FD-NL2SQL: Feedback-Driven Clinical NL2SQL that Improves with Use

*Suparno Roy Chowdhury[1] *Tejas Anvekar[1] *Manan Roy Choudhury[1]
Muhammad Ali Khan[2] Kaneez Zahra Rubab Khakwani[2] Mohamad Bassam Sonbol[2]
†Irbaz Bin Riaz[2] †Vivek Gupta[1]
Arizona State University[1] Mayo Clinic[2]
Project Page Demo Video Code
riaz.irbaz@mayo.edu, vgupt140@asu.edu

Abstract

Clinicians exploring oncology trial repositories often need ad-hoc, multi-constraint queries over biomarkers, endpoints, interventions, and time, yet writing SQL requires schema expertise. We demo FD-NL2SQL, a feedback-driven clinical NL2SQL assistant for SQLite-based oncology databases. Given a natural-language question, a schema-aware LLM decomposes it into predicate-level sub-questions, retrieves semantically similar expert-verified NL2SQL exemplars via sentence embeddings, and synthesizes executable SQL conditioned on the decomposition, retrieved exemplars, and schema, with post-processing validity checks. To improve with use, FD-NL2SQL incorporates two update signals: (i) clinician edits of generated SQL are approved and added to the exemplar bank; and (ii) lightweight logic-based SQL augmentation applies a single atomic mutation (e.g., operator or column change), retaining variants only if they return non-empty results. A second LLM generates the corresponding natural-language question and predicate decomposition for accepted variants, automatically expanding the exemplar bank without additional annotation. The demo interface exposes decomposition, retrieval, synthesis, and execution results to support interactive refinement and continuous improvement.

1 Introduction

Clinical trial databases are central to modern oncology research and drug development. Public registries such as ClinicalTrials.gov (Zarin et al., 2011) and institutional repositories contain rich structured data, including trial phase, biomarkers, eligibility criteria, endpoints, recruitment status, and sponsor information, supporting competitive intelligence, hypothesis generation, regulatory planning, and translational research. As oncology increasingly shifts toward biomarker-driven and precision trials, efficient access to this structured data is critical.

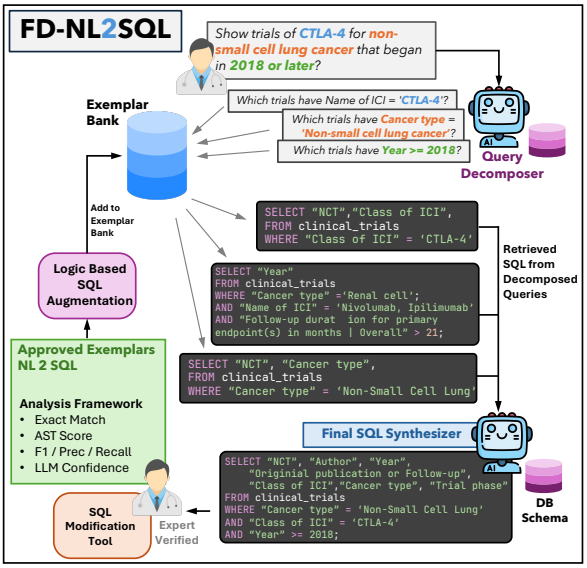

Figure 1: A clinician question is decomposed into schema-aligned predicate sub-questions; for each predicate, semantically similar expert-approved exemplars are retrieved; This guides schema-grounded SQL synthesizer. Users can edit and approve the final SQL to update the exemplar bank. To expand coverage with minimal annotation, approved SQL is augmented by a single atomic mutation (e.g., operator or column substitution) and retained only if it returns non-empty results. A second LLM back-translates augmented SQL into a NL-question & predicate sub-questions, new samples are added to the bank for continual improvement.

Yet these databases remain difficult to query. Access typically requires SQL expertise and detailed schema knowledge, Schemas are often complex, spanning multiple relational tables for eligibility, interventions, endpoints, & disease ontologies. Clinicians and translational researchers, though domain experts, are rarely trained in database querying, leading to analyst-mediated workflows that slow iterative exploration. In the high-stakes setting of oncology drug development, this friction directly affects research velocity & decision quality.

Existing tools only partially mitigate this

*These authors contributed equally.
†Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 764–773

gap. Registry interfaces rely on keyword-based search (Zarin et al., 2011), which cannot reliably enforce structured multi-constraint filtering (e.g., biomarker + phase + endpoint + recruitment criteria). Business intelligence dashboards (Chen et al., 2012) provide predefined reports but remain rigid and cannot cover the combinatorial space of exploratory clinical queries.

Natural Language to SQL (NL2SQL) systems have advanced substantially in recent years, beginning with neural approaches such as Seq2SQL (Zhong et al., 2017) & large-scale benchmarks such as Spider (Yu et al., 2018). More recent work has introduced schema-aware reasoning (e.g., RAT-SQL (Wang et al., 2020)) & constrained decoding for syntactic validity (e.g., PICARD (Scholak et al., 2021)). Large Language Models (LLMs) further demonstrate strong in-context semantic parsing capabilities (Brown et al., 2020). However, these systems are largely designed for general-purpose benchmarks & do not explicitly incorporate domain-aware constraint decomposition, exemplar retrieval grounded in clinical schemas, or interactive feedback loops tailored to high-stakes biomedical querying. In specialized domains such as oncology, naive generation without schema-aligned grounding & domain-specific retrieval can lead to brittle or clinically implausible queries.

To address these limitations, we introduce a domain-aware NL2SQL system designed specifically for oncology clinical trial databases. Our approach integrates three core components. First, we perform LLM-guided self-evolving decomposition of a user's question into atomic, schema-aligned sub-questions that each correspond to a filterable predicate. Second, we retrieve semantically similar seed exemplars using Sentence-BERT embeddings (Reimers and Gurevych, 2019), enabling structured grounding in prior validated query patterns. Third, we perform retrieval-guided SQL synthesis with controlled decoding and post-processing to ensure structural validity and constraint satisfaction. This decomposition-retrieval-synthesis architecture improves robustness by aligning generation with both schema structure and domain-specific precedent.

Beyond static query translation, our system operates as a *living clinical review assistant*. Each generated query can be previewed, refined, and corrected interactively. User feedback on retrieved exemplars and synthesized SQL is incorporated into the seed bank, improving retrieval neighborhoods and generation fidelity over time. This feedback-driven refinement is consistent with emerging paradigms of interactive and adaptive language model systems (Brown et al., 2020), but is operationalized here in a structured, database-grounded clinical setting.

From a clinician's perspective, this approach substantially reduces dependency on technical needs, accelerates hypothesis testing, & enables real-time, multi-constraint exploration of trial criterias. By combining domain-aware decomposition, exemplar-guided synthesis, and iterative feedback, the system bridges the gap between oncology expertise and structured data access, transforming static registries into interactive analytical tools.

Finally, our main contributions are:

- A schema-aware, predicate-level decomposition strategy that improves robustness of clinical NL2SQL in complex oncology databases.
- A retrieval-guided SQL synthesis pipeline that grounds LLM generation in expert-verified exemplars for reliable query construction.
- Feedback-driven, self-evolving clinical trial query assistant that improves continuously through clinician interaction.

2 Related Work

Recent advances in text-to-SQL have shifted from supervised semantic parsing toward large language model (LLM) prompting and in-context learning. DIN-SQL (Pourreza and Rafiei, 2023) demonstrates that decomposed prompting improves SQL generation by breaking complex questions into intermediate reasoning steps. Similarly, execution-guided decoding (Wang et al., 2018) improves robustness by validating generated queries against database constraints during generation. These approaches highlight the importance of structural grounding when generating executable SQL. Retrieval-based prompting has also emerged as an effective strategy for improving LLM reasoning. In-context example selection significantly affects downstream generation quality (Liu et al., 2022), and retrieval-augmented generation (RAG) (Lewis et al., 2020) shows that grounding outputs in external memory improves reliability. Our method extends this paradigm by retrieving semantically similar question-SQL exemplars using dense embeddings and conditioning synthesis on predicate-

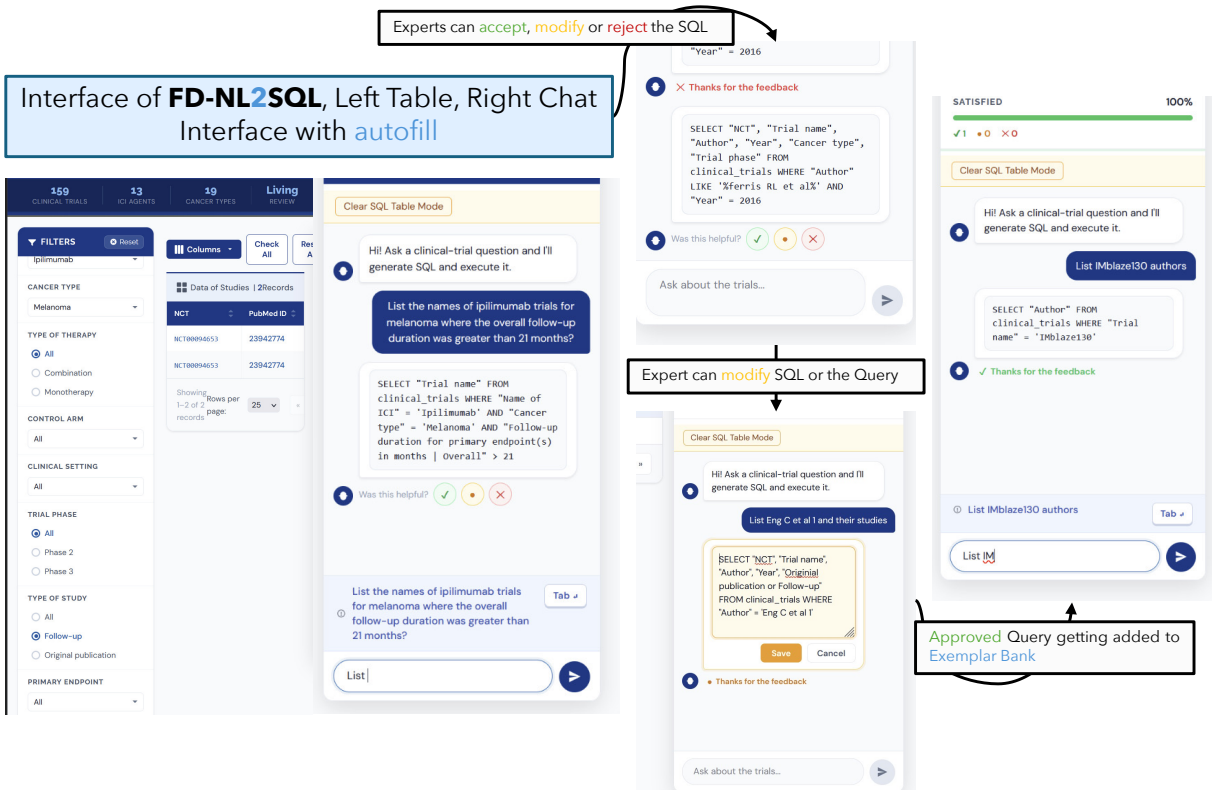

Figure 2: FD-NL2SQL demo UI and feedback loop. Clinicians issue a natural-language query in the chat interface (right) and view executed results in the table view (left). The system shows the generated SQL, which an expert can *accept*, *modify*, or *reject*; accepted / edited queries are saved back to the exemplar bank to improve future retrieval and synthesis (autofill supports rapid refinement).

aligned decompositions, rather than relying solely on flat prompt demonstrations.

Within the biomedical domain, pretrained scientific language models such as SciBERT (Beltagy et al., 2019) have demonstrated gains on domain-specific NLP tasks. However, prior work primarily focuses on unstructured text understanding rather than structured clinical database querying. Our system bridges biomedical language understanding with schema-aware SQL synthesis, enabling clinician-driven, multi-constraint exploration of oncology clinical trial databases.

3 FD-NL2SQL

3.1 System Architecture and Workflow

Illustrated in Figure 1, FD-NL2SQL is an interactive NL2SQL assistant for oncology clinical-trial databases. The system follows a modular pipeline combining LLM-based reasoning with retrieval over an evolving exemplar bank and lightweight programmatic checks. The demo UI exposes intermediate artifacts, retrieved exemplars, synthesized SQL, and results to support transparent refinement and feedback-driven improvement.

Resources. We assume a SQLite database $\mathcal{D}$ with schema metadata and an exemplar bank $\mathcal{S} = \{(s_j, y_j)\}_{j=1}^{M}$ of expert-approved NL2SQL pairs. We pre-compute sentence embeddings $\mathbf{e}(s_j)$ for all s_j and maintain an index for fast $topk$ retrieval.

1) Schema grounding. Before generation, we introspect $\mathcal{D}$ to build a schema dictionary (tables, columns, types, and join keys). This schema context is injected into prompts and used for post-generation validation (e.g., column existence and join feasibility).

2) Decomposition-retrieval. Given a user question x, an LLM produces a schema-aligned, `WHERE`-oriented decomposition $\mathcal{X}(x) = \{x_1, \ldots, x_n\}$ where each x_i targets one atomic predicate (column, operator, value). For each x_i, we retrieve the top-k_r nearest exemplars from $\mathcal{S}$ using cosine similarity in embedding space, storing $(s_j, y_j, \text{score}(x_i, s_j))$. To reduce prompt noise, we additionally extract a compact `WHERE`-pattern hint from each retrieved SQL when available.

3) Retrieval-guided SQL synthesis & execution. A second LLM synthesizes the final SQL $\hat{y}$ con-

ditioned on (i) x, (ii) $\mathcal{X}(x)$, and (iii) the retrieved exemplar bundles $\{(x_i, \mathcal{N}_i)\}_{i=1}^{n}$. The model is instructed to use retrieved SQL as structural templates while satisfying all constraints in x and avoiding irrelevant literal copying. We parse the output into a single executable `SELECT` / `WITH` statement, apply lightweight guards (read-only policy, schema checks, timeout), and execute $\hat{y}$ against SQLite to render results in the UI.

3.2 Feedback and Exemplar Bank Expansion

4) Expert approval. The Figure 2 interface allows users to edit generated SQL. If an expert approves the corrected query $y^\star$ for question x, the pair $(x, y^\star)$ is appended to the exemplar bank $\mathcal{S}$ and embedded for future retrieval.

5) SQL augmentation for bank growth. To expand exemplar coverage with minimal manual annotation, FD-NL2SQL augments approved queries in two sub-steps:

a) **Logic-based SQL mutation.** Starting from an approved query $y^\star$, we apply exactly one atomic transformation, such as (i) operator change (= → >=, `LIKE`, etc.), (ii) column substitution within a compatible type group, or (iii) controlled value edits (e.g., year thresholds). The mutated query $\tilde{y}$ is retained only if it executes successfully and returns a non-empty result on $\mathcal{D}$; otherwise it is discarded.

b) **NL Back-Translation** For each retained $\tilde{y}$, a separate LLM generates (i) a natural-language question $\tilde{x}$ consistent with $\tilde{y}$ and (ii) predicate-level sub-questions $\mathcal{X}(\tilde{x})$ aligned with the mutated constraints. The resulting pair $(\tilde{x}, \tilde{y})$ (and optional decomposition) is added back to $\mathcal{S}$, improving retrieval and synthesis coverage over time.

4 Experimental Setup

4.1 Dataset Construction

Seed set. We start with 500 seed questions authored by a Mayo Clinic oncology scientist to reflect realistic evidence-review queries over the IOTOX[1] table (e.g., cancer type, class of ICI, trial phase, endpoints / follow-up, temporal filters). Each question is paired with a gold SQLite query, verified by execution, and stored as JSON (question, SQL, optional metadata). The seed set serves as (i) the initial exemplar bank for retrieval and (ii) the pool for few-shot demonstrations.

Programmatic benchmark expansion. To evaluate generalization beyond the seed distribution, we create benchmark by applying a *single* atomic transformation to each seed pair $(x, y) : (\tilde{x}, \tilde{y}) = T(x, y)$. where T edits either the projection (`SELECT`) or constraints (`WHERE`) while preserving the overall query intent. We generate the following variant types (counts in parentheses): **1) Projection edits:** keep only 2 selected columns (275); drop one selected column (238). **2) Predicate dropping:** drop one `WHERE` condition (179); keep only one `WHERE` condition (115); drop 2 `WHERE` conditions (99); remove `WHERE` entirely (20). **3) Value edits:** swap a text equality value (40); relax a numeric threshold (34).

Validity filtering and de-duplication. Each generated SQL $\tilde{y}$ is executed against SQLite. We retain a variant only if it (i) is read-only (`SELECT/WITH`), (ii) executes without error, and (iii) returns a non-empty result. We then normalize and de-duplicate SQL to remove repeats. Finally, we use an open-source LLM `Gemma3-27B-it` (Gemma, 2025) to generate natural-language query using Prompt A for each augmented SQL. This constituted to 1500 NL2SQL samples & authors of this paper verified its correctness.

Real-world query diversity. To assess whether benchmark expansion preserves high-level intent despite starting from 500 expert-authored seed questions, we compute cosine similarity between each source question and its generated counterpart using `Qwen3-Embedding-8B` (Zhang et al., 2025). Averaging across original-question groups yields an average per-original mean cosine similarity of **0.84**, suggesting that the generated questions preserve the underlying clinical intent while expanding query diversity.

4.2 Baselines and Benchmark Settings

We compare FD-NL2SQL against four representative NL2SQL baselines: 1) Zero-Shot, 2) Few-Shot, 3) Chain-of-Thoughts, using both open-source and closed source LLMs (`Gemma3-27B-it` (Gemma, 2025), `Qwen3-30B-A3B-Instruct-2507` (Qwen, 2025), and `gpt-5-nano/mini` (GPT-5, 2025)) and 4) `SQL-R1-14B` (MA et al., 2025) finetuned reasoning based model trained with reinforcement-learning objectives. We use deterministic de-

[1] https://iotox.living-evidence.com/

coding where supported (e.g., temperature 0.0). All prompts of baselines (Prompt C, Prompt B, Prompt D) and our method (Prompt E, Prompt F) are given in Appendix A. `SQL-R1-14B` does not contain settings that allow us to test with Few-Shot prompting which is why it is not included as part of the benchmark.

4.3 Evaluation Metrics

We report execution-based correctness on the same SQLite database, plus lightweight structural / text scores. Specifically, we compute: (i) **eEM** (exact match of predicted vs. gold result multisets after column alignment and value canonicalization), (ii) **eF1** (soft row-overlap F1 after one-to-one row matching with numeric tolerance and token overlap for strings), (iii) **CHRF** (computed on the matched (row-aligned) predicted vs. gold result strings) (iv) **AST**[2] (clause-weighted token-F1 over `SELECT` / `FROM` / `WHERE` with weights $(w_s, w_w, w_f) = (0.5, 0.4, 0.1)$ when parsing succeeds), (v) LLM **Conf** from `structured_logprobs`[3] over the generated SQL tokens, and We additionally log diagnostic flags for non-read-only SQL, multiple statements, and `LIMIT` without `ORDER BY`.

5 Results and Discussions

In this section, we evaluate the performance of FD-NL2SQL and analyze its behavior across realistic oncology clinical trial queries. We report quantitative results on execution-based metrics and examine how decomposition, retrieval grounding, and feedback-driven augmentation contribute to robustness and continuous improvement. We further discuss practical implications and observed limitations in clinician-facing usage.

RQ1: Why does FD-NL2SQL perform better? Table 1 shows that FD-NL2SQL consistently improves execution-based correctness over standard prompting (zero-shot, few-shot, CoT) across all backbone models, with the largest gains for backbones such as `Qwen3` and `gpt-5-nano`. The gains are reflected across complementary metrics: **eEM** increases when the predicted query returns the *exact* gold result set; **eF1** increases when partial correctness improves (e.g., more constraints are satisfied even if not all); and higher **AST** indicates the generated SQL follows the gold structure more closely, reducing schema / logic drift. This translates into stronger **HM** scores, indicating that FD-NL2SQL improves overall in balance of correctness, structure, and confidence.

Prompting-only baselines are brittle for multi-constraint clinical questions: zero-shot frequently drops predicates or selects incompatible columns, few-shot remains sensitive to which demonstrations are shown, and CoT often degrades execu-

Table 1: **Quantitative results** across prompting strategies, finetuned model and FD-NL2SQL (Ours). We report CHRF, eEM, eF1, AST, and Conf; *HM* denotes the harmonic mean of {Conf, eF1, eEM}. Green marks the best score *within a given strategy for each model*, red marks the worst score within that strategy for each model, and underlines denote the best score overall across all models and strategies for a metric. Note `gpt-5n` depicts `nano`, and `gpt-5m` for `mini` variant. We also experiment with empty-queries from 390 Mayo clinician-authored questions whose gold SQL returns the empty set, measuring robustness when the correct answer is no result. CHRF is omitted because it is uninformative when both predicted and gold queries return the empty set, leading to consistent scores of 100

(A) Full 1,500-query Benchmark

Strategy	Models	CHRF	eEM	eF1	AST	Conf	*HM*
Zero-Shot	Qwen3	81.82	26.90	46.51	72.54	39.01	35.58
	Gemma3	84.07	29.48	39.95	59.41	65.69	40.44
	SQL-R1	63.78	03.32	01.75	41.78	10.39	03.10
	gpt-5n	89.13	11.60	18.87	52.58	–	–
	gpt-5m	**94.64**	38.33	47.36	51.21	–	–
Few-Shot	Qwen3	83.02	25.60	52.95	80.43	34.70	34.58
	Gemma3	91.32	**30.85**	**45.72**	85.47	53.16	41.04
	gpt-5n	89.58	15.80	31.97	75.53	–	–
	gpt-5m	94.52	31.27	46.85	83.03	–	–
CoT	Qwen3	77.96	21.65	29.73	85.21	00.00	00.00
	Gemma3	77.89	13.47	34.88	00.00	00.25	00.73
	SQL-R1	50.24	02.85	03.35	49.15	01.69	02.42
	gpt-5n	86.93	13.67	33.47	77.56	–	–
	gpt-5m	86.28	10.33	19.87	82.02	–	–
Ours	Qwen3	**92.02**	**40.93**	**55.57**	**86.36**	**66.30**	**52.16**
	Gemma3	**92.62**	**32.60**	44.48	**86.21**	**70.48**	**44.55**
	gpt-5n	**90.89**	**32.53**	**50.25**	**86.32**	–	–
	gpt-5m	92.68	**39.20**	**55.85**	**87.14**	–	–

(B) Empty-query stress test.

Strategy	Model	eEM	eF1	AST	Conf	*HM*
Zero-Shot	Qwen3	95.38	95.38	76.61	38.39	63.80
	Gemma3	95.13	95.13	59.80	73.88	86.81
Few-Shot	Qwen3	95.38	95.38	88.16	33.88	59.42
	Gemma3	92.05	92.05	91.44	55.06	75.21
CoT	Qwen3	50.26	50.26	90.95	00.00	00.00
	Gemma3	93.59	93.59	89.40	01.01	02.97
Ours	Qwen3	**95.64**	**95.64**	**93.51**	**61.49**	**80.70**
	Gemma3	**96.15**	**96.15**	**93.08**	**75.44**	**88.09**

[2] sqlglot
[3] structured-logprobs

tion by introducing extra noise and incorrect intermediate assumptions that leak into the final SQL. The reasoning-finetuned baseline `SQL-R1` performs particularly poorly in our setting, which we attribute to domain/schema mismatch: oncology trial tables use domain-specific fields and naming conventions, so plausible-but-ungrounded SQL (guessed columns, joins, or values) can parse yet execute to the wrong results. FD-NL2SQL addresses these failure modes by enforcing *schema-aligned predicate decomposition* (reducing omission) and *in-domain exemplar grounding* (retrieved expert-approved templates), which together stabilize generation and improve both execution and structure.

RQ2: Why such tool is necessary in medical domain? Clinical evidence review often requires exploratory querying over structured trial attributes. Translating these requests into database queries requires both SQL proficiency and detailed schema knowledge, which is unrealistic for clinicians and researchers under time constraints. Our baseline results show that *"just prompt an LLM"* is not a reliable substitute: even strong general models can produce partially correct SQL yet fail exact execution, which is problematic when precise filtering and reproducible counts matter. Because clinical trial databases also evolve over time, static prompts and one-off engineering are brittle, motivating a system that lowers the expertise barrier while remaining robust to schema complexity and domain-specific terminology.

RQ3: What is the practical utility of FD-NL2SQL beyond leaderboard? The primary objective is *executable SQL*, which is inherently reproducible, and auditable properties that matter in medical workflows more than a natural-language answer alone. The system's trace (decomposition $\rightarrow$ retrieved exemplars $\rightarrow$ synthesized SQL $\rightarrow$ results) supports interactive refinement: users can see which constraints were extracted, which prior exemplars influenced the query, and what exactly will be executed. This directly addresses common clinician needs such as *"tighten this criterion," "drop this filter,"* or *"change the endpoint window,"* without requiring users to learn schema details from scratch. The results table also suggests an important practical pattern: FD-NL2SQL narrows performance gaps between backbones, enabling smaller/cheaper models to approach the reliability of larger models when paired with decomposition and exemplar guidance. In deployed settings, this can translate into lower latency and cost while preserving usability.

RQ4: How does FD-NL2SQL improve reliability and trust? Reliability in medical-domain NL2SQL is not only about average accuracy, but about avoiding silent failures and enabling verification. We therefore emphasize execution-based metrics (eEM / eF1) and structural checks (AST-Sim), and we log diagnostic flags for unsafe/non-read-only SQL, multiple statements, and `LIMIT` without `ORDER BY`. These safeguards reduce the chance that a model output is executed in an unsafe or misleading way. In addition, we report **LLM confidence**, which provides a likelihood signal that can be used to triage queries for review (e.g., low-confidence generations may warrant user confirmation or stronger retrieval evidence). Finally, the feedback loop makes the system more dependable over time: expert-approved SQL is added to the exemplar bank, and single-step SQL mutations are retained only if they execute and return non-empty results before being back-translated into new NL2SQL exemplars. This combination of human approval, database-grounded filtering, and transparent intermediate artifacts supports a "living" assistant that can adapt to evolving clinical trials while remaining inspectable and reproducible.

6 Conclusion

We presented FD-NL2SQL, a feedback-driven clinical NL2SQL demo system for oncology trial databases that couples schema-aligned predicate decomposition with exemplar-guided SQL synthesis. Across multiple backbone models, FD-NL2SQL improves execution-based correctness and structural fidelity over standard prompting, demonstrating that retrieval grounding and constraint-level decomposition are more reliable than prompting alone for multi-constraint clinical queries. Beyond accuracy, the system is designed for real evidence-review workflows: it produces executable, auditable SQL grounded in traceable evidence, FD-NL2SQL continuously improves with use by incorporating approved queries and by safely expanding its exemplar bank via single-step SQL mutations validated and LLM back-translation. Together, this enables a practical "living" assistant that lowers the barrier to structured trial exploration while maintaining transparency and reproducibility required in the medical domain.

Acknowledgment

This research was supported by the Mayo Clinic and Arizona State University Alliance for Health Care Collaborative Research Seed Grant Program (Award ID: AWD00041508; Sponsor Award ID: ARI-358187) for the project: 'Artificial intelligence-assisted digital, living, interactive clinical practice guidelines for cancer providers and patients.' The frontend for FD-NL2SQL was developed by Sherwin Vishesh Jathanna who created the design mapped from the original Iotox clinical review trials.

Limitations

Our benchmark is built from 500 expert-authored seed questions with controlled single-edit variants, enabling targeted stress testing but not the full diversity of real clinician queries, such as complex joins, nested SQL, richer aggregations, or broader linguistic variation. Exemplar expansion retains only augmented SQL that executes and returns non-empty results, which may bias coverage toward frequent conditions and simpler query patterns. Finally, confidence from `structured_logprobs` is not uniformly available across model APIs, and our evaluation is limited to offline SQL correctness rather than prospective user studies or formal deployment assessments.

Ethics Statement

FD-NL2SQL is intended for clinician-facing exploration of oncology clinical trial repositories in research and evidence review, not for patient-specific diagnosis or treatment decisions. It operates on publicly available Mayo Clinic clinical trial data and a derived SQLite database containing no patient-level records, PHI, or PII; seed questions were authored by Mayo Clinic collaborators and contain no sensitive data.

Because LLMs may generate incorrect or incomplete SQL, the system emphasizes transparency and safety by producing executable, auditable SQL, exposing intermediate reasoning steps, and applying safeguards such as read-only constraints, schema validation, and timeouts. Only expert-approved SQL edits are added to the exemplar bank, while automatic augmentation retains only variants that execute successfully and return non-empty results, which may bias coverage toward more frequent query patterns.

Any future institutional deployment should include access controls, audit logging, and safeguards against entering sensitive information into free-text interfaces. The system is intended to augment, not replace, expert clinical judgment.

References

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Hsinchun Chen, Roger H. L. Chiang, and Veda C. Storey. 2012. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188.

Team Gemma. 2025. Gemma 3.

Team GPT-5. 2025. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

Peixian MA, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. SQL-r1: Training natural language to SQL reasoning model by reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: decomposed in-context learning of text-to-sql with self-correction. In *Proceedings of the 37th*

International Conference on Neural Information Processing Systems, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Team Qwen. 2025. Qwen3 Technical Report. *Preprint*, arXiv:2505.09388.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *Preprint*, arXiv:1807.03100.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Deborah A Zarin, Tony Tse, Rebecca J Williams, Robert M Califf, and Nicholas C Ide. 2011. The clinicaltrials.gov results database — update and key issues. *The New England journal of medicine*, 364(9):852–860.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *Preprint*, arXiv:1709.00103.

A Prompts Details

Prompt A: SQL-2-NL

```
Read the original question, then rewrite the following SQL
      ↪  to a question to sound naturally what a
      ↪ clinician would ask while keeping intent exactly
      ↪ the same as the SQL.

Rules:
- Output ONLY one natural-language question on one line.
- Do not mention SQL, table names, databases, or column
      ↪ names.
- Preserve every filter / constraint implied by the SQL (
      ↪ entities, values, thresholds, years, etc.).
- Do not add new constraints and do not remove any
      ↪ constraints.
- Keep clinical-trial wording natural and concise.
- Do NOT use markdown, code fences, lables, or backticks.
- Keep explicit numbers exactly as digits (example: 3,
      ↪ 2018).
- Prefer explicit disease and treatment names rather than
      ↪ abbreviations.

Original question:
{original_question}

Ground-truth SQL:
{sql_text}
{hint_block}
```

Prompt B: Few-Shot NL2SQL

```
<USER>QUESTION: Show bladder PD-L1 trials conducted in
      ↪ 2018 or later.</USER>
<ASSISTANT> RESPONSE: SELECT "NCT", "Author", "Year", "
      ↪ Originial publication or Follow-up", "Class of
      ↪ ICI", "Cancer type", "Trial phase" FROM clinical-
      ↪ trials WHERE "Cancer type" = 'Bladder' AND "Class
      ↪  of ICI" = 'PD-L1' AND "Year" >= 2018;</ASSISTANT
      ↪ >

<USER> QUESTION: Find renal cell trials of nivolumab given
      ↪  in combination therapy.</USER>
<ASSISTANT> RESPONSE SELECT "NCT", "Author", "Year", "
      ↪ Originial publication or Follow-up", "Name of ICI
      ↪ ", "Monotherapy/combination", "Cancer type", "
      ↪ Type of combination", "Trial phase" FROM clinical
      ↪ -trials WHERE "Cancer type" = 'Renal cell' AND "
      ↪ Name of ICI" = 'Nivolumab' AND "Monotherapy/
      ↪ combination" = 'Combination';</ASSISTANT>

<USER> QUESTION: Which melanoma trials used vaccine as the
      ↪  control arm?</USER>
<ASSISTANT> RESPONSE: SELECT "NCT", "Author", "Year", "
      ↪ Originial publication or Follow-up", "Control arm
      ↪ ", "Cancer type", "Control regimen", "Trial phase
      ↪ " FROM clinical-trials WHERE "Cancer type" = '
      ↪ Melanoma' AND "Control arm" = 'Vaccine';</
      ↪ ASSISTANT>

<USER> QUESTION: List head and neck trials requiring a
      ↪ biomarker other than PD-L1 for inclusion.</USER>
<ASSISTANT> RESPONSE: SELECT "NCT", "Author", "Year", "
      ↪ Originial publication or Follow-up", "Is any
      ↪ other biomarker used for inclusion", "Cancer type
      ↪ ", "Primary endpoint" FROM clinical-trials WHERE
      ↪ "Cancer type" = 'Head and Neck' AND "Is any other
      ↪  biomarker used for inclusion" = 'Yes';</
      ↪ ASSISTANT>

{Zero-Shot Prompt}
```

Prompt C: Zero-Shot SQL2NL

```
You are an expert natural language to SQL generator.
The task is to query cancer drug trial studies from the "
      ↪ clinical-trials" table.

Rules:
- Output only the SQL query (no prose, no markdown, no
      ↪ code fences).
- The SQL MUST start with SELECT.
- Use only columns from this table schema: {SCHEMA}
- Put double quotes around EVERY column name exactly as in
      ↪  the schema.
- Use single quotes for string literals.

Column definitions and data guidance from the schema:
- "NCT" (TEXT): trial identifier string, e.g., '
      ↪ NCT02477826', 'NCT00094653', 'NCT00527735'.
- "PubMed ID" (TEXT/ID-like): publication identifier, e.g
      ↪ ., '31562796', '32201234', '20525992'.
- "Trial name" (TEXT): trial label, e.g., 'Checkmate 227',
      ↪  'CASPIAN', 'KEYNOTE-002'.
- .....
- "Included in MA" (TEXT): 'Yes' or 'No'.

Question: {QUESTION}
Generate SQL:
```

Prompt D: Chain of Thoughts NL2SQL

```
You are an expert natural language to SQL generator.
The task is to query cancer drug trial studies from the "
      ↪ clinical-trials" table.

You MUST follow the reasoning scaffold below and SHOW your
      ↪  work.
Use only columns from this table schema: {SCHEMA}

STRICT OUTPUT FORMAT:
- Wrap the user question in: <USER>...</USER>
- Wrap the assistant output in: <ASSISTANT>...</ASSISTANT>
- Inside <ASSISTANT>, output TWO tagged blocks in this
      ↪ exact order:
  1. <THINKING>...</THINKING>
  2. <SQL>...</SQL>

SQL RULES (STRICT):
- The SQL MUST start with SELECT.
- Use only columns from the schema.
- Put double quotes around EVERY column name exactly as in
      ↪  the schema.
- Use single quotes for string literals.
- End with a semicolon.
- No extra text outside <THINKING> and <SQL>.

REASONING SCAFFOLD TO SHOW (do not skip steps, keep
      ↪ numbering): <THINKING>
1. Normalize terms: Rewrite entities to match schema
      ↪ values (title case for categorical values).
2. Identify intent: State the user intent in one sentence
      ↪ (list/filter trials, etc.).
3. Map to columns: List each extracted constraint and the
      ↪ exact schema column it maps to.
4. Choose constraints: Specify operators (=, >=, IN, LIKE)
      ↪  and how constraints combine (AND/OR).
5. Compose query:
    - List the selected output columns and justify briefly
      ↪  (key identifiers + question-relevant fields).
    - Then write the final SQL in <SQL>.
</THINKING>

GOOD EXAMPLE (CORRECT SCHEMA USE): .......
BAD EXAMPLE (INCORRECT / WHAT NOT TO DO): ......

Now, answer the following.

<USER>QUESTION: {QUESTION}</USER> <ASSISTANT> <THINKING>
      ↪ </THINKING> <SQL> </SQL> </ASSISTANT>
```

Prompt E: SQL Query Decomposition

```
You are an expert SQL query decomposition assistant for
      ↪ the clinical-trials table.

Schema: {SCHEMA}
Question: {QUESTION}

Task:
Decompose the question into atomic, retrieval-friendly sub
      ↪ -questions that map to SQL WHERE predicates.
Each sub-question should represent exactly one filter
      ↪ condition (column, operator, value).
Preserve all key constraints exactly (e.g., cancer type,
      ↪ number of arms, trial phase, control arm, drug
      ↪ name).

WHERE-focused decomposition rules:
- Focus on data points that belong in WHERE conditions.
- Use explicit operators when implied: =, >=, <=, >, <,
      ↪ !=, LIKE/contains.
- Keep literal values explicit (for example: 'Colorectal',
      ↪  >= 3, 'Phase 3').
- If a question has multiple constraints, split into one
      ↪ sub-question per constraint.
- Avoid SELECT/projection/output-only instructions (for
      ↪ example: "show NCT and Author").

Output Rules:
- Do NOT output SQL.
- Do NOT output markdown or code fences.
- Do NOT add explanations.
- Output ONLY valid JSON in this exact schema:
  {"decomposed_queries": ["...", "..."]}

Good decomposition example:
Original question: Which colorectal trials had 3 or more
      ↪ arms and used a multikinase inhibitor as a
      ↪ control?
Output:{"decomposed_queries": [
  "Which trials have Cancer type = 'Colorectal'?",
  "Which trials have Number of arms >= 3?",
  "Which trials have Control arm = 'Multikinase inhibitor
      ↪ '?"]}

Bad decomposition example:
Original question: Which colorectal trials had 3 or more
      ↪ arms and used a multikinase inhibitor as a
      ↪ control?
Bad output: {"decomposed_queries": [
  "Find relevant colorectal studies.",
  "Get trials with enough arms.",
  "Show NCT and Author."]}

Why bad:
- Too vague and not directly aligned to schema WHERE
      ↪ fields.
- Drops exact operators/literals required for retrieval.
- Includes SELECT-style output instruction instead of a
      ↪ filter condition.
```

Prompt F: Final SQL Synthesizer

```
You are an expert natural language to SQL generator. The
      ↪ task is to query cancer drug trial studies from
      ↪ the "clinical-trials" table.

Output rules: only the SQL query (no prose, no markdown,
      ↪ no code fences).

Core rules:
- The SQL MUST start with SELECT.
- Use only columns from this table schema: {SCHEMA}
- Put double quotes around EVERY column name exactly as in
      ↪  the schema.
- Use single quotes for string literals.
- Do not invent columns, values, joins, or predicates not
      ↪ supported by the input evidence.
Evidence priority (highest to lowest):
1) Parent question intent and constraints.
2) Decomposed sub-question predicates.
3) Retrieved seed SQL examples (hints only).

Predicate budget rule:
- The final WHERE clause must include only intent-
      ↪ supported predicates from parent question +
      ↪ decomposition.
- Do not add extra predicates from retrieved seeds.
- If decomposition has N independent constraints, final
      ↪ WHERE should represent those N constraints unless
      ↪  one is contradictory or duplicate.

Synthesis constraints:
- Include a WHERE predicate only if it is clearly
      ↪ supported by parent question/decomposition.
- Do NOT copy unrelated literals from retrieved examples.
- Prefer canonical dataset values when available in
      ↪ retrieved examples (e.g., 'Renal cell' over free-
      ↪ form variants).
- If input says PD1/PD-L1/CTLA-4 class, map to "Class of
      ↪ ICI" (not "Name of ICI").
- If input names a specific drug (e.g., Nivolumab,
      ↪ Pembrolizumab), map to "Name of ICI".
- Distinguish biomarker fields:
  - "Is PD-L1 positivity inclusion criteria"
  - "Is any other biomarker used for inclusion"
- Distinguish combination fields:
  - "Type of combination" when question asks chemo/anti-
      ↪ VEGF/vaccine type
  - "Monotherapy/combination" only for mono vs combination
      ↪  status
  - "Treatment regimen" only for agent/regimen names, not
      ↪ combination category
- Avoid over-constraining: do not add extra filters beyond
      ↪  what is asked.

Strict value policy for WHERE predicates:
- For categorical columns, use ONLY values from the
      ↪ allowed list below.
- Do not invent or paraphrase category values.
- Prefer canonical literals exactly as listed.
- If a user phrase is a synonym, map it using the
      ↪ normalization rules below.

.........

Retrieved examples:
Example 1
Question: Find Bladder cancer trials using PD1 where PD-L1
      ↪  positivity was not an inclusion criterion.
SQL:
SELECT "NCT", "Author", "Year", "Original publication or
      ↪ Follow-up", "Class of ICI", "Cancer type", "Trial
      ↪  phase"
FROM clinical-trials
WHERE "Cancer type" = 'Bladder' AND "Class of ICI" = 'PD1'
      ↪  AND "Is PD-L1 positivity inclusion criteria" = '
      ↪ No';

.......

Question: {QUESTION}
Generate SQL:
```