Probing Ambiguity: Adversarial Intent Resolution for Robust Text-to-SQL

Youbing Yin, Xujun Peng, Muhammad S. Uddin, Erin Babinsky, Daben Liu, Anoop Kumar
Capital One
{youbing.yin, xujun.peng, muhammad.uddin,
erin.babinsky, daben.liu, anoop.kumar}@capitalone.com

Abstract

Text-to-SQL systems typically assume a one-to-one mapping between natural-language queries and correct SQL translations. In practice, however, many queries are inherently ambiguous, admitting multiple valid interpretations that yield distinct outputs. We introduce Adversarial Intent Resolution (AIR), a multi-agent framework that actively probes and resolves such ambiguity without any task-specific training. AIR first constructs competing interpretations, verifies through SQL execution that they produce genuinely different results, and generates targeted clarification questions grounded in the observed data differences. Evaluating on leading benchmarks for ambiguous questions, AIR demonstrates strong performance, efficiency, and robust cross-benchmark transferability. On the AMBROSIA benchmark under schema-only context—a realistic production setting where exposing the full database is infeasible—AIR outperforms IntentRL, the state-of-the-art reinforcement learning model, across all metrics (e.g., F1 0.679 vs. 0.618). Crucially, AIR exhibits only ∼7% performance degradation without the full database, compared to ∼20% for IntentRL. On the AmbiQT benchmark, where ambiguity arises from schema design rather than linguistic structure, AIR achieves higher F1 under both context settings without any benchmark-specific adaptation.

1 Introduction

Text-to-SQL systems have made remarkable progress, with recent LLM-based approaches achieving over 85% execution accuracy on benchmarks like Spider (Yu et al., 2018; Lei et al., 2025) and BIRD (Li et al., 2024b) through in-context learning (Gao et al., 2023; Pourreza and Rafiei, 2024) and fine-tuning (Li et al., 2024a). These advances bring closer the promise of natural-language database interfaces, enabling non-technical users to query complex schema without writing SQL. Yet this progress rests on a critical assumption: *that each natural-language question maps to exactly one correct SQL translation.*

In practice, this assumption is fundamentally at odds with how humans formulate questions. Consider: "Show $150 wellness programs and day spas." Should the price filter apply to both wellness programs and day spas, or only wellness programs? Under one interpretation the answer is two rows; under the other, six rows, a very different result set. This is not an edge case. Ambiguous queries routinely admit multiple valid SQL translations that return different results (Bhaskar et al., 2023; Saparina and Lapata, 2025a).

Yet the dominant paradigm maps each question to a single SQL query and declares success (Yao et al., 2026; Singh et al., 2026; Liu et al., 2026). Even when systems are explicitly prompted for diverse outputs, they overwhelmingly generate surface-level variants of the top prediction rather than structurally distinct alternatives (Bhaskar et al., 2023). Diversity-promoting decoding strategies such as beam search and nucleus sampling fail to explore genuinely different interpretations. A recent work trains models via reinforcement learning (RL) to generate multiple candidate interpretations (Saparina and Lapata, 2026), but still treats ambiguity as a generation problem rather than a *probing and verification* problem. Moreover, in realistic production settings the full database may not be available as the context to LLM due to privacy, latency, and cost constraints, restricting the LLM to schema-level context only. It remains unclear how these approaches perform when schema-only is used as context to guide disambiguation.

We propose **Adversarial Intent Resolution (AIR)**, a multi-agent framework that requires no task-specific training and resolve ambiguity through *adversarial probing*, actively attempting to construct a competing interpretation. An Anchor agent proposes the most likely interpretation; an

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 55493–55504
October 24-29, 2026 ©2026 Association for Computational Linguistics

Adversary agent, constrained to diverge, actively constructs alternatives; and execution-based verification retains only interpretation pairs that produce different data—constituting a constructive proof of ambiguity.

We evaluate AIR on AMBROSIA (Saparina and Lapata, 2025a) and AmbiQT (Bhaskar et al., 2023). On AMBROSIA, under schema-only context, the realistic production setting, AIR outperforms IntentRL (Saparina and Lapata, 2026), the RL-trained state-of-the-art, across all metrics (e.g., F1 0.679 vs. 0.618). AIR degrades only one-third as much as IntentRL when moving from full database to schema-only context. With full database available, AIR achieves competitive recall (0.818 vs. 0.854) while generating fewer outputs (1.76 vs. 5.00 per query). On AmbiQT, AIR achieves higher F1 without any benchmark-specific adaptation.

Our main contributions are:

1. We develop a multi-agent framework that uncovers ambiguity through adversarial probing, verifies it through execution-based divergence, and resolves it through contrastive clarification, requiring no task-specific training.

2. We show that AIR outperforms the RL-trained state-of-the-art on all metrics under schema-only context, the realistic production setting, and more crucially, with only one-third the degradation.

3. We demonstrate transferability to new ambiguity types without explicit benchmark-specific adaptation.

2 Related Work

Ambiguity in Text-to-SQL. Recent benchmarks have formalized the ambiguity problem: AmbiQT (Bhaskar et al., 2023) pairs ambiguous questions with multiple gold SQL queries arising from schema-level ambiguities, and AMBROSIA (Saparina and Lapata, 2025a) covers scope, attachment, and vagueness ambiguities with execution-based evaluation. To address ambiguity, Saparina and Lapata (2025b) separates disambiguation from SQL generation by training an infilling model to recover missed interpretations, and Saparina and Lapata (2026) uses RL to generate multiple interpretations in a single forward pass. A common limitation is their *passive* treatment: candidates are generated but no method actively constructs a competing interpretation and verifies, through execution, that it produces different data. AIR shares the principle of separating disambiguation from SQL generation (Saparina and Lapata, 2025b), but replaces trained components with adversarial probing—using negative constraints to force exploration of the interpretation space.

Clarification Question Generation. Several systems address ambiguity by generating clarifying questions. Ding et al. (2026) designed a framework to identify and resolve ambiguities in Text-to-SQL tasks by engaging users in a multi-turn dialogue to clarify their intent before generating the final database query. Zhang and Choi (2025) explored interactive clarification with LLMs in broader semantic parsing tasks. These approaches generate clarifications based on perceived ambiguity without first proving that different interpretations lead to different data outcomes. AIR generates clarifications only after execution-based verification confirms genuine divergence, reducing unnecessary user interruptions.

Multi-Agent and Adversarial LLM Frameworks. Multi-agent architectures have shown promise for improving reasoning quality. Du et al. (2024) demonstrated that multi-agent debate improves factuality, and in Text-to-SQL, Wang et al. (2025) proposed MAC-SQL, where specialized agents handle decomposition, generation, and refinement. These frameworks share a *convergent* objective: agents iteratively improve a single answer. Liang et al. (2024) instead uses structured disagreement to encourage divergent thinking. However, in AIR, the Adversary agent seeks to falsify the Anchor's uniqueness, forcing exploration of the interpretation space rather than convergence toward a single answer. Adversary must diverge from one concrete anchor interpretation rather than disagreeing in general, and execution-based verification proves the two interpretations are genuinely distinct rather than relying on disagreement alone.

3 The Adversarial Intent Resolution Framework

The Adversarial Intent Resolution (AIR) framework treats Text-to-SQL ambiguity as a *probing and verification* problem. Without any task-specific training, AIR actively constructs competing interpretations through adversarial probing, verifies

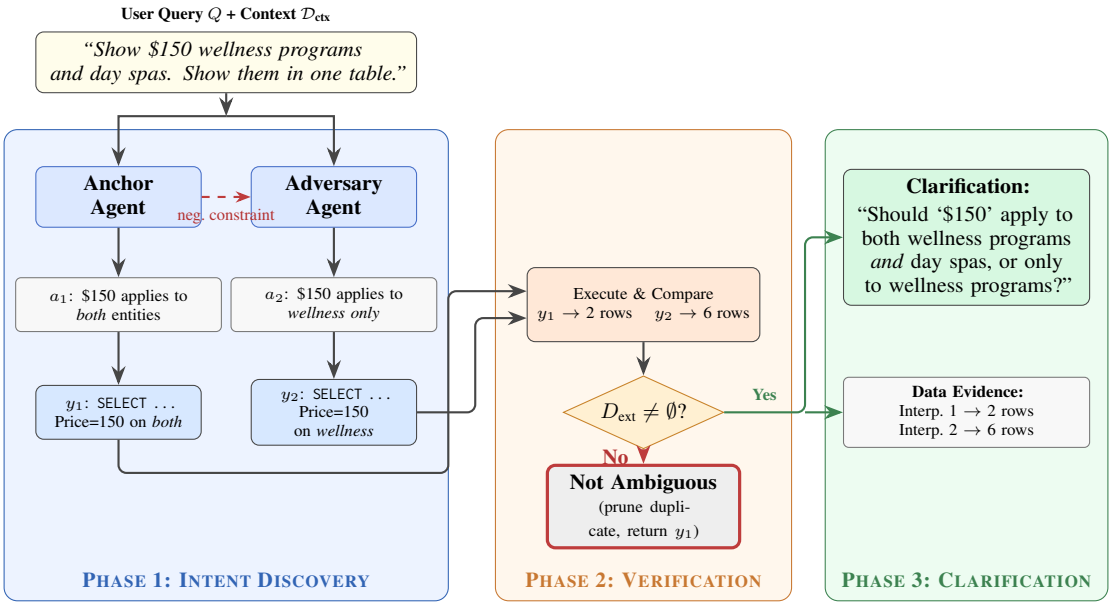

Figure 1: Overview of the Adversarial Intent Resolution (AIR) framework. Given an ambiguous query, the Anchor agent proposes a primary interpretation while the Adversary agent—constrained to diverge—discovers an alternative. Both are compiled to SQL and executed against the database. If the result sets differ ($D_{\text{ext}} \neq \emptyset$), a contrastive clarification grounded in actual data differences is presented to the user for further clarification; otherwise the duplicate interpretation is pruned.

through SQL execution that they produce different results, and generates targeted clarification questions grounded in the observed data differences.

3.1 Problem Formulation

Given a natural-language query Q, a database $\mathcal{D}$, and a context $\mathcal{D}_{\text{ctx}}$ provided to the LLM (which may be the full database or only the schema), a standard Text-to-SQL system produces a single SQL query $\hat{y}$ maximizing $P(y \mid Q, \mathcal{D}_{\text{ctx}})$. We characterize the set of valid interpretations $\mathcal{I}(Q, \mathcal{D}_{\text{ctx}})$ as a collection of pairs (a_i, y_i), where each a_i is a natural-language interpretation and y_i is its corresponding SQL translation.

Intent discovery and SQL generation operate on $\mathcal{D}_{\text{ctx}}$ (which may be schema-only in production), while execution-based verification runs against the full database $\mathcal{D}$.

AIR targets extensional ambiguity: a query is ambiguous when at least two valid interpretations produce different observable results on $\mathcal{D}$, i.e., $\exists i \neq j$ with $\text{Exec}(y_i, \mathcal{D}) \neq \text{Exec}(y_j, \mathcal{D})$. Two interpretations that differ in wording but happen to return the same result on the current data are not flagged, since the ambiguity would not change the answer.

3.2 Phase 1: Adversarial Intent Discovery

AIR employs a multi-agent architecture where agents reason exclusively at the intent level, producing natural-language interpretations before generating executable SQL.

Anchor Agent. The Anchor agent produces the primary interpretation:

$$(a_1, c_1) = \textsc{Anchor}(Q, \mathcal{D}_{\text{ctx}}) \tag{1}$$

where a_1 is a natural-language interpretation stating which ambiguous elements were resolved and how, and $c_1 \in [0, 1]$ is a confidence score indicating this is the only reasonable interpretation.

Adversary Agent. The Adversary receives the Anchor's interpretation a_1 as a negative constraint and attempts to find an alternative:

$$(a_2, c_2, f) = \textsc{Adversary}(Q, \mathcal{D}_{\text{ctx}}, a_1) \tag{2}$$

where $f \in \{\texttt{true}, \texttt{false}\}$ indicates whether a valid alternative was found, with a confidence score of $c_2 \in [0, 1]$.

Confidence-Based Gating. AIR applies confidence-based gating at two stages: (1) if $c_1 \geq \tau_{\text{anchor}}$ (default $\tau_{\text{anchor}} = 0.95$), the Adversary is suppressed entirely; (2) if the Adversary finds an alternative but with $c_2 < \tau_{\text{adv}}$ (default $\tau_{\text{adv}} = 0.5$),

the alternative is discarded as insufficiently plausible.

Multi-Round Adversary. For queries with more than two valid interpretations, the Adversary is called iteratively: after discovering a_2, it receives all previously discovered interpretations $\{a_1, a_2, \ldots\}$ as negative constraints and searches for yet another distinct reading. This continues until no new alternative is found or a maximum of K interpretations is reached (default $K = 5$).

3.3 Phase 2: Extensional Divergence Verification

Different interpretations may yield identical query results, so language-level differences alone do not constitute genuine ambiguity. AIR verifies divergence by compiling each interpretation to SQL and executing it.

SQL Generation. For each interpretation a_i, a SQL generation agent translates it into executable SQL:

$$y_i = \text{SQLGEN}(Q, \mathcal{D}_{\text{ctx}}, a_i) \quad (3)$$

This step operates with an execution feedback loop: if the generated SQL produces a database error, it receives the error message and retries (up to 3 attempts).

Execution-Based Pruning. For each pair of interpretations, we compute the symmetric difference of their result sets:

$$D_{\text{ext}}(y_i, y_j) = \text{Exec}(y_i, \mathcal{D}) \,\Delta\, \text{Exec}(y_j, \mathcal{D}) \quad (4)$$

If $D_{\text{ext}} = \emptyset$, the two interpretations are equivalent on the current data and one is pruned. Only interpretations that produce results distinct from all others are retained as genuinely ambiguous.

3.4 Phase 3: Contrastive Clarification Synthesis

When genuine ambiguity is confirmed, a Synthesizer agent generates a clarification grounded in the actual data differences:

$$r = \text{SYNTH}(Q, \{(a_i, \text{Exec}(y_i, \mathcal{D}))\}_{i=1}^{n}) \quad (5)$$

where n is the number of retained interpretations after pruning. The Synthesizer receives each interpretation paired with its execution results, enabling it to produce *contrastive* clarifications that highlight how results differ under each interpretation, rather than asking generic clarification questions.

3.5 Few-Shot Prompting

The intent discovery and SQL generation agents share a common set of five few-shot examples drawn from AMBROSIA's few-shot example split (Saparina and Lapata, 2025a), covering the three ambiguity types (scope, attachment, vagueness) plus a non-ambiguous control. These examples help teach the agents the structural patterns that create each ambiguity type and their corresponding SQL translations.

3.6 Illustrative Example

Figure 2 shows AIR applied to an attachment ambiguity query from AMBROSIA.

4 Experimental Setup

4.1 Datasets

AMBROSIA. AMBROSIA (Saparina and Lapata, 2025a) is a Text-to-SQL benchmark where natural-language questions exhibit linguistic ambiguity (scope, attachment, vagueness), leading to multiple valid SQL translations with different execution results. Each query is paired with gold interpretations and a SQLite database enabling execution-based evaluation. We use the data split from Saparina and Lapata (2026): 131 ambiguous and 309 unambiguous test examples (440 total).

AmbiQT. To assess cross-benchmark generalization, we evaluate on AmbiQT (Bhaskar et al., 2023). On this benchmark, ambiguity originates from schema design rather than linguistic structure: real-world Spider (Yu et al., 2018) databases are augmented with schema-level redundancies (synonym columns, synonym tables, normalized splits, pre-computed aggregates) so that a single question admits multiple structurally valid SQL translations. Following Saparina and Lapata (2026), we use the preprocessed validation split.

4.2 Context Conditions

We evaluate under two context settings for $\mathcal{D}_{\text{ctx}}$:

- **Full database:** $\mathcal{D}_{\text{ctx}} = \mathcal{D}$, the system receives the complete database (schema plus all data) as the context
- **Schema only:** $\mathcal{D}_{\text{ctx}}$ contains only the schema of the database, with no data.

The schema-only setting simulates production scenarios where exposing actual data to the LLM is infeasible due to privacy, scale, or latency constraints.

Query: *"Show $150 wellness programs and day spas. Show them in one table."*

Phase 1: Adversarial Intent Discovery
Anchor Agent: "$150" modifies the conjunction "wellness programs and day spas" → filter *both* tables by Price = 150.
Adversary Agent (receives Anchor as negative constraint): "$150" attaches only to "wellness programs" → filter wellness programs by price, but include *all* day spas.

Phase 2: Extensional Divergence Verification

SQL 1 (Anchor):
```
SELECT Name FROM Wellness_Retreat
 WHERE Price = 150
UNION
SELECT Name FROM Day_Spa
 WHERE Price = 150
```

SQL 2 (Adversary):
```
SELECT Name FROM Wellness_Retreat
 WHERE Price = 150
UNION
SELECT Name FROM Day_Spa
```

Execution results differ: SQL 1 returns {Mountain Retreat, Oasis Day Spa} (2 rows); SQL 2 returns {Mountain Retreat, Oasis Day Spa, Escape Day Spa, Paradise Day Spa, Serenity Day Spa, Zen Day Spa} (6 rows). $D_{\text{ext}} \neq \emptyset$ → ambiguity confirmed, both retained.

Phase 3: Contrastive Clarification Synthesis
"I found two possible interpretations:
1. Both wellness programs AND day spas that cost $150 → Mountain Retreat, Oasis Day Spa (2 results)
2. Wellness programs at $150, plus all day spas regardless of price → 6 results
Which did you mean?"

Figure 2: AIR correctly identifies an attachment ambiguity: does "$150" modify both conjuncts or only "wellness programs"? The adversarial probe finds the alternative interpretation, execution confirms divergent outputs (2 vs. 6 rows), and the synthesizer presents a contrastive clarification grounded in actual data differences.

We emphasize that "schema-only" describes what is passed to the LLM as context, not whether the system can query the database: SQL execution (Phase 2) and the Synthesizer (Phase 3) still run against the full database $\mathcal{D}$ as a backend operation under standard database access controls, identical to any ordinary application query. The privacy, cost, and latency concern is specific to sending database content to a third-party LLM as context.

4.3 Implementation Details

We evaluate AIR with three models representing diverse architectures: Claude Sonnet 4.5 (Anthropic, 2025), Gemma 4 26B A4B (Google, 2026), and gpt-oss-20b (OpenAI et al., 2025). All configurations use temperature 0 and their default reasoning capabilities. No task-specific training or fine-tuning is performed for any configuration.

4.4 Baselines

Prompt Baseline. We establish a prompting baseline by directly instructing the LLM to generate multiple SQL interpretations for each query. We evaluate this with Claude Sonnet 4.5.

IntentRL. Our primary comparison is IntentRL (Saparina and Lapata, 2026), the current state-of-the-art on AMBROSIA. IntentRL is an RL model fine-tuned on AMBROSIA's training split, using a composite reward for ambiguity detection and multi-interpretation SQL generation. It generates 5 SQL queries per question in a single forward pass. We select IntentRL over DisambigParse (Saparina and Lapata, 2025b), a two-stage method that first predicts interpretations then generates SQL. On ambiguous queries, IntentRL outperforms DisambigParse across all metrics (e.g., recall 82.4% vs. 59.7%, precision 77.5% vs. 26.3%). It also surpasses prompting with a 235B model (46.2% recall) and supervised fine-tuning (68.5% recall). This makes it the most stringent test of whether inference-time adversarial reasoning can match training-based approaches.

4.5 Evaluation Metrics

Following Saparina and Lapata (2025a), we use execution-based evaluation metrics. Since ambiguous queries map to multiple correct SQL translations, we report recall, precision, and F1. We also report One-Found (at least one gold execution output covered by predictions) and All-Found (all gold execution outputs covered by predictions).

5 Results

5.1 AMBROSIA Results

5.1.1 Performance with Schema-Only as Context

In production deployments, exposing full database contents to an LLM is often infeasible due to privacy, latency, and cost constraints. A training-free

Context	Method	Recall	Precision	F1	One-Found	All-Found
Full database	Prompt Baseline (Sonnet 4.5)	0.697	0.592	0.610	0.797	0.590
	SOTA (IntentRL)	**0.854**	**0.758**	**0.786**	**0.886**	**0.823**
	AIR – Claude Sonnet 4.5 (ours)	0.818	0.707	0.729	0.880	0.758
	AIR – Gemma 4 26B A4B (ours)	0.804	0.584	0.640	0.876	0.729
	AIR – gpt-oss-20b (ours)	0.759	0.728	0.718	0.845	0.673
Schema only	Prompt Baseline (Sonnet 4.5)	0.610	0.539	0.548	0.698	0.520
	SOTA (IntentRL)	0.691 (−19%)	0.589 (−22%)	0.618 (−21%)	0.723 (−18%)	0.664 (−19%)
	AIR – Claude Sonnet 4.5 (ours)	**0.757** (−7%)	**0.658** (−7%)	**0.679** (−7%)	**0.813** (−8%)	**0.701** (−8%)
	AIR – Gemma 4 26B A4B (ours)	0.750 (−7%)	0.518 (−11%)	0.578 (−10%)	0.820 (−6%)	0.682 (−6%)
	AIR – gpt-oss-20b (ours)	0.648 (−15%)	0.642 (−12%)	0.626 (−13%)	0.728 (−14%)	0.568 (−16%)

Table 1: Performance on AMBROSIA test split. Percentages indicate degradation from full database to schema-only context. IntentRL leads under full database context, but under schema-only, the realistic production setting, AIR (Claude Sonnet 4.5) outperforms IntentRL across all metrics with roughly one-third the degradation (e.g., F1: ∼7% vs. ∼20%). AIR with smaller open-source models also exhibits less degradation than IntentRL, confirming that the robustness stems from the framework architecture.

approach that operates effectively with schema-only access is therefore highly desirable. Table 1 presents performance under this realistic setting. AIR (Claude Sonnet 4.5) outperforms IntentRL across all metrics: recall (0.757 vs. 0.691), precision (0.658 vs. 0.589), F1 (0.679 vs. 0.618), One-Found (0.813 vs. 0.723), and All-Found (0.701 vs. 0.664). Taking F1 as an example, this represents a 10% relative improvement, demonstrating that AIR's adversarial probing architecture produces effective disambiguation even without access to the actual data.

Even with smaller open-source models, AIR outperforms IntentRL for the schema-only setting (F1: gpt-oss-20b 0.626 vs. IntentRL's 0.618), validating that AIR's advantage stems from the framework rather than just the backbone model.

Two further comparisons in Table 1 isolate the framework's contribution from backbone model capacity. First, the Prompt Baseline uses the same Claude Sonnet 4.5 backbone as AIR but without the adversarial structure, scoring schema-only F1 0.548 versus AIR's 0.679, a 13.1-point gap attributable to the adversarial probing architecture rather than model capacity. Second, gpt-oss-20b is a mixture-of-experts model with ∼3.6B active parameters per forward pass, comparable to or smaller than IntentRL's backbone in active compute, yet AIR with gpt-oss-20b still exceeds IntentRL's schema-only F1 (0.626 vs. 0.618). Together, these comparisons indicate that AIR's advantage comes from the adversarial framework itself rather than backbone scale.

Metric	AIR	IntentRL
Avg. output / query	1.76	5.00
Avg. unique outputs / query	1.76	2.99
Extra outputs (> gold)	145 (33.0%)	336 (76.4%)

Table 2: Comparison of SQL execution outputs between AIR and IntentRL on AMBROSIA with the full database as context.

5.1.2 Performance with Full Database as Context

Table 1 also shows performance when the full database is available as context. IntentRL leads with higher metrics (recall 0.854, F1 0.786) than AIR with Claude Sonnet 4.5 (recall 0.818, F1 0.729). However, this comes at the cost of generating 5 predictions per query regardless of ambiguity, of which only 2.99 are unique execution outputs—meaning ∼40% of IntentRL's generations are duplicates. AIR's adversarial probing and execution-based pruning ensure all outputs are unique, generating only 1.76 per query on average. More predicted candidates mean more chances to match a gold output, but this also comes at a direct cost to user experience. In a production Text-to-SQL system, each additional interpretation requires a follow-up clarification from the user. IntentRL produces extra outputs on 336 of 440 queries (76.4%), while AIR does so on only 145 (33.0%). IntentRL would require clarification on more than double the queries. Table 2 summarizes the generation efficiency.

Context	AIR (Claude Sonnet 4.5)	IntentRL
Schema only	204 (46.4%)	102 (23.2%)
Full database	216 (49.1%)	89 (20.2%)

Table 3: Perfect match accuracy on AMBROSIA. A query is correct only if all gold outputs are perfectly matched and no extra outputs are produced. Under schema-only context, AIR retains 94.4% of the best full database performance without requiring actual data for intent discovery or SQL generation.

5.1.3 Robustness: Schema-Only vs. Full Database

When moving from full database to schema-only context, AIR (Claude Sonnet 4.5) degrades by only ∼7% across recall, precision, and F1, while IntentRL degrades by ∼19–22% on the same metrics. This ∼3× gap reveals that IntentRL has learned to rely on actual data in the database to generate correct SQL. AIR's adversarial architecture, which reasons at the intent level before committing to SQL, maintains its effectiveness regardless of whether actual data in the database is available or not. Combined with its training-free design, this makes AIR practical for privacy-sensitive and latency-constrained production environments where schema-only access is the only option.

5.1.4 Perfect Match Accuracy

The recall, precision, and F1 metrics above use partial credit: e.g., a prediction covering 3 of 4 gold cell values counts as a partial match. This can obscure whether a system produces usable outputs in practice. In a deployed data conversation system, a user should only be asked for clarification when there is genuine ambiguity, and the system's interpretation set should perfect match the true set of valid interpretations, no missing readings, no extra ones. We therefore calculate perfect match accuracy, a metric with two requirements: (1) every gold execution output is exactly reproduced by some prediction (using the same cell-multiset comparison as All-Found), and (2) the number of unique predicted outputs equals the number of gold outputs, penalizing over-generation. A query is correct only if both conditions hold.

Table 3 shows results under both context settings. Under schema-only context, AIR achieves 204 perfect matches, maintaining 94.4% of the best full database performance (216) without requiring access to actual data in the database to discover intents and generate SQL queries. This demonstrates that AIR's adversarial probing can identify the correct interpretation set from schema information alone. In contrast, IntentRL's fixed 5-candidate strategy produces more unique outputs than gold interpretations on 76% of queries, failing condition (2) even when all gold outputs are covered (273 of 362 All-Found queries fail due to extra outputs). AIR's targeted generation tends to trigger clarification only when genuinely necessary. Examples of AIR failure cases are provided in Appendix B.

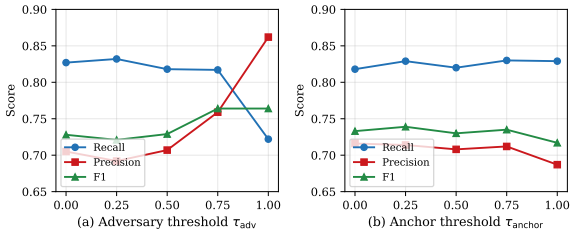

Figure 3: Effect of confidence thresholds on AMBROSIA with the full database as context. (a) The adversary threshold τ_{adv} controls a clear recall–precision tradeoff: higher values filter more aggressively, improving precision at the cost of recall. (b) The anchor threshold τ_{anchor} has minimal effect, with all metrics varying within ∼3 percentage points.

5.1.5 Threshold Sensitivity Analysis

Figure 3 shows how AIR's two confidence thresholds affect the recall–precision tradeoff on AMBROSIA with the full database as context.

The adversary confidence threshold τ_{adv} (Figure 3(a)) is the primary tuning parameter. At $\tau_{adv} = 0$ (accept all adversary alternatives), recall peaks at 0.827 but precision is lowest (0.705). As the threshold increases, low-confidence alternatives are filtered: at $\tau_{adv} = 0.75$, precision rises to 0.759 while recall remains competitive (0.817), yielding the highest F1 (0.764). It should be noted that, at $\tau_{adv} = 1.0$, all adversary alternatives are rejected, reducing AIR to a single-interpretation system: precision reaches 0.862 but recall drops to 0.722. The 10.5% recall gap between $\tau_{adv} = 0$ and $\tau_{adv} = 1$ confirms that adversarial probing contributes meaningful coverage, while the adversary's self-reported confidence provides a useful signal for filtering implausible alternatives.

This tradeoff has an asymmetric cost in deployment: missing a genuine ambiguity silently returns a possibly wrong answer, while a false alarm only costs one extra clarification round. This asymmetry favors operating points that trade some precision

for recall, i.e., lower values of τ_{adv} than the default. All main results in this paper use $\tau_{\text{adv}} = 0.5$ as a balanced default; deployments where missed ambiguity is the more costly failure mode should lower τ_{adv}.

The anchor confidence threshold τ_{anchor} (Figure 3b) has minimal impact: all metrics vary within $\sim$3 percentage points across the full range. This indicates that the adversary threshold alone suffices to control output quality. All main results in this paper use $\tau_{\text{adv}} = 0.5$ and $\tau_{\text{anchor}} = 0.95$.

5.2 Cross-Benchmark Transferability

Table 4 presents results on AmbiQT, where ambiguity arises from schema design (synonym columns/tables, split tables, pre-aggregated views) rather than linguistic structure. This benchmark tests whether AIR's adversarial probing generalizes beyond the linguistic ambiguity types.

AIR achieves higher F1 under both context settings (0.686 vs. 0.594 full database; 0.675 vs. 0.570 schema-only), driven by substantially higher precision (0.758 vs. 0.590). IntentRL achieves higher recall (0.695 vs. 0.650) and All-Found (0.525 vs. 0.439) through generating more predictions per query. Notably, AIR under schema-only (F1 0.675) achieves near-identical performance to its full database setting (F1 0.686). This confirms that schema-level ambiguities are inherently context-independent and that AIR's adversarial probing does not rely on actual data.

6 Discussion

Why schema-only matters. In production deployments, exposing full database contents to an LLM raises privacy, latency, and cost concerns. This concern is specifically about what is sent to the LLM as context; "schema-only" governs intent discovery and SQL generation, not SQL execution, which still runs agains full database under standard database access control. AIR's adversarial probing operates at the intent level, reasoning about what a query *could mean* rather than what the data contains, which explains why removing data has low impact ($\sim$7% degradation). IntentRL's larger degradation ($\sim$20%) appears to stem from data-dependent shortcuts acquired during RL fine-tuning. On AmbiQT, where IntentRL was not trained, both methods degrade $<$4%.

Why adversarial probing outperforms both prompting and RL. Prompting and RL represent two ends of a spectrum, but both treat interpretation discovery as a single-pass problem. Prompting for "multiple interpretations" produces surface-level variants because the model's probability mass concentrates around its most likely reading (prompt baseline recall: 0.697 vs. AIR's 0.818). RL rewards diversity but without verifying that candidates are genuinely distinct, leading to over-generation. AIR solves both: the negative constraint forces the Adversary to explore *away* from the Anchor's reading, and execution-based pruning retains only candidates with provably different outputs.

Targeted generation vs. fixed-budget coverage. IntentRL generates 5 candidates per query regardless of ambiguity, producing extra outputs on 76% of queries with each risking an unnecessary user clarification. AIR generates alternatives only when a plausible competing interpretation is found (1.76 outputs on average). The adversary threshold τ_{adv} provides a tuning knob: lowering it increases coverage at the cost of more clarifications; raising it reduces interruptions at the cost of missing some ambiguities.

Execution as proof of ambiguity. Ambiguity in AIR is not merely detected but *proven*: two interpretations are genuinely ambiguous only when their SQL translations produce different results. Under schema-only context, execution still provides a divergence signal through structural differences (different joins, column selections, or filtering logic), explaining why AIR's precision remains high (0.658 vs. 0.589 for IntentRL) despite the absence of data content.

Evaluation granularity. The execution-based metrics rely on execution output matching between predicted and gold outputs. This penalizes predictions that are reasonable from a user's perspective but differ in surface form. For instance, AIR may include an identifying column alongside the requested data (e.g., returning `labPosition`, `salaryRange` when only `salaryRange` is gold), or interpret "accommodate 500 people" as $\geq$500 rather than exactly 500. Both are plausible choices that receive no credit under strict output matching. These cases indicate that execution output matching does not distinguish between SQL generation errors and legitimate interpretation differences, potentially understating practical correctness. An LLM-as-judge evaluation that assesses equivalence at multiple levels including intent coverage, SQL

Context	Method	Recall	Precision	F1	One-Found	All-Found
Full database	IntentRL	**0.695**	0.590	0.594	**0.866**	**0.525**
	AIR – Claude Sonnet 4.5 (ours)	0.650	**0.758**	**0.686**	0.862	0.439
Schema only	IntentRL	**0.684**	0.549	0.570	**0.842**	**0.527**
	AIR – Claude Sonnet 4.5 (ours)	0.642	**0.743**	**0.675**	0.847	0.436

Table 4: Performance on AmbiQT validation set to test cross-benchmark generalization. AIR under schema-only achieves near-identical performance to its full database setting. AIR achieves higher F1 through substantially better precision, while IntentRL's exhaustive generation yields higher recall and All-Found.

structural correctness, and output semantics could provide a more faithful measure of system quality for ambiguity-aware Text-to-SQL.

7 Conclusion

We presented AIR, a multi-agent framework for ambiguity-aware Text-to-SQL that requires no task-specific training. AIR detects ambiguity through adversarial probing, verifies it through execution-based divergence, and resolves it through contrastive clarification.

Under schema-only context, the realistic production setting where privacy, cost, and latency constraints prevent data exposure, AIR outperforms the RL-trained IntentRL across all metrics (e.g., F1 0.679 vs. 0.618), with roughly one-third the performance degradation when full database is unavailable ($\sim$7% vs. $\sim$20%). On perfect match accuracy, AIR under schema-only retains 94.4% of the best full-database performance without requiring actual data for intent discovery or SQL generation. On AmbiQT, AIR achieves higher F1 without any benchmark-specific adaptation, demonstrating cross-benchmark transferability.

AIR's targeted generation (1.76 outputs per query) tends to trigger clarification only when a genuine competing interpretation is found, in contrast to IntentRL's fixed 5-candidate strategy that produces extra outputs on 76% of queries. These results show that structured adversarial reasoning at inference time offers a practical alternative to training-based approaches for ambiguity-aware Text-to-SQL, with superior robustness, precision, and deployability.

Limitations

AIR requires multiple LLM calls per query (Anchor, Adversary, SQL generation for each interpretation, and Synthesizer), resulting in $\sim$ 40 seconds end-to-end latency per query using unoptimized serial calls to the Bedrock Claude Sonnet 4.5 API. The approximate per-stage median breakdown is: Anchor $\sim$9s, Adversary $\sim$22s, SQL generation $\sim$6s, Synthesizer $\sim$5s. The Adversary dominates due to its multi-round reasoning constrained by prior interpretations; even the Anchor alone takes $\sim$9s, likely reflecting API network latency rather than model computation. Optimizations such as parallelization of agent calls and caching would reduce this for real-time production deployment.

Both benchmarks we evaluate on are single-turn; measuring cumulative user burden and task completion across multi-turn clarification dialogues, and extending evaluation to larger real-world Text-to-SQL benchmarks such as BIRD once multi-interpretation annotations are available, are directions for future work.

While we evaluate whether the correct interpretation set is produced, we do not evaluate the quality of the generated clarification questions from a user perspective. A human evaluation or LLM-as-judge assessment of their naturalness, informativeness, and usefulness to end users is left to future work.

Ethical Considerations

This work does not create any new datasets. All experiments use existing public benchmarks (AMBROSIA and AmbiQT) that have been verified to contain no sensitive or harmful information.

References

Anthropic. 2025. System card: Claude sonnet 4.5.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving Text-to-SQL generation under ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7053–7074, Singapore. Association for Computational Linguistics.

Zhongjun Ding, Yin Lin, Tianjing Zeng, Rong Zhu, Bolin Ding, and Jingren Zhou. 2026. Ambisql: Interactive ambiguity detection and resolution for Text-to-SQL. *Preprint*, arXiv:2508.15276.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL empowered by large language models: A benchmark evaluation. *Preprint*, arXiv:2308.15363.

Google. 2026. Gemma 4 model card. `https://ai.google.dev/gemma/docs/core/model_card_4`. Accessed 2026-05-24.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *Preprint*, arXiv:2310.03714.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise Text-to-SQL workflows. In *International Conference on Learning Representations*, volume 2025, pages 28691–28735.

Haoyang Li, Jing Hui, Jian Qu, and 1 others. 2024a. CodeS: Towards building open-source language models for Text-to-SQL. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024b. Can LLM already serve as a database interface? a big bench for large-scale database grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems*.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Mengyi Liu, Xieyang Wang, Jianqiu Xu, Weijia Yi, and Ouri Wolfson. 2026. A systematic review of natural language interfaces for databases. *Frontiers of Computer Science*, 20(11):2011623.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Mohammadreza Pourreza and Davood Rafiei. 2024. DIN-SQL: Decomposed in-context learning of Text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*.

Alane Saparina and Mirella Lapata. 2025a. AMBROSIA: A benchmark for parsing ambiguous questions into database queries. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*.

Irina Saparina and Mirella Lapata. 2025b. Disambiguate first, parse later: Generating interpretations for ambiguous questions. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.

Irina Saparina and Mirella Lapata. 2026. Reasoning about intent for ambiguous requests. *Preprint*, arXiv:2511.10453.

Garima Singh, Prachi Chhabra, and Tathagat Banerjee. 2026. A comprehensive review of llm-based Text-to-SQL systems: Methods, datasets, and trends. *Computational Systems and Artificial Intelligence*, 2(1):1–6.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Zhang, Zhao Yan, and Zhoujun Liu. 2025. MAC-SQL: A multi-agent collaborative framework for Text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 540–557.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Gaurav Nuti, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2026. Arctic-Text2SQL-R1: Simple rewards, strong reasoning in Text-to-SQL. *Preprint*, arXiv:2505.20315.

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*. Association for Computational Linguistics.

Michael JQ Zhang and Eunsol Choi. 2025. Clarify when necessary: Resolving ambiguity through interaction with LMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5541–5558, Albuquerque, New Mexico. Association for Computational Linguistics.

A Agent Prompts

This appendix presents the core prompts (implemented as DSPy (Khattab et al., 2023)) for AIR's three main agents. All agents share a common set of few-shot examples covering scope, attachment, and vagueness ambiguity types (see Section 3.5).

A.1 Anchor Agent Prompt

System: Identify the single most likely interpretation of the user's query against the database schema. Analyze the question step-by-step, map terms to schema elements, and commit to one clear interpretation. State your mapping choices — do NOT discuss alternative interpretations or what you didn't choose.

Inputs:

- `query`: The user's natural language query
- `database_dump`: Database schema and sample data

Outputs:

- `interpretation`: Step-by-step analysis: parse the question, map each key term to table.column in the schema, identify any ambiguity, state your interpretation.
- `confidence`: Confidence 0.0–1.0 that this is the only reasonable interpretation.

A.2 Adversary Agent Prompt

System: Determine if the query has a genuinely different interpretation from the anchor's. A valid alternative interpretation must be completely supported by the natural language query without hallucination.

Inputs:

- `query`: The user's natural language query
- `database_dump`: Database schema and sample data
- `anchor_interpretation`: The anchor's interpretation of the query

Outputs:

- `found_alternative`: true if a valid but genuinely different interpretation exists, false otherwise.
- `alternative_interpretation`: Step-by-step analysis for the alternative interpretation. Write 'NONE' if no alternative exists.
- `alternative_confidence`: Confidence 0.0–1.0 that the alternative is valid. 0.0 if no alternative.

A.3 SQL Generation Agent Prompt

System: Generate a SQL query from the interpretation. Translate the interpretation into valid SQLite SQL. Output ONLY raw SQL, no markdown fences. SELECT only the columns the question asks for.

Inputs:

- `query`: The user's natural language query
- `database_dump`: Database schema and sample data
- `interpretation`: Step-by-step interpretation mapping question terms to schema elements

Outputs:

- `sql_query`: Raw SQL query implementing the interpretation.

B AIR Error Examples

The following examples illustrate representative failure modes where AIR generates incorrect outputs.

Extra interpretations. For the query "Provide a list of all horror films and also list thriller films with a budget of 20 million. Show them in one table," the ground truth applies the budget filter only to thrillers:

`SELECT title WHERE genre = 'Horror' OR (genre = 'Thriller' AND budget = 20M)` $\rightarrow$ 3 rows

AIR's Adversary generates an additional interpretation where the budget applies to both genres:

`SELECT title WHERE (genre = 'Horror' OR genre = 'Thriller') AND budget = 20M` $\rightarrow$ 2 rows

Extra columns. For "What is the salary range for female laboratory technicians and research scientists? Show them in one table," the ground truth selects only `salaryRange`, but AIR selects `labPosition, salaryRange`, adding an identifying column to disambiguate the salary ranges.

Extra rows. For "List all lobbies or terraces that can accomodate 500 people. Show them in one table," the ground truth applies the size constraint to both types:

`SELECT Name WHERE type IN ('Lobby','Terrace') AND Size = 500` $\rightarrow$ 2 rows

AIR interprets "accommodate 500" as $\geq$500 rather than exactly 500:

`SELECT Name WHERE type = 'Lobby' AND Size >= 500 OR type = 'Terrace'` $\rightarrow$ 4 rows

Missing interpretations. For "What experience do I need for the open position with the greatest salary?" the schema has both `Minimum_Years` and `Preferred_Years` columns. The ground truth expects three interpretations (each column separately, plus both). AIR generates only:

`SELECT Minimum_Years FROM Jobs ORDER BY Salary DESC LIMIT 1`

The Adversary does not recognize that "experience" is vague across the two columns and fails to propose the alternative mappings.

License

The AMBROSIA dataset is used under its original license as provided by Saparina and Lapata (2025a). The AmbiQT dataset is used under its original license as provided by Bhaskar et al. (2023). All open source models (Gemma 4 26B A4B, gpt-oss-20b, IntentRL) are used under their original licenses.

Computational Budget

AIR is training-free and requires only inference-time computation. The total experimental budget spans 3 backbone models, 2 context settings on AMBROSIA (440 samples), a prompt baseline, and 9 threshold sweep configurations. Additionally, 2 context settings on AmbiQT (1,911 samples).

Claude Sonnet 4.5 was accessed via Amazon Bedrock API. Open source models, gpt-oss-20b and IntentRL, were run internally on A100 GPU nodes. Total cumulative inference time was approximately 200 GPU-hours across all experiments.