EviNorm: Evidence-Guided Schema Normalization for Temporal Tabular Reasoning

Ashish Thanga[1*], Vibhu Dixit[1*], Abhilash Shankarampeta[2], Vivek Gupta[1†]

[1]Arizona State University [2]UC San Diego

AUTOMATED KNOWLEDGE BASE CONSTRUCTION

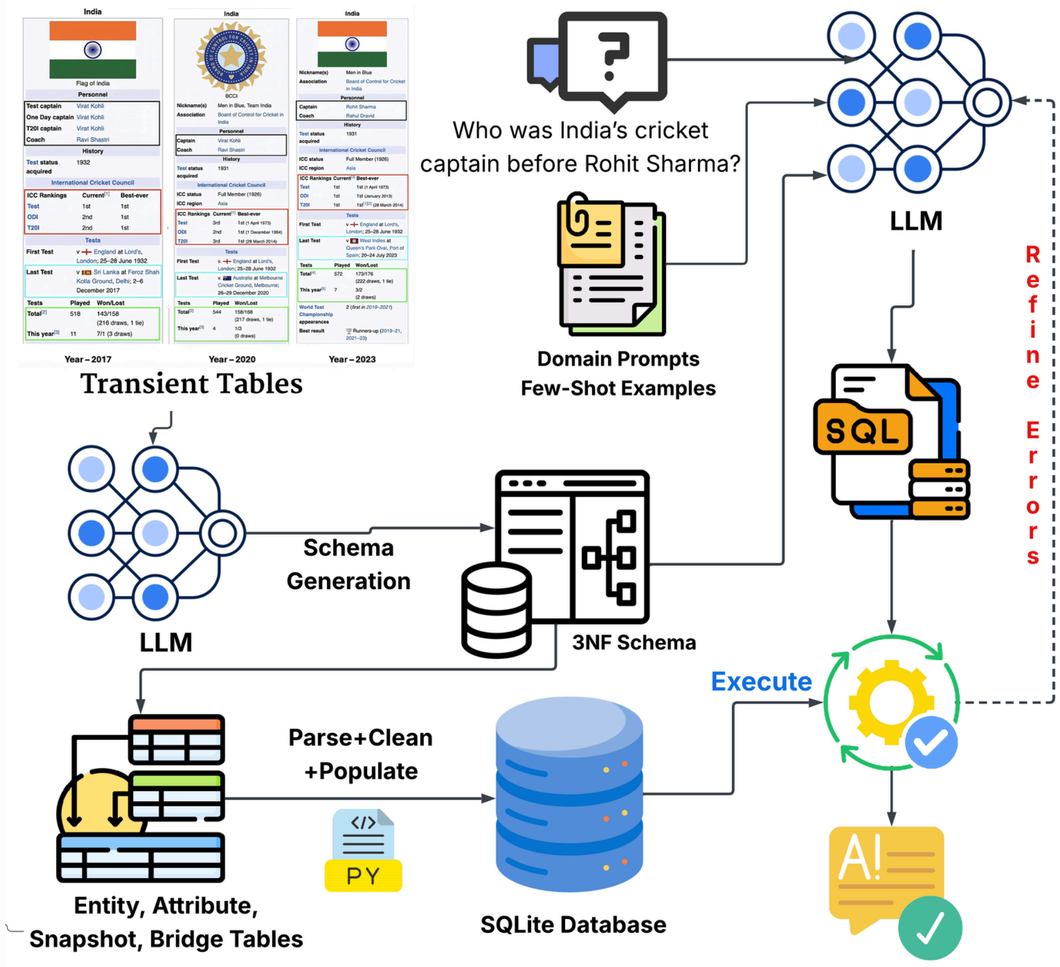
EMNLP2026 BUDAPEST

Motivation

Problem: LLMs struggle with evolving tables; prompting alone achieves just **68.89 EM** on TransientTables.

Our idea: Synthesize a 3NF knowledge base from infobox timelines, then answer queries using SQL.

Why it works: Timestamps and foreign keys enable temporal reasoning, while SQL provides accurate date operations and traceable, executable answers.



Task

TransientTables: 3,000+ questions | 1,200+ entities | 10 domains | 8-12 snaps/entity

Q: *"Who was the coach when Balaji Pai was T20I captain?"* (Gibraltar, cricket_team)

Schema tables used: TeamSnapshots | TeamCaptaincy | Coaches | Players | CaptainRoles

```
SELECT DISTINCT c.coach_name FROM Coaches c
JOIN TeamSnapshots ts ON c.coach_id = ts.coach_id
JOIN TeamCaptaincy tc ON ts.snapshot_id = tc.snapshot_id
JOIN Players p ON tc.player_id = p.player_id
JOIN CaptainRoles cr ON tc.captain_role_id = cr.captain_role_id
WHERE cr.role_title = 'T20I Captain'
  AND p.player_name = 'Balaji Pai' AND ts.team_id = <Gibraltar>;
```

Coaches and captains match on snapshot_id to ensure overlapping tenures.

EviNorm Pipeline

1 Schema synthesis (once per domain, offline)

One prompted call over **2–3 example timelines** returns a **3NF schema** with four table types: **entity, attribute, snapshot, bridge**.

2 Population (once per domain, offline)

Scripts clean and insert each JSON snapshot, one per snapshot_id.

3 Schema-guided SQL (per question)

The LLM writes SQL from the schema plus 10–15 execution-validated examples. SQLite runs it.

Experiments and Results

Query model	IRE+CoT	Flash	70B	8B
Gemini-2.0-Flash	48.98	80.39	72.70	60.37
Qwen-2.5-7B	30.40	77.84	75.30	60.91
Llama-3.1-8B	29.83	78.08	78.70	61.78
Llama-3.3-70B	41.08	69.86	79.52	64.79
GPT-4o-mini	44.53	69.56	78.54	62.88
Gemini-2.5-Pro	68.89	68.25	78.51	63.13
Column mean	43.95	*74.00*	*77.21*	*62.31*

Exact match (EM). Rows: query model. Columns: schema source.

80.39 EM — **Beats SOTA by +11.5**
Best prompting: 68.89

14.7-20.0 pts — **The schema is the lever**
vs 4.4-12.1 from changing the query model

16/18 Configs — **Beat the prompting baseline**
exceptions: Gemini-2.5-Pro, whose baseline is already strong.

Why Some Schemas Work Better

Over-normalization is relationally valid but generatively costly: Each additional join increases the risk of query errors.

Excessive fragmentation increases SQL generation errors (Economy domain)

2 Tables — **Gemini 2.5 Flash**
Schemas yield the single best cell

18 Tables — **Llama-3.1-8B**
Weakest generator

Over-normalized: Gemini 2.5 Pro's tables force a self-join, doubling the query.
What helps (hypotheses): fewer tables, specific names, one shared timestamp key

Error Analysis

70% of residual failures stem from data understanding rather than SQL or schema generation.



Data understanding: Entity mismatches ("Paul Kihara Kariuki" vs. "P.K. Kariuki"), and incorrect mappings (wrong captain).
SQL generation: Aggregate misuse dominates like ORDER BY without proper GROUP BY.
Schema generation: Even rare errors propagate across downstream queries.

Conclusion

Invest model capability in schema construction, then leverage cheaper query models:

- Schema impact: Schema changes improved accuracy far more than query-model changes.
- Transferability: Strong schemas performed well across all six query models.
- Cost efficiency: Gains came with nearly unchanged token costs.

Future Directions

- **Cross-domain reasoning:** Join schemas across domains to answer questions linking events and data.
- **Beyond Wikipedia:** Extend schema synthesis to noisier corporate, government, and scientific datasets.
- **Self-improving schemas:** Automatically validate and refine schemas based on query performance.

SCAN FOR PAPER

CONTACT: vgupt140@asu.edu