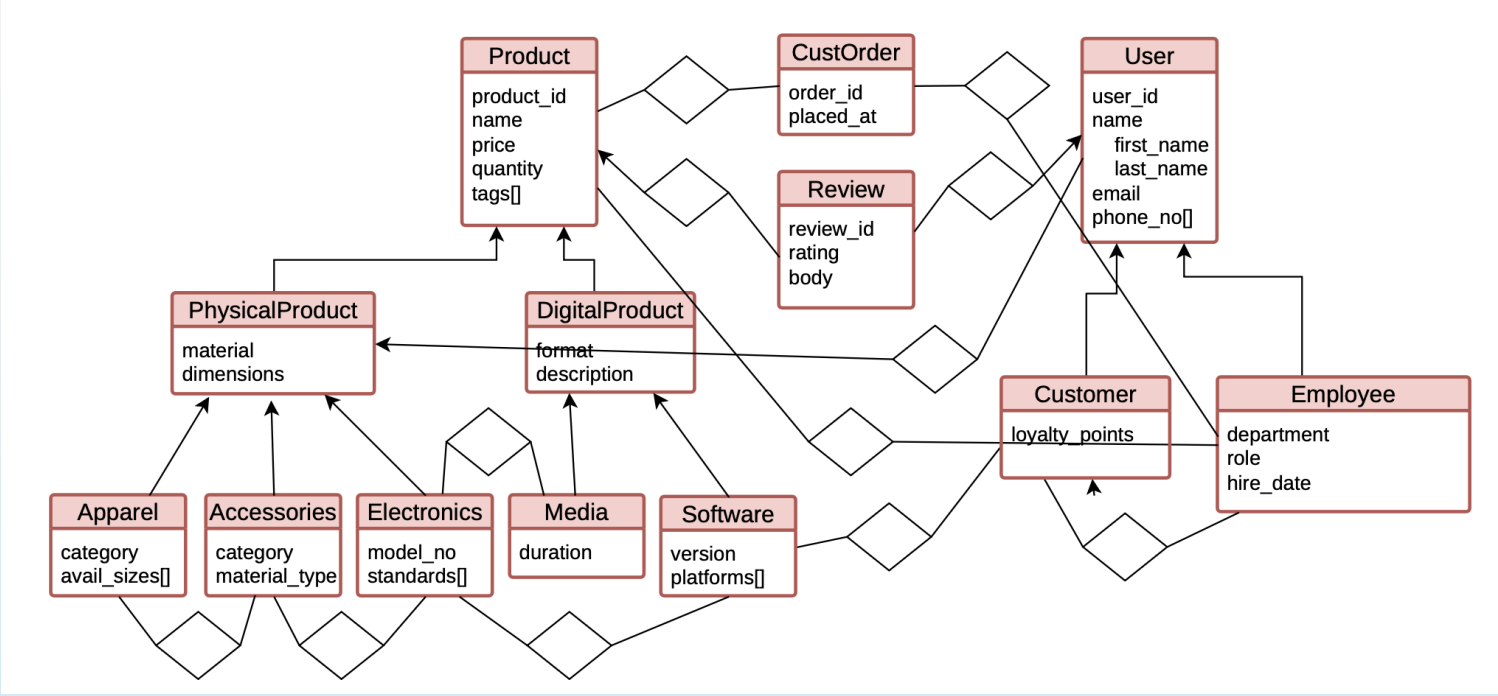


Q1, Q4-Q9: Relatively straightforward aggregations

Q12: How many live streams hosted by each streamer have at least one derived clip?

Q15: Who are the top 50 creators that have produced content tagged with 3 or more tags?

Retail Dataset



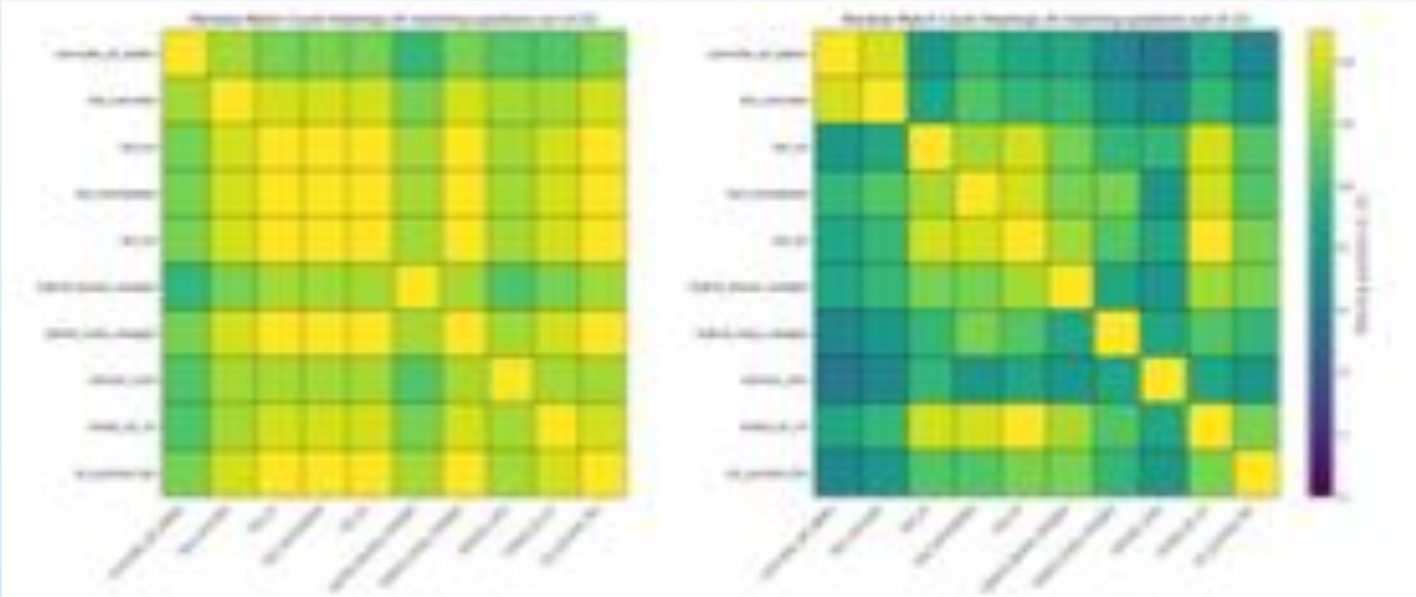
Example Query 1: For each department, compute the total number of employees in that department and the average number of orders processed per employee, rounded to two decimal places.
Include all departments that have at least one employee.
Order by average orders processed descending; break ties by department name ascending.

Example Query 2: For each department, report the number of employees in that department, the total number of distinct products recommended by those employees, and the total number of distinct orders processed by those employees.
Return one row per department.
Order by department name ascending.

Cross-schema Heatmaps

Gemini 3.1 Pro Preview

88.7% Consistency

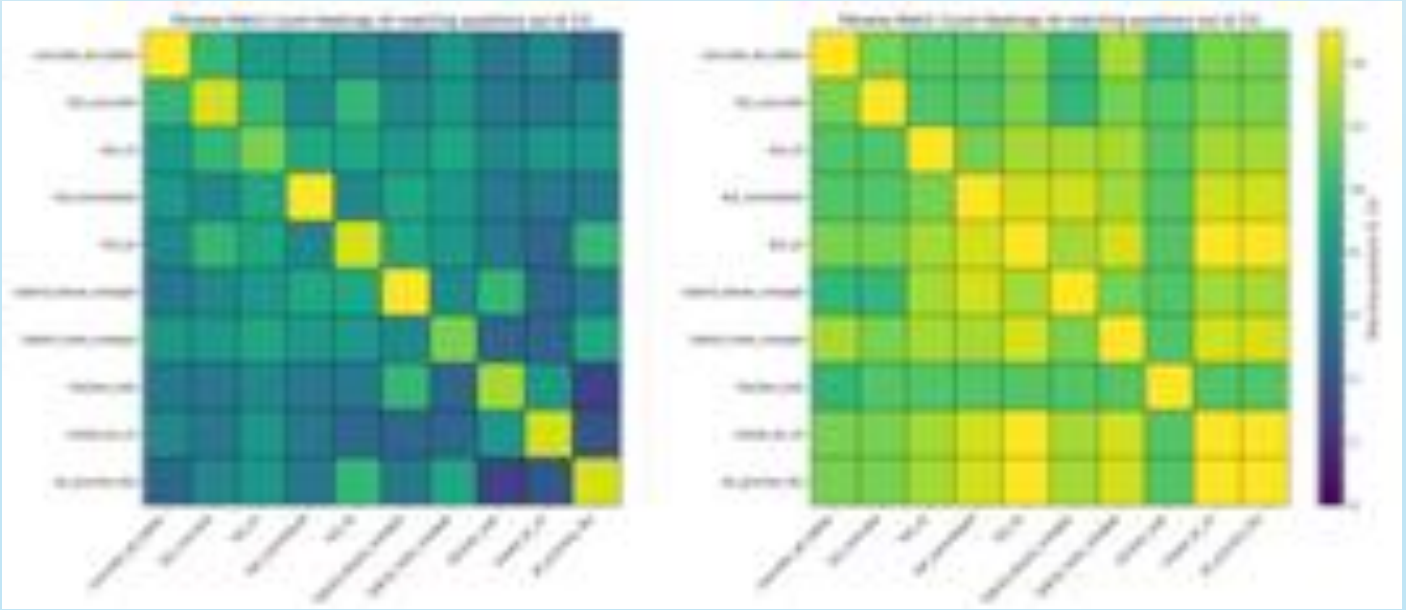


ChatGPT 5.4

69.33% Consistency

DeepSeek V4 Pro

47.4% Consistency



Claude Opus 4.7

82.2% Consistency

What if we were to add E/R context?

```
### Entities and attributes

**Product hierarchy (3 levels)**
```
Product   product_id (PK), product_name, price, quantity, tags[]
├── PhysicalProduct   + material, dimensions
│   ├── Apparel   + category, available_sizes[]
│   ├── Accessories   + category, material_type
│   └── Electronics   + model_no, supported_standards[]
└── DigitalProduct   + format, description
    ├── Media   + duration
    └── Software   + version, supported_platforms[]
```
Dataset: 5,000 products total — 1,500 Apparel, 1,000 Accessories, 1,000 Electronics, 800 Media, 700 Software.

**User hierarchy (2 levels)**
```
User   user_id (PK), name, first_name, last_name, email, phone_no[]
├── Customer   + loyalty_points
└── Employee   + department, role, hire_date
```
Dataset: 2,000 users total — 1,500 Customers, 500 Employees.

**Flat entities**
- `CustOrder` — order_id (PK), placed_at
- `Review` — review_id (PK), rating, body

## Schema 1 — `full_sti`

**Strategy:** Single Table Inheritance (STI) for both hierarchies. List fields as PostgreSQL arrays. Foreign keys as inline columns.

**Product hierarchy → `products` table**
All 5,000 product instances are stored in a single `products` table. A `_type` text column acts as a discriminator (values: `Apparel`, `Accessories`, `Electronics`, `Media`, `Software`). Every column from every subclass appears in this one table; columns that do not apply to a given subtype are NULL. Key columns: `id` (SERIAL PK), `_type`, `product_id`, `product_name`, `price`, `quantity`, `tags` (TEXT[]), `material`, `dimensions`, `category`, `available_sizes` (TEXT[]), `material_type`, `model_no`, `supported_standards` (TEXT[]), `format`, `description`, `duration`, `version`, `supported_platforms` (TEXT[]).

**User hierarchy → `users` table**
All 2,000 user instances are stored in a single `users` table with a `_type` discriminator (values: `Customer`, `Employee`). Key columns: `id`, `_type`, `user_id`, `name`, `first_name`, `last_name`, `email`, `phone_no` (TEXT[]), `loyalty_points`, `department`, `role`, `hire_date`.
```

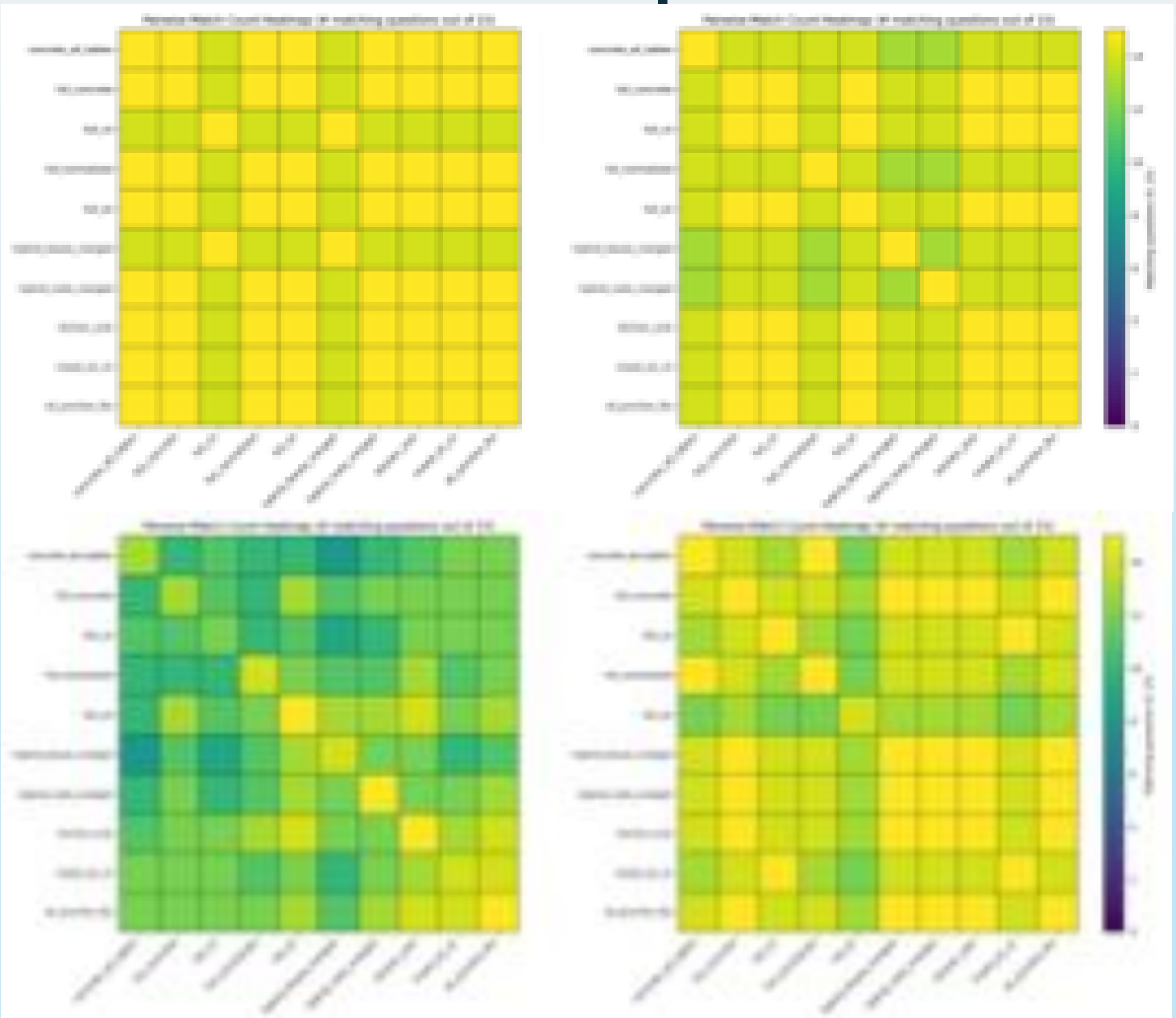
Cross-schema Heatmaps

Gemini 3.1 Pro Preview

ChatGPT 5.4

DeepSeek V4 Pro

Claude Opus 4.7

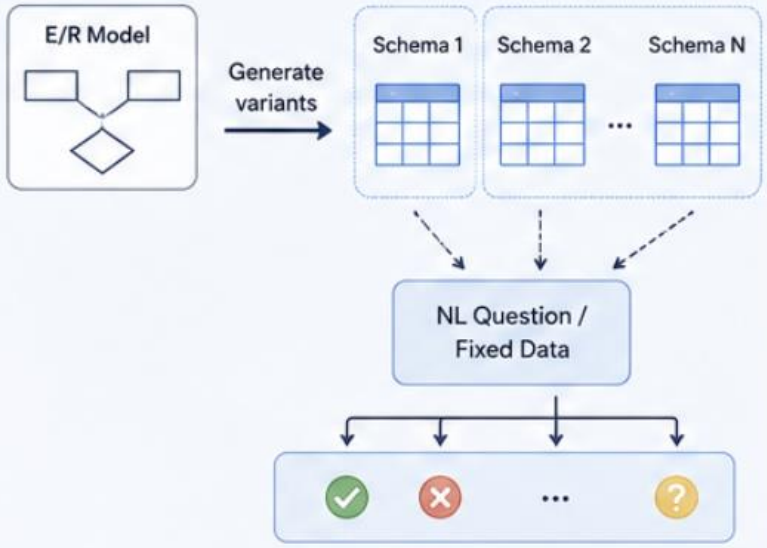


Limitations

- Relatively simple set of questions
- Zero-shot usage of LLMs
  - Recent NL2SQL techniques use agentic workflows
- Equivalence based on execution against a single database
  - Can build upon the work on SQL Query Mutants to better check equivalence
- Lack of ground truth restricts conclusions
- No real-world schema evolution that leads to messy and un-normalized schemas (“schema decay”)

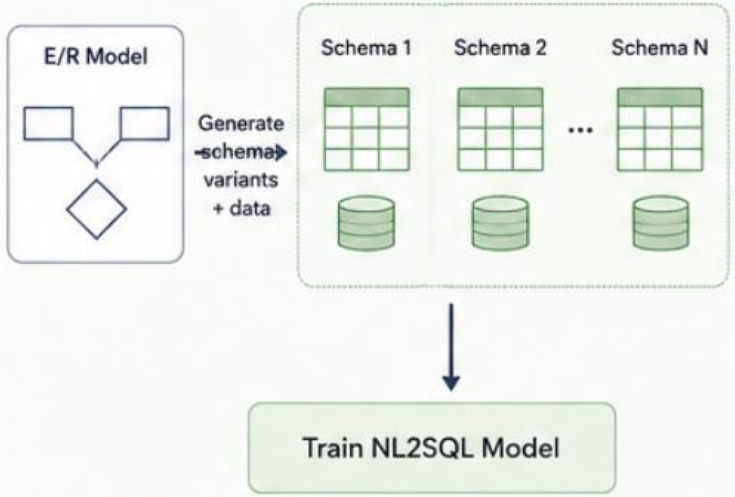
Towards Schema Robustness

1 Test robustness to schema variations at scale



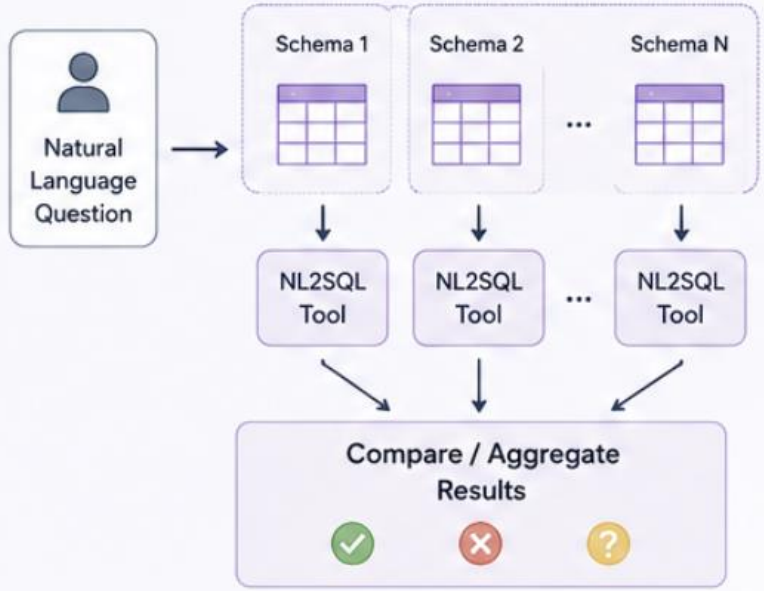
Systematically evaluate how schema design choices affect NL2SQL behavior across a large space of E/R-equivalent schemas.

2 Generate synthetic datasets for training



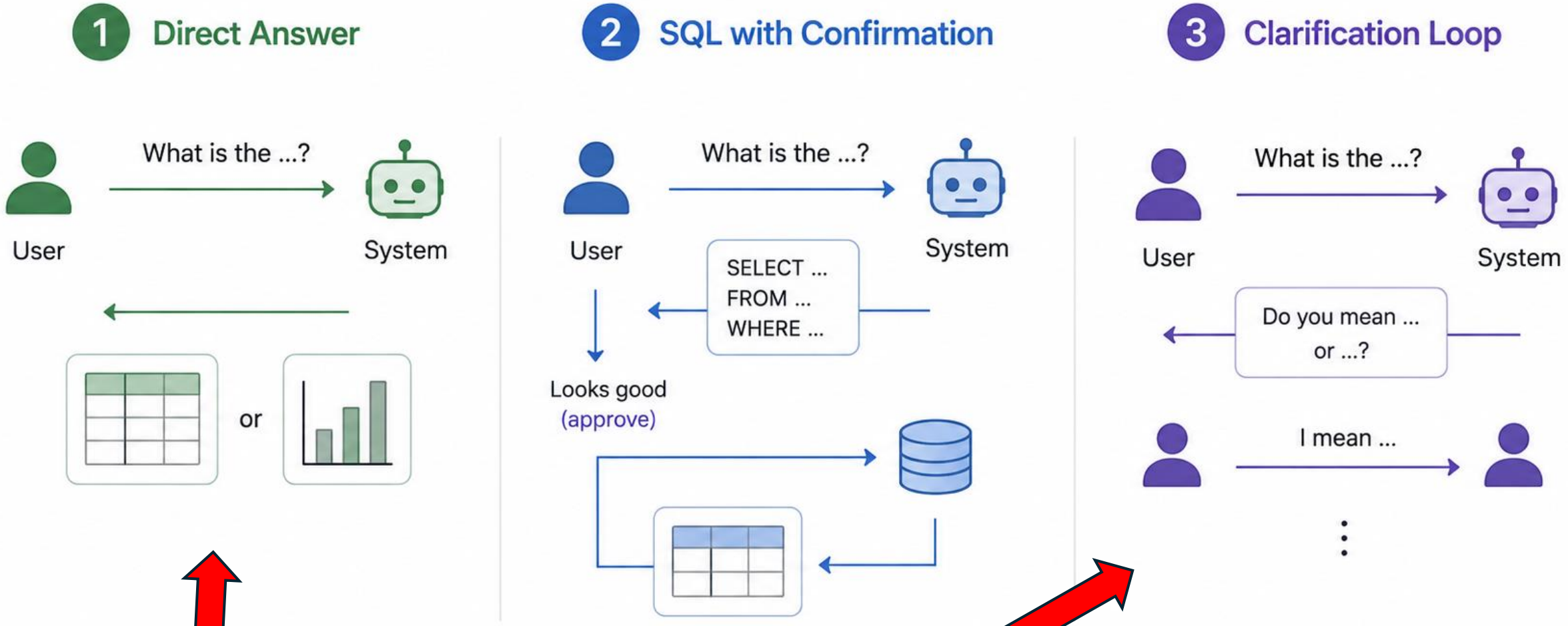
Create diverse, E/R-consistent training data to help models generalize better across different schema representations.

3 Inference-time ensembling over E/R-equivalent schemas



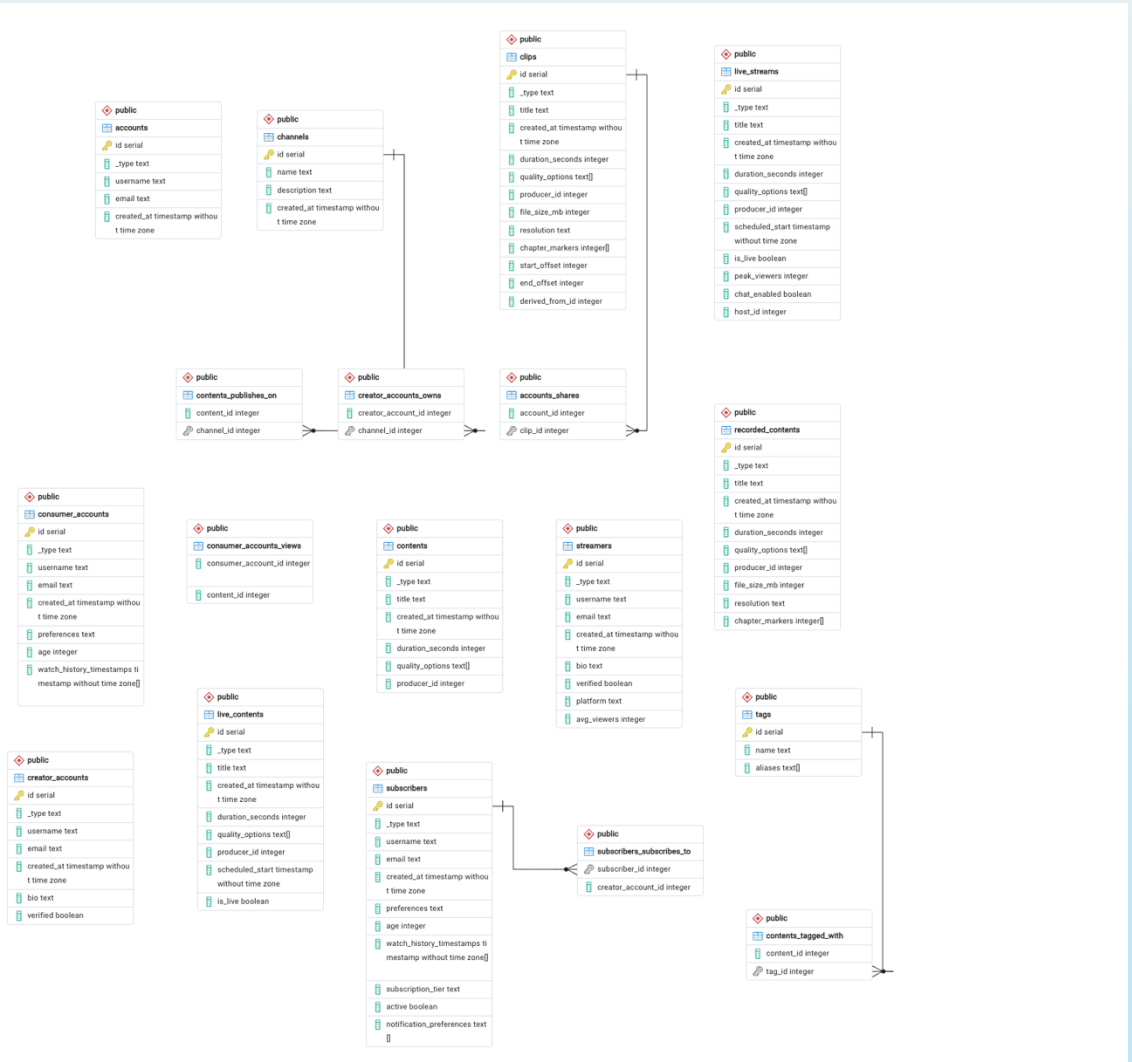
At query time, run the question over multiple E/R-equivalent schemas and compare answers to detect inconsistencies.

User-System Interaction Patterns

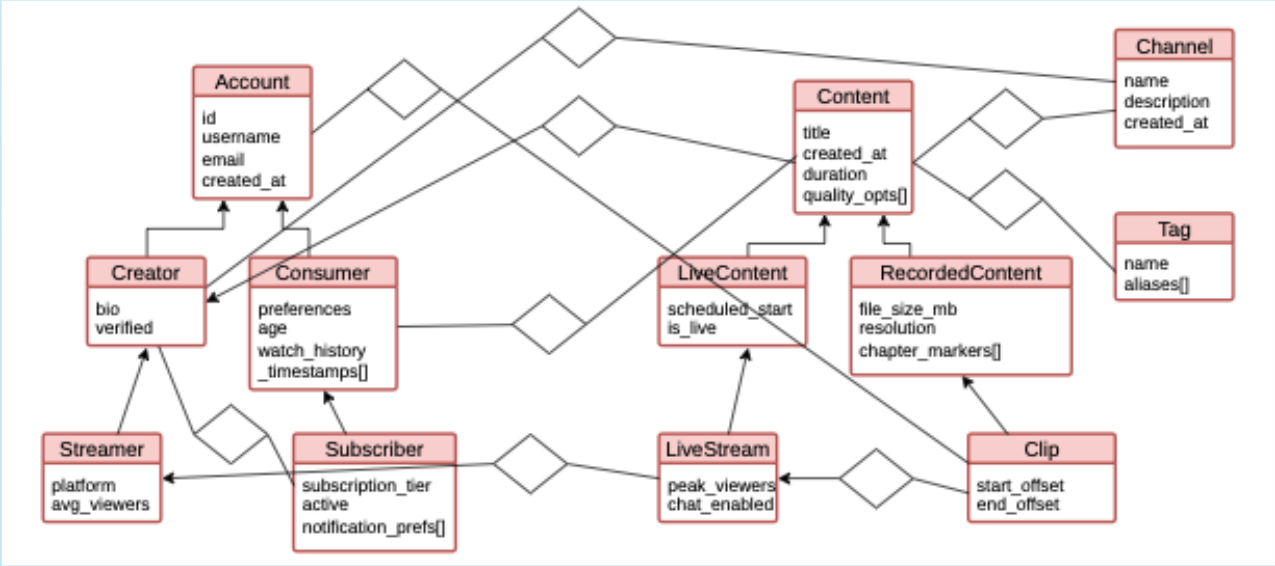


Lack formal representation of the intent →
Cannot remove ambiguity

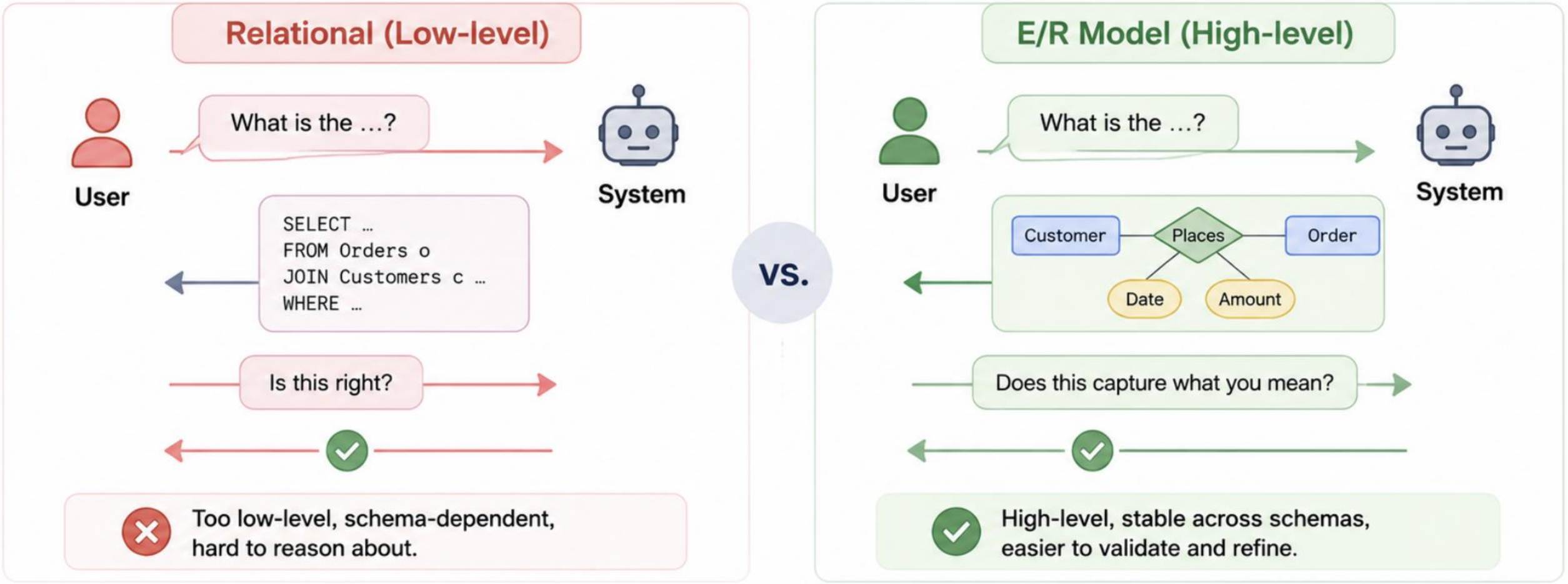
Need to Use Higher Levels of Abstraction



vs



Need to Use Higher Levels of Abstraction



Thanks !!!