

000 001 002 003 004 005 SYNSQL: SYNTHETIC DATABASE GENERATION FOR 006 ROBUST EVALUATION OF TEXT-TO-SQL SYSTEMS 007 008 009

010 **Anonymous authors**
011

012 Paper under double-blind review
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027

028 ABSTRACT 029

030 A central challenge in test-time scaling for text-to-SQL is generating test
031 databases that can reliably validate arbitrary queries, yet existing tools remain
032 narrow in scope and limited in capability. We introduce SynSQL, a framework for
033 synthesizing test databases conditioned on natural language questions and schema
034 structure. Unlike prior approaches that generate data from gold queries, SynSQL
035 leverages large language models to generate tables directly from question-schema
036 alignment, while remaining compatible with gold queries when available for eval-
037 uation. The framework consists of a schema selector, a synthesizer, and a critic
038 with iterative refinement, which jointly align semantic cues from the question with
039 structural constraints from the schema to guide database synthesis. Experiments
040 on the Spider and BIRD benchmarks demonstrate that SynSQL produces realis-
041 tic, constraint-respecting databases that effectively stress-test text-to-SQL models.
042 SynSQL not only complements the coverage of human-curated benchmarks but
043 also outperforms prior test database generation methods across diverse schema
044 complexities. On Spider, SynSQL achieves a 93.04% success rate, surpassing
045 the original human-authored dataset (92.55%), and on BIRD it attains a 79.23%
046 agreement rate, substantially higher than prior automated methods, all while op-
047 erating without access to gold queries during data generation.
048

049 1 INTRODUCTION 050

051 Verifying program correctness, ensuring that a system behaves as intended, has long been a cen-
052 tral challenge in computer science, complicated by the undecidability of the halting problem (Tur-
053 ing, 1936). Foundational work in formal methods sought principled ways to reason about cor-
054 rectness (Hoare, 1969), but scaling such approaches to modern machine learning systems remains
055 elusive. In practice, benchmarks have emerged as practical surrogates for correctness: carefully
056 curated collections of test data that enables systematic evaluation and comparison across systems.
057 From early datasets such as MNIST (LeCun et al., 2002) to large-scale language understanding
058 benchmarks like SQuAD (Rajpurkar et al., 2016), GLUE (Wang et al., 2018), and HELM (Liang
059 et al., 2022), benchmarks have repeatedly provided the common ground on which progress in AI
060 research is measured.
061

062 In text-to-SQL, benchmarks such as Spider (Yu et al., 2018) and BIRD (Li et al., 2023) have
063 played this role with significant impact. By pairing natural language (NL) questions with SQL
064 queries over curated relational databases, these benchmarks have enabled rapid iteration and com-
065 parison of text-to-SQL systems. However, despite their central role, recent analyses have shown
066 that benchmark-driven evaluation remains fragile, narrow in scope, and often misaligned with real
067 deployment needs (Mitsopoulou & Koutrika, 2025; Renggli et al., 2025).
068

069 A core challenge lies in the quality of databases themselves. Since benchmarks rely on fixed, static
070 databases, any inconsistencies or artifacts directly undermine evaluation reliability. Issues such as
071 referential integrity violations, unexpected NULL values, case mismatches between questions and
072 database content, or noisy entries can lead to misleading outcomes: false positives (incorrect queries
073 that nevertheless return the expected result) or false negatives (semantically equivalent queries that
074 yield different outputs). Consequently, the same query may be judged differently depending on the
075 database (Zhong et al., 2020).
076

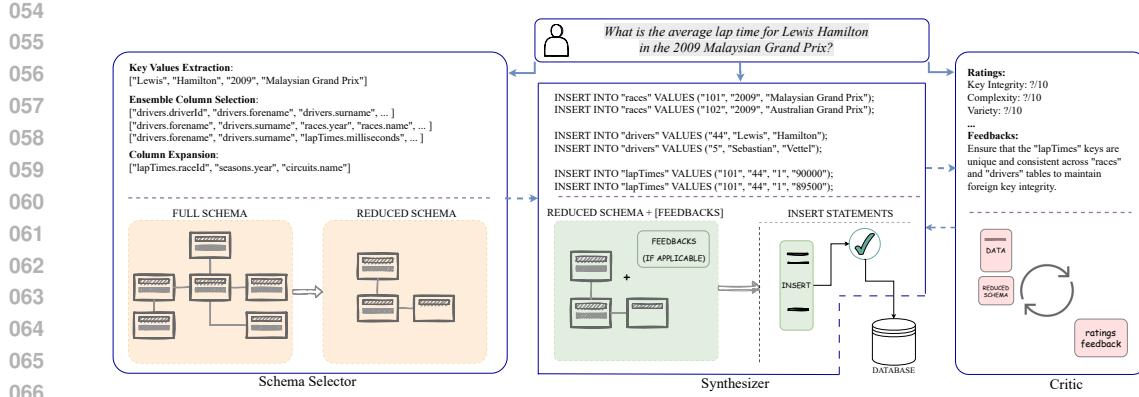


Figure 1: Overview of the SynSQL framework. The schema selector identifies relevant schema elements and reduces the schema space. The synthesizer generates test data based on the NL question and reduced schema. The critic evaluates the quality of the generated data and provides feedback for improvement.

Beyond benchmark construction, evaluation often takes place in settings where suitable databases do not exist. During system development, teams face cold-start scenarios in which schemas exist but no database content is available, or privacy-sensitive applications where real data cannot be shared (Ge et al., 2024). In such cases, developers are left with fragile proxy metrics or ad hoc synthetic data, neither of which provides robust evaluation. Addressing these challenges requires frameworks capable of synthesizing realistic, constraint-respecting test databases tailored for text-to-SQL evaluation.

Limitations of Prior Work Existing approaches for test database generation, such as AGENDA (Deng et al., 2005) and Qex (Veanes et al., 2010), as well as more recent systems like XData (Chandra et al., 2015; Somwase et al., 2024) and TestSuiteAccuracy (Zhong et al., 2020), focus on generating input tables to test SQL queries. These methods assume access to a gold SQL query q , presumed correct, and generate neighboring or mutated queries that are syntactically similar but semantically incorrect. Test data is then designed to differentiate q from its mutations. While effective in query-centric testing, this paradigm is bounded by the complexity of queries the framework can handle and does not generalize to settings when evaluation must be driven directly by natural language questions and schema structure.

Our Approach In contrast to prior work, we explore a more realistic and challenging setting: test database generation directly from the NL question and schema. This removes the dependency of the data synthesis process on curated SQL annotations and their mutations, enabling broader applicability. Our central research question is: *Can a meaningful test database be constructed using only the NL question and schema, in order to assess the correctness of the generated SQL query?*

Our hypothesis is that semantic signals from the question, when combined with schema structure, can guide the synthesis of input tables that reveal errors, ambiguities, or misinterpretations in SQL generation. To this end, we introduce SynSQL, a framework for automatic databases synthesis that leverages large language models (LLMs) for question understanding and systematically aligns question semantics with schema constraints. SynSQL adopts a modular, three-agent design (Figure 1): the *Schema Selector* prunes irrelevant relations to simplify schemas; the *Synthesizer* generates test tables conditioned on the NL question and reduced schema; and the *Critic* evaluates the table data and provides feedback for iterative refinement. This interactive pipeline produces realistic, minimal, and constraint-respecting databases that adapt to both schema and query semantics.

Our contributions can be summarized as follows: (1) We introduce SynSQL, a framework for synthetic database generation in text-to-SQL tasks to extend the existing evaluation methods. (2) Our framework ensures generation of targeted, minimal, and realistic databases that respect schema constraints (e.g. foreign key relationships and uniqueness) and align with question semantics to differentiate between correct and incorrect SQL queries. (3) Through comprehensive experiments

108 on the Spider and BIRD benchmarks, we demonstrate that SynSQL complements the coverage
 109 of human-curated benchmarks and also outperforms previous automated test database generation
 110 methods across varying schema complexities. Specifically, on Spider, SynSQL achieves a 93.04%
 111 success rate, surpassing the original human-authored databases (92.55%), and on BIRD, it reaches
 112 a 79.23% agreement rate, outperforming prior automated approaches.
 113

114 2 RELATED WORK

115
 116
 117
 118 **Test Data Generation for SQL** Test data generation for SQL queries has been explored through
 119 various strategies, including random data generation (Bati et al., 2007), constraint-based ap-
 120 proaches (Veanes et al., 2010; Shah et al., 2011), and mutation-based techniques (Tuya et al.,
 121 2007). Qex (Veanes et al., 2010) synthesizes tuples using SMT solving to guarantee non-empty
 122 query results. XData (Shah et al., 2011) builds on this idea by generating datasets that differentiate
 123 a given SQL query from its syntactically similar but semantically incorrect mutations, with recent
 124 follow-up work (Somwase et al., 2024) supporting more complex SQL constructs. TestSuiteAccu-
 125 racy (Zhong et al., 2020), introduced in the context of the Spider benchmark (Yu et al., 2018),
 126 employs fuzzing techniques (Padhye et al., 2019) to create large numbers of random databases, from
 127 which a small subset is selected to maximize code coverage with respect to the gold query. In
 128 contrast, APEL (Zhong et al., 2022) synthesizes small human-interpretable input datasets to enable
 129 non-programmers to reliably label program outputs.
 130

131 **Synthetic Data Generation with LLMs** LLMs have been increasingly used for generating
 132 synthetic data across NLP tasks (Wang et al., 2022; Ramesh et al., 2022; Nad  s et al., 2025; Yoo
 133 et al., 2021; Dai et al., 2025), including program test synthesis (Wang et al., 2024; Chen et al.,
 134 2022; Yuan et al., 2023; Bhatia et al., 2024; Yang et al., 2024). In the domain of text-to-SQL,
 135 recent work has focused on producing large synthetic NL-SQL corpora for supervised learning,
 136 such as Gretel (Meyer et al., 2024) and Omni-SQL (Li et al., 2025). Complementary work also
 137 explores question generation over structured sources using LLM reasoning strategies (Liang et al.,
 138 2023), though these methods target question synthesis rather than database construction. Beyond
 139 data generation, LLMs have also been used in database-related tasks such as SQL dialect translation
 140 and query adaptation (Ngom & Kraska, 2024; Daviran et al., 2025). These efforts demonstrate the
 141 versatility of LLMs in understanding schema structure and SQL semantics but do not address the
 142 challenge of creating executable, constraint-respecting test databases.
 143

144 **Relation to Our Work** Unlike prior work that focuses on generating NL-SQL pairs or mutating
 145 SQL queries, our framework targets the synthesis of complete test databases directly from ques-
 146 tion-schema alignment. SynSQL’s three-stage design—schema selection, LLM-guided data syn-
 147 thesis, and iterative criticism—enforces schema integrity and semantic alignment, enabling realistic
 148 and compact databases tailored for evaluating text-to-SQL systems.
 149

150 3 METHODOLOGY

151 3.1 PROBLEM FORMULATION

152 Given a natural language question Q , a database schema S , and optional auxiliary knowledge K ,
 153 the task of synthetic data generation using a language model π can be formulated as:
 154

$$155 D_{syn} = f(Q, S, K; \pi) \quad (1)$$

156 where D_{syn} denotes the generated synthetic database and f is a generation function parameterized
 157 or guided by the language model π . When D_{syn} is used to evaluate the correctness of a predicted
 158 SQL query q_{pred} against a reference query q_{gold} , it must be populated with realistic rows that reflect
 159 the semantic cues of Q ; for example, if Q asks for *What was Brent Thomason’s major?*, the database
 160 should contain name of majors for student named Brent Thomason. In addition, D_{syn} should en-
 161 able clear differentiation between correct and incorrect queries by producing different results for

162 semantically distinct queries, while also respecting all schema constraints such as foreign keys and
 163 domain-specific rules.

165 **3.2 OVERVIEW OF FRAMEWORK**

167 We introduce SynSQL, a three-staged framework for generating high-quality synthetic databases tai-
 168 lored for evaluating text-to-SQL systems. The pipeline consists of a Schema Selector, a Synthesizer,
 169 and a Critic. As shown in Figure 1, SynSQL incorporates a feedback loop between the synthesizer
 170 and the critic. After an initial round of data generation, the critic evaluates the output and provides
 171 structured feedback. If the data achieves a score above a predefined threshold (e.g., 8 out of 10), the
 172 process terminates. Otherwise, the synthesizer incorporates the feedback into a new round of data
 173 generation. This loop typically continues for a fixed number of rounds or until the generated data
 174 satisfies all quality criteria.

175 **3.3 SCHEMA SELECTION**

178 Real-world database schemas often contain many tables and columns (sometimes in the range of
 179 hundreds), while most NL questions reference only a small subset. Providing the full schema to
 180 the model can lead to the generation of unnecessary or inconsistent data (e.g., mismatched column
 181 counts or malformed tables). The challenge is to accurately identify the relevant schema elements
 182 while preserving enough context for realistic and coherent data synthesis.

183 Our schema selection strategy differs from that of state-of-the-art text-to-SQL systems (Talaei et al.,
 184 2024; Pourreza et al., 2024; Gao et al., 2024), which typically map question keywords to schema
 185 elements (e.g., table and column names) and database values in order to prune the schema down
 186 to only those components strictly necessary for query generation. In our setting, however, no real
 187 database content is available, and the data itself must be generated synthetically. Consequently,
 188 SynSQL prioritizes recall over minimality: rather than restricting to the smallest matching subset,
 189 we retain all schema elements that could plausibly support realistic data synthesis, including those
 190 not explicitly mentioned in the question, such as foreign keys or related attributes.

191 **Algorithm 1** Schema Selector for SynSQL

192 **Require:** Natural language question Q , full schema S , auxiliary knowledge K , model π

193 1: **Ensure:** Reduced schema $S_{reduced}$ relevant to Q
 194 2: Initialize $S_{core} \leftarrow \emptyset$
 195 3: **for** temperature t in $\{0, 0.3, 0.7\}$ **do**
 196 4: Query π with (Q, S, K) at temperature t to extract core elements
 197 5: $S_{core} \leftarrow S_{core} \cup$ elements returned by π
 198 6: **end for**
 199 7: Query π for semantically related columns to S_{core}
 200 8: $S_{aux} \leftarrow$ related columns returned by π
 201 9: $S_{reduced} \leftarrow S_{core} \cup S_{aux}$
 202 10: **return** $S_{reduced}$

204
 205 To identify this relevant subset, we employ an ensemble expansion strategy. As outlined in Algo-
 206 rithm 1, the LLM is queried multiple times at different temperature settings to capture a diverse set
 207 of core columns. The union of the results is taken to maximize recall. Next, we expand the set by
 208 querying the LLM for semantically similar or functionally related columns. This helps in discover-
 209 ing auxiliary schema elements that support better data realism. At the end of this process, we have a
 210 reduced schema that includes only the relevant elements needed to cover the question. This reduced
 211 schema is then used by the data synthesizer to generate test data.

212 The benefit of this approach is that we can generate smaller, more focused databases that are easier
 213 to inspect and validate, while still capturing the necessary complexity to effectively evaluate SQL
 214 queries. These compact synthetic databases especially enable human-in-the-loop evaluation in cold-
 215 start development or privacy-sensitive environments, where real data is unavailable or restricted.
 Though not the focus of this work, SQL query correctness can then be determined by comparing

216 the results against expected outputs, observed by the human evaluator, even in the absence of gold
 217 queries or large-scale benchmarks.
 218

219 **3.4 DATA SYNTHESIS**
 220

221 Existing approaches in database synthesis focus on techniques that generate data to differentiate be-
 222 tween reference SQL query and its neighbors/mutations. This process is bounded by the complexity
 223 of queries their framework can handle. If the system cannot yet process a certain query structure,
 224 then it also cannot generate databases that can kill mutations for that structure. For example, in
 225 XData (Somwase et al., 2024), a feature like `ORDER BY` falls outside its scope. Or in TestSuiteAc-
 226 curacy (Zhong et al., 2020), the authors note that too many `WHERE` operations can lead to ineffective
 227 distinguishing of neighboring queries.

228 In contrast, our synthesizer does not rely on reference queries and focuses on generating realistic
 229 databases that align with the semantics of the NL question. We leverage LLMs to generate SQL
 230 `INSERT` statements that can populate the reduced schema with synthetic yet realistic data. The gen-
 231 eration process is guided by the intent of the NL question and aims to produce data that both aligns
 232 with the semantics and provides meaningful contrast between correct and incorrect SQL queries.

233 **Data Validation** Database schemas typically encode structural constraints, including primary and
 234 foreign keys, uniqueness, and value ranges. Generating synthetic data that satisfies these constraints,
 235 while preserving realistic value distributions, is challenging. Failure to respect these constraints can
 236 result in data that causes SQL queries to fail or produce misleading results. Once the initial data is
 237 generated, we apply rule-based postprocessing to ensure correctness and schema compliance. This
 238 involves removing tables or columns present in data but not in the schema, only retaining data that
 239 respects schema constraints, enforcing case sensitivity based on named entities and literals extracted
 240 from the question, and ensuring that the number of values matches the number of columns in each
 241 table. When discrepancies are found, we either pad with `NULLs` or truncate values as needed. This
 242 step is critical for maintaining the syntactic and semantic integrity of the SQL statements.
 243

244 **3.5 DATA CRITIC**
 245

246 The synthetic data produced by the synthesizer may not always meet the desired quality standards.
 247 There could be issues such as misalignment with the NL question, violations of schema constraints,
 248 or lack of edge cases in the data distribution. Inspired by the idea of self-correction for LLMs (Pan
 249 et al., 2023), we introduce a Critic module that evaluates the generated data and provides feedback
 250 for improvement which determines whether the data should be accepted or refined. It scores the
 251 data on a scale of 1 to 10 across several dimensions, including alignment with question hints, key
 252 and referential integrity, schema coverage, complexity of the data, variety in records, and overall
 253 relevance to the question. Instead of using simple pass/fail rules, the critic provides a detailed
 254 evaluation that allows the synthesizer to improve in targeted ways. If the average score meets the
 255 quality threshold, the data is accepted. Otherwise, the critic’s feedback is incorporated into a new
 256 iteration of data generation. This loop ensures that the final dataset not only conforms to the schema
 257 and executes without errors but also meaningfully tests the correctness of SQL predictions in the
 258 context of the original question.

259 **4 EXPERIMENTS**
 260

261 Our experimental evaluation is designed to assess two core aspects: (1) the quality of the syn-
 262 thetic data generated by SynSQL compared to human-curated datasets and competitive automated
 263 approaches, and (2) the contribution of different components of SynSQL via an ablation study.
 264

265 **4.1 EXPERIMENTAL SETUP**
 266

267 **Datasets** We evaluate SynSQL on two widely used text-to-SQL benchmarks: Spider (Yu et al.,
 268 2018) and BIRD (Li et al., 2023). Spider features natural language questions paired with SQL
 269 queries over relatively simple schemas, while BIRD includes more complex queries involving mul-
 270 tiple joins, nested subqueries, and advanced aggregations. This contrast enables a comprehensive

270 assessment of SynSQL across varying schema complexity and linguistic difficulty. Our auxiliary
 271 knowledge K (as defined in Section 3.1) is set to the evidence or hints associated with each question
 272 for BIRD, and to an empty string for Spider, which does not provide such hints.
 273

274 **Baselines** We compare SynSQL to three baselines: (1) **BIRD Original** and (2) **Spider Original**,
 275 which refer to the original human-authored databases included in the dev splits of the respec-
 276 tive benchmarks and serve as the gold standard for test data; and (3) **TestSuiteAccuracy (TSA)**
 277 by Zhong et al. (2020) that generates test databases by finding neighboring queries to each gold
 278 query and applying fuzzing-based techniques to produce random databases, benefiting from access
 279 to the gold query during data generation. Among existing automated test data generation methods,
 280 TSA is the only approach that has an available codebase and can be applied to any standalone dataset
 281 such as Spider or BIRD, making it the only directly comparable baseline for our setting.
 282

283 **Text-to-SQL Systems** To evaluate the ability of SynSQL generated databases to separate correct
 284 queries from incorrect ones, we use two competitive text-to-SQL models to generate candidate SQL
 285 queries: **DIN-SQL** (Pourreza & Rafiei, 2023) and **DAIL-SQL** (Gao et al., 2023). The predictions
 286 produced by these models are evaluated against the gold queries using both human-curated and
 287 synthetic databases.
 288

289 4.2 EVALUATION METRICS

290 We employ three complementary metrics to assess the quality and utility of the generated databases:
 291

292 **Success Rate (SR)** Using this metric we measure the fraction of questions for which the gold SQL
 293 query produces a non-empty result on the test database. This indicates whether the generated data
 294 captures the semantic intent of the natural language question (from the perspective of the human
 295 who wrote the gold query). Random data often fails here, so aligning with question intent is crucial.
 296

297 **Execution Accuracy (EX)** We can evaluate the fraction of questions for which the model-
 298 generated SQL query produces the same result as the gold SQL query when executed on the test
 299 database (Zhong et al., 2017). The comparison of this metric with human-curated databases, mea-
 300 sures the database’s ability to distinguish between correct and incorrect SQL queries.
 301

302 **Agreement Rate (AR)** Inspired by Cohen’s Kappa (Cohen, 1960) score, this metric assesses
 303 the level of agreement between the discriminative power of model-generated and human-curated
 304 databases on a query-by-query basis. Using this metric we can show that if the evaluation of each
 305 query on the synthetic database agrees with the evaluation on the original database, then the synthetic
 306 database is in agreement with the original one. Formally, AR is defined as:
 307

$$308 \text{AR} = \frac{P_o - P_e}{1 - P_e} \quad (2)$$

309 where P_o is the observed agreement (i.e., the proportion of queries where both databases yield
 310 the same evaluation result) and P_e is the expected agreement by chance, calculated based on the
 311 marginal probabilities of each database’s evaluations. For more on this metric please refer to Ap-
 312 pendix section A.2.
 313

314 5 RESULTS

315 5.1 MAIN RESULTS

316 We present the performance of the proposed SynSQL framework using a mix of proprietary and
 317 open-source language models: GPT-4.1-mini, Gemini-2.5-Flash, and Qwen-3-8B. The critic module
 318 uses a maximum of three refinement iterations, terminating early if the data achieves a quality score
 319 of 8.0 or higher on a 10-point scale. For our vanilla baseline, we prompt the LLM to generate
 320 synthetic data in a single pass without schema reduction or critic feedback. The vanilla baseline
 321 is equivalent to SynSQL with only the synthesizer component, using the same prompting strategy
 322 but operating on the full schema without iterative refinement. For more details on implementation,
 323 please refer to section 6.

324
 325 Table 1: Performance comparison of SynSQL and baseline methods on BIRD and Spider dev sets.
 326 SR: success rate (%), AR: agreement rate (%), EX: execution accuracy (%), and performance gap
 327 (Δ) is defined as $|EX_{\text{orig}} - EX_{\text{method}}|$.

328 329	Dataset	Method	SR	DAIL-SQL			DIN-SQL		
				AR↑	EX	$\Delta \downarrow$	AR↑	EX	$\Delta \downarrow$
330 331 332 333 334 335	BIRD	BIRD (Original)	99.87	100.00	52.93	-	100.00	41.39	-
		Vanilla GPT-4.1-Mini	69.43	58.57	68.12	15.19	61.88	55.61	14.22
		Vanilla Gemini-2.5-Flash	67.14	61.56	66.95	14.02	65.64	54.95	13.56
		SynSQL(Qwen-3-8B)	73.60	70.64	58.21	5.28	72.94	47.78	6.39
		SynSQL(Gemini-2.5-Flash)	80.57	75.46	55.61	2.68	79.23	44.00	2.61
		SynSQL(GPT-4.1-Mini)	82.07	73.51	55.02	2.09	78.90	43.09	1.70
336 337 338 339 340	Spider	Spider (Original)	92.55	100.00	80.66	-	100.00	80.46	-
		Vanilla GPT-4.1-Mini	91.88	78.20	83.07	2.41	76.44	82.85	2.39
		Vanilla Gemini-2.5-Flash	82.59	74.33	84.24	3.58	72.46	83.08	2.62
		SynSQL(Qwen-3-8B)	77.18	66.99	84.24	3.58	66.89	81.72	1.26
		SynSQL(Gemini-2.5-Flash)	92.84	80.69	81.72	1.06	76.53	81.33	0.87
		SynSQL(GPT-4.1-Mini)	93.04	78.05	81.24	0.58	76.01	79.79	0.67

341
 342 Table 2: Performance comparison of SynSQL and TestSuiteAccuracy (TSA) alongside the original
 343 benchmarks. BIRD results are computed on a subset of 922 examples where TSA evaluation was
 344 feasible.

346	Dataset	Method	SR	DAIL-SQL			DIN-SQL		
				AR↑	EX	$\Delta \downarrow$	AR↑	EX	$\Delta \downarrow$
348 349 350	BIRD*	BIRD (Original)	99.57	100.00	51.74	-	100.00	38.72	-
		TSA Zhong et al. (2020)	77.76	60.97	57.27	5.53	64.71	46.31	7.59
		SynSQL(Gemini-2.5-Flash)	81.67	74.76	54.56	2.82	76.14	41.97	3.25
		SynSQL(GPT-4.1-Mini)	82.86	70.85	53.90	2.16	76.45	40.67	1.95
352 353 354	Spider	Spider (Original)	92.55	100.00	80.66	-	100.00	80.46	-
		TSA Zhong et al. (2020)	99.03	83.15	76.31	4.35	79.99	75.73	4.73
		SynSQL(Gemini-2.5-Flash)	92.84	80.69	81.72	1.06	76.53	81.33	0.87
		SynSQL(GPT-4.1-Mini)	93.04	78.05	81.24	0.58	76.01	79.79	0.67

355
 356 **Success Rate.** As shown in Table 1, SynSQL with GPT-4.1-Mini as the base model, achieves a
 357 success rate of 82.07% on BIRD, outperforming both the vanilla GPT-4.1-Mini baseline (69.43%)
 358 and other LLM configurations. On Spider, SynSQL again leads with 93.04%, even surpassing the
 359 original human-authored database (92.55%). This is partly due to inconsistencies in the Spider
 360 original databases, such as missing data or formatting issues (e.g., trailing spaces), which SynSQL
 361 avoids by design. Appendix figures 18 and 19 provide examples of such issues.

363
 364 **Agreement Rate.** SynSQL achieves high agreement rates with the original databases, indicating
 365 that it effectively preserves and complements the evaluation characteristics of human-curated data.
 366 On BIRD, SynSQL(Gemini-2.5-Flash) attains AR scores of 75.46% for DAIL-SQL and 79.23%
 367 for DIN-SQL, significantly outperforming the vanilla GPT-4.1-Mini baseline (61.56% and 65.64%,
 368 respectively). On Spider, SynSQL again achieves substantial agreement scores, closely matching
 369 the original database’s performance.

370
 371 **Execution Accuracy and Gap.** On BIRD, the EX gap between SynSQL (GPT-4.1-Mini) and the
 372 original data is just 2.09 for DAIL-SQL and 1.70 for DIN-SQL, an order of magnitude smaller
 373 than the gaps observed for vanilla GPT-4.1-Mini (15.19 and 14.22, respectively). On Spider, all
 374 methods perform more closely due to its simpler schema and queries. SynSQL again performs
 375 competitively with EX gaps under 1.0. Consequently, SynSQL closely preserves the ranking of
 376 text-to-SQL models observed with the original databases. For instance, DAIL-SQL consistently
 377 outperforms DIN-SQL across all SynSQL configurations, closely mirroring the execution gaps seen
 378 with the original data. This consistency in model ranking further validates the effectiveness of
 379 SynSQL-generated databases for complementary robust evaluation.

378 **Comparison with TestSuiteAccuracy (TSA).** As shown in Table 2, while TSA performs well
 379 on Spider, achieving the highest success rate at 99.03%, its effectiveness drops on BIRD, with a
 380 success rate of 77.76%, significantly lower AR of 60.97% for DAIL-SQL, and larger EX gaps (e.g.,
 381 5.53 for DAIL-SQL). In contrast, SynSQL achieves higher success (82.86%), higher agreement rate
 382 (74.76%), and lower EX gaps (2.16), indicating better semantic alignment and discriminative power
 383 on complex schemas.

384 TSA’s reliance on gold SQL queries offers an advantage on simpler benchmarks but becomes a
 385 liability on datasets like BIRD. For example, it fails to resolve foreign key references in databases
 386 like `european_football_2`, and generates impractically large test databases (e.g., multiple GBs
 387 per question) in `card_games` and `codebase_community`. Consequently, we excluded such
 388 problematic cases and report results on a filtered subset of 922 BIRD questions. For fairness, the
 389 same subset was used to evaluate SynSQL and the BIRD original databases in Table 2. Overall,
 390 SynSQL not only outperforms TSA on BIRD but also delivers stable performance across datasets,
 391 generating minimal and realistic test databases. For more on realism and minimalistic nature of these
 392 databases, see A.9. These results demonstrate SynSQL’s practicality and robustness for realistic
 393 text-to-SQL evaluation scenarios.

394
 395 **Performance Varying Schema Complexity** We also analyze SynSQL’s performance rates across
 396 question-schema complexity levels on the BIRD dev set, defined by the total number of columns
 397 across all tables referenced in each gold query: Low (1–15), Medium (16–60), and High (61+).
 398 As shown in Appendix Figure 3, SynSQL consistently outperforms vanilla GPT-4.1-mini, with the
 399 largest margin at high complexity, 76.99% vs. 36.15% for success rate and 71.07% vs. 33.19% for
 400 agreement rate. This demonstrates SynSQL’s robustness as schema complexity increases.

402 5.2 FINE-GRAINED ANALYSIS OF EXECUTION ACCURACY

404 To better understand the impact of synthetically generated data on execution accuracy, we conducted
 405 an error analysis comparing the outputs of DAIL-SQL queries executed on the original BIRD dev
 406 set versus the SynSQL-generated data. We randomly sampled 250 questions from the BIRD dev
 407 set and executed the corresponding DAIL-SQL queries on both the original BIRD and the SynSQL-
 408 generated databases. Comparing the outputs to the ground truth, we found 35 instances of disagree-
 409 ments where the evaluation results differed across the two datasets. Of these, SynSQL produced
 410 correct (positive) results in 21 cases where BIRD did not, resulting in a higher execution accuracy.
 411 Conversely, 14 cases were evaluated as positive on the original BIRD dataset but negative on Syn-
 412 SQL.

413 A closer examination revealed that 15 out of the 35 dis-
 414 agreements are attributed to data quality issues in the orig-
 415 inal BIRD dev set. For instance, in the `toxicology`
 416 database, the `molecule_id` column in the `bond` table
 417 is a foreign key referencing the `molecule` table. How-
 418 ever, 101 rows in the `bond` table contain `molecule_id`
 419 values not present in the `molecule` table, causing other-
 420 wise correct queries to fail (e.g., question 286). Similarly,
 421 in the `thrombosis_prediction` database, the `ID` col-
 422 umn in the `Examination` table is a foreign key referenc-
 423 ing the `Patient` table, yet 694 rows in `Examination`
 424 have `ID` values absent from `Patient`, leading to possible
 425 false negatives (e.g., question 1273). These are examples of
 426 one of the key advantages of SynSQL: by generating data
 427 that respects all schema constraints, it can avoid false negatives caused by such data quality issues
 in human-curated datasets.

428 Another example is question 357: *What type of promotion is of card ‘Duress’?* The gold query
 429 includes a `NOT NULL` condition for `promoTypes`, while the predicted query does not. On the
 430 original BIRD database, this leads to false negative due to the presence of many `NULL` artifacts.
 431 However, since the question and evidence do not specify the presence of `NULL` values, the predicted
 query is arguably correct. SynSQL-generated database has a question-oriented design which gener-

Table 3: Analysis of evaluation disagreements between SynSQL and BIRD from a sample of 250 questions with 35 discrepancies.

BIRD → SynSQL	Change
Negative → Positive	21
From False Negative to True Positive	10
From True Negative to False Positive	11
Positive → Negative	14
From False Positive to True Negative	5
From True Positive to False Negative	9

432 ates data that aligns with the semantic signals of the question, resulting in a more accurate evaluation
 433 of such cases.

434 Table 3 summarizes the transitions in execution results on a sample of 250 questions. Notably,
 435 SynSQL corrects 10 false negatives and 5 false positives. These findings suggest that SynSQL’s
 436 constraint enforcement and question-driven synthesis approach helps mitigate common data quality
 437 issues found in human-curated datasets, such as referential integrity violations and unexpected NULL
 438 values, resulting in more accurate and reliable evaluation of text-to-SQL systems on cases affected
 439 by such inconsistencies.

440

441

442 5.3 ABLATION STUDIES

443

444 Table 4 shows the effect of schema selection on performance. The SynSQL method with ensemble-
 445 expansion outperforms all ablated versions, confirming that both phases contribute meaningfully to
 446 success rate, especially on BIRD, where complex schemas increase the difficulty of accurate column
 447 selection. Using the oracle schema yields highest success rate, indicating further improvements
 448 in schema selection could enhance performance. We also observe that the average column count
 449 selected by the schema selector is significantly lower than the full schema, demonstrating SynSQL’s
 450 ability to generate compact databases while maintaining high success rates. **However, aggressive**
 451 **reduction risks omitting columns required by gold queries, causing otherwise correct queries to fail.**
 452 **This highlights the inherent tension between minimizing schema complexity and preserving query**
 453 **executability.** Despite this limitation, SynSQL’s ensemble-expansion strategy achieves a balance
 454 that maintains high success rates while generating significantly more compact databases than the
 455 **full schema.**

456

457 As illustrated in Appendix figures 16 and 17, these compact synthetic databases are easier to inspect
 458 and validate, facilitating future directions such as human-in-the-loop evaluation and generating ex-
 459 pected outputs via table reasoning.

460

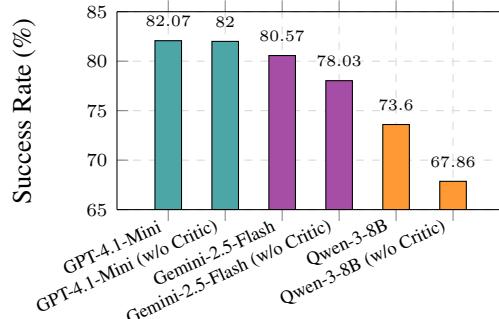
461

462 Table 4: Ablation study on schema selection strategies in SynSQL. We report success rate (SR) and
 463 average column count (CC) selected by the schema selector on the BIRD and Spider dev sets. All
 464 variants use GPT-4.1-Mini as both the base model and the critic.

465 Method	466 BIRD		467 Spider	
	468 SR (%)	469 CC	470 SR (%)	471 CC
SynSQL w Oracle Schema	91.46	4.71	94.58	2.85
SynSQL	82.07	8.37	93.04	6.71
SynSQL w/o Expansion	79.53	5.42	92.75	3.92
SynSQL w/o Ensemble-Expansion	77.38	4.99	91.88	3.56
SynSQL w/o Schema Selection (Full-Schema)	71.25	75.56	92.94	24.55

472

473 As illustrated in Figure 2, the impact
 474 of the critic component on the success
 475 rate of SynSQL across three LLMs on
 476 the BIRD dev set is significant. Incorpor-
 477 ating the critic consistently improves
 478 performance: Qwen-3-8B improves from
 479 67.86% to 73.60%. The effect is less
 480 pronounced for GPT-4.1-Mini, which al-
 481 ready performs strongly, but the critic still
 482 ensures more stable and reliable results.
 483 These results demonstrate that the critic
 484 plays a key role in improving the qual-
 485 ity of synthetic data. By enforcing align-
 486 ment with the question intent, schema
 487 constraints, and data diversity, the critic
 488 enables the synthesizer to produce more
 489 accurate and executable SQL queries. This



487 Figure 2: Impact of the critic component on success
 488 rate (%) of SynSQL with three different LLMs on the
 489 BIRD dev set.

486 iterative feedback loop is particularly val-
 487 uable for smaller or weaker models, which are more prone to generating invalid or incomplete outputs.
 488

489 6 CONCLUSION

490 In this work, we presented SynSQL, a novel framework for synthetic database generation in text-to-
 491 SQL evaluation, producing databases that respect schema constraints and align with the natural lan-
 492 guage question intent. Extensive experiments on Spider and BIRD benchmarks show that SynSQL
 493 not only complements human-curated datasets but also outperforms existing automated methods,
 494 particularly as schema complexity increases. These results highlight SynSQL’s ability to enhance
 495 the effectiveness and reliability of text-to-SQL evaluation. The realistic and minimal nature of the
 496 synthetic data generated by SynSQL, combined with its ability to address key challenges in current
 497 evaluation practices, such as referential integrity violations or unexpected and noisy artifacts, paves
 498 the way for more robust and scalable evaluation of text-to-SQL systems encountered at real-world
 499 Text-to-SQL challenges.

501 502 REPRODUCIBILITY STATEMENT

503 The link to the anonymous codebase for SynSQL is available in the README.md file in supple-
 504 mentary materials. SynSQL is implemented in Python using the LangChain framework (Chase, 2022).
 505 For data generation, we employ a mix of proprietary and open-source language models: GPT-4.1-
 506 mini (OpenAI, 2023), Gemini-2.5-Flash, and Qwen-3-8B. Each experiment uses a single model as
 507 the base for the schema selector, synthesizer, and critic. Qwen-3-8B experiments were run on a
 508 server with NVIDIA A100 GPUs (40GB RAM), while GPT-4.1-mini and Gemini-2.5-Flash were
 509 accessed via their respective APIs. For the TestSuiteAccuracy (TSA) baseline, we use the official
 510 implementation provided by Zhong et al. (2020) and a server equipped with an AMD EPYC 7601
 511 32-Core Processor and 1TB RAM.

512 513 ETHICS STATEMENT

514 Both datasets used in our experiments, Spider and BIRD, are publicly available and widely used
 515 benchmarks in the text-to-SQL research community. We have ensured that our use of these datasets
 516 complies with their respective licenses and terms of use. The synthetic data generated by SynSQL
 517 is created solely for research purposes and does not contain any personally identifiable information
 518 or sensitive content. The language models employed in our framework, including GPT-4.1-mini,
 519 Gemini-2.5-Flash, and Qwen-3-8B, are accessed through their respective APIs or open-source im-
 520 plementations. We adhere to the usage policies and guidelines set forth by the providers of these
 521 models to ensure ethical use. We also acknowledge the potential bias inherent in large language
 522 models, which may inadvertently influence the synthetic data generation process. This falls within
 523 the broader challenges of bias in AI and LLMs. Additionally, in accordance with the ICLR 2026
 524 Code of Ethics, we acknowledge the use of large language models to assist with the polishing of the
 525 writing in this paper.

526 527 REFERENCES

528 Hardik Bati, Leo Giakoumakis, Steve Herbert, and Aleksandras Surna. A genetic approach for
 529 random testing of database systems. In *Proceedings of the 33rd international conference on Very
 530 large data bases*, pp. 1243–1251, 2007.

531 Shreya Bhatia, Tarushi Gandhi, Dhruv Kumar, and Pankaj Jalote. Unit test generation using gener-
 532 ative ai: A comparative performance analysis of autogeneration tools. In *Proceedings of the 1st
 533 International Workshop on Large Language Models for Code*, pp. 54–61, 2024.

534 Bikash Chandra, Bhupesh Chawda, Biplab Kar, KV Maheshwara Reddy, Shetal Shah, and S Sudar-
 535 shan. Data generation for testing and grading sql queries. *The VLDB Journal*, 24(6):731–755,
 536 2015.

537 Harrison Chase. Langchain: Building applications with llms through composability. <https://github.com/hwchase17/langchain>, 2022.

540 Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu
 541 Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.
 542

543 Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological mea-
 544 surement*, 20(1):37–46, 1960.

545 Haixing Dai, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao,
 546 Shaochen Xu, Fang Zeng, Wei Liu, et al. Auggpt: Leveraging chatgpt for text data augmentation.
 547 *IEEE Transactions on Big Data*, 2025.

548 Mohammadreza Daviran, Brian Lin, and Davood Rafiei. Sql-exchange: Transforming sql queries
 549 across domains. *arXiv preprint arXiv:2508.07087*, 2025.

550 Yuetang Deng, Phyllis Frankl, and David Chays. Testing database transactions with agenda. In
 551 *Proceedings of the 27th international conference on Software engineering*, pp. 78–87, 2005.

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

555 Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao
 556 Hong, Zhiling Luo, et al. Xiyuan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv
 557 preprint arXiv:2411.08599*, 2024.

558 Yunqing Ge, Jianbin Qin, Shuyuan Zheng, Yongrui Zhong, Bo Tang, Yu-Xuan Qiu, Rui Mao,
 559 Ye Yuan, Makoto Onizuka, and Chuan Xiao. Privacy-enhanced database synthesis for bench-
 560 mark publishing (technical report). *arXiv preprint arXiv:2405.01312*, 2024.

561 Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications
 562 of the ACM*, 12(10):576–580, 1969.

563 Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to
 564 document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

565 Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo,
 566 Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on
 567 real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*, 2024.

568 Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang,
 569 Tieying Zhang, Jianjun Chen, Rui Shi, et al. Omnisql: Synthesizing high-quality text-to-sql data
 570 at scale. *arXiv preprint arXiv:2503.02240*, 2025.

571 Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhu Li, Bowen Li, Bailin Wang, Bowen Qin,
 572 Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for
 573 large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*,
 574 36:42330–42357, 2023.

575 Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian
 576 Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language
 577 models. *arXiv preprint arXiv:2211.09110*, 2022.

578 Yuanyuan Liang, Jianing Wang, Hanlun Zhu, Lei Wang, Weining Qian, and Yunshi Lan. Prompting
 579 large language models with chain-of-thought for few-shot knowledge base question generation.
 580 *arXiv preprint arXiv:2310.08395*, 2023.

581 Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten
 582 Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. Synthetic-Text-To-
 583 SQL: A synthetic dataset for training language models to generate sql queries from natural lan-
 584 guage prompts, April 2024. URL [https://huggingface.co/datasets/gretelai/
 585 synthetic-text-to-sql](https://huggingface.co/datasets/gretelai/synthetic-text-to-sql).

586 Anna Mitsopoulou and Georgia Koutrika. Analysis of text-to-sql benchmarks: limitations, chal-
 587 lenges and opportunities. In *Proceedings 28th International Conference on Extending Database
 588 Technology, EDBT 2025*, pp. 199–212. OpenProceedings. org, 2025.

594 Mihai Nadăș, Laura Dioșan, and Andreea Tomescu. Synthetic data generation using large language
 595 models: Advances in text and code. *IEEE Access*, 2025.

596

597 Amadou Latyr Ngom and Tim Kraska. Mallet: Sql dialect translation with llm rule generation.
 598 In *Proceedings of the Seventh International Workshop on Exploiting Artificial Intelligence Tech-*
 599 *niques for Data Management*, pp. 1–5, 2024.

600

601 OpenAI. Gpt-4 technical report. <https://arxiv.org/abs/2303.08774>, 2023.

602

603 Rohan Padhye, Caroline Lemieux, Koushik Sen, Mike Papadakis, and Yves Le Traon. Semantic
 604 fuzzing with zest. In *Proceedings of the 28th ACM SIGSOFT International Symposium on*
605 Software Testing and Analysis, pp. 329–340, 2019.

606

607 Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang
 608 Wang. Automatically correcting large language models: Surveying the landscape of diverse self-
 609 correction strategies. *arXiv preprint arXiv:2308.03188*, 2023.

610

611 Mohammadreza Pourreza and Tom Kubik. How to get gemini to deeply under-
 612 stand your database. <https://cloud.google.com/blog/products/databases/how-to-get-gemini-to-deeply-understand-your-database>, Nov. 14 2025.
 613 Accessed: 2025-11-27.

614

615 Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-
 616 sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348,
 617 2023.

618

619 Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok
 620 Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. Chase-sql: Multi-path reasoning
 621 and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*,
 622 2024.

623

624 Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions
 625 for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

626

627 Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-
 628 conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

629

630 Cedric Renggli, Ihab F Ilyas, and Theodoros Rekatsinas. Fundamental challenges in evaluating
 631 text2sql solutions and detecting their limitations. *arXiv preprint arXiv:2501.18197*, 2025.

632

633 Shetal Shah, S Sudarshan, Suhas Kajbaje, Sandeep Patidar, Bhanu Pratap Gupta, and Devang Vira.
 634 Generating test data for killing sql mutants: A constraint-based approach. In *2011 IEEE 27th*
635 International Conference on Data Engineering, pp. 1175–1186. IEEE, 2011.

636

637 Lei Sheng and Shuai-Shuai Xu. Csc-sql: Corrective self-consistency in text-to-sql via reinforcement
 638 learning. *arXiv preprint arXiv:2505.13271*, 2025.

639

640 Sunanda Somwase, Parismita Das, and S Sudarshan. Data generation for testing complex queries.
 641 *arXiv preprint arXiv:2409.18821*, 2024.

642

643 Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi.
 644 Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

645

646 Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Pro-*
647 ceedings of the London Mathematical Society, 42(1):230–265, 1936.

648

649 Javier Tuya, Ma José Suárez-Cabal, and Claudio De La Riva. Mutating database queries. *Infor-*
650 mation and Software Technology, 49(4):398–417, 2007.

651

652 Margus Veanes, Nikolai Tillmann, and Jonathan De Halleux. Qex: Symbolic sql query explorer.
 653 In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pp.
 654 425–446. Springer, 2010.

648 Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman.
 649 Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv*
 650 *preprint arXiv:1804.07461*, 2018.
 651

652 Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. Software test-
 653 ing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software*
 654 *Engineering*, 50(4):911–936, 2024.
 655

656 Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and
 657 Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions.
 658 *arXiv preprint arXiv:2212.10560*, 2022.
 659

660 Lin Yang, Chen Yang, Shutao Gao, Weijing Wang, Bo Wang, Qihao Zhu, Xiao Chu, Jianyi Zhou,
 661 Guangtai Liang, Qianxiang Wang, et al. An empirical study of unit test generation with large
 662 language models. *arXiv e-prints*, pp. arXiv–2406, 2024.
 663

664 Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woomyoung Park. Gpt3mix:
 665 Leveraging large-scale language models for text augmentation. In *Findings of the Association for*
 666 *Computational Linguistics: EMNLP 2021*, pp. 2225–2239, 2021.
 667

668 Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li,
 669 Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex
 670 and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.
 671

672 Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng.
 673 No more manual tests? evaluating and improving chatgpt for unit test generation. *arXiv preprint*
 674 *arXiv:2305.04207*, 2023.
 675

676 Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-sql with distilled test suites.
 677 In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*
 678 (*EMNLP*), pp. 396–411, 2020.
 679

680 Ruiqi Zhong, Charlie Snell, Dan Klein, and Jason Eisner. Non-programmers can label programs
 681 indirectly via active examples: A case study with text-to-sql. *arXiv preprint arXiv:2205.12422*,
 682 2022.
 683

684 Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from
 685 natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.
 686

687

691 A APPENDIX

694 A.1 SUCCESS AND AGREEMENT RATES BY SCHEMA COMPLEXITY

696 We define schema complexity levels based on the number of columns involved in the gold SQL
 697 query. Specifically, we count the total number of columns across all tables referenced in each gold
 698 query; higher column counts generally correlate with more complex joins, filters, and reasoning
 699 steps. Based on the distribution of complexity levels in the BIRD dataset, we define three buckets:
700 Low Complexity: Questions with a total column count of 1-15. **Medium Complexity:** Questions
 701 with a total column count of 16-60. **High Complexity:** Questions with a total column count of 61
 or more.

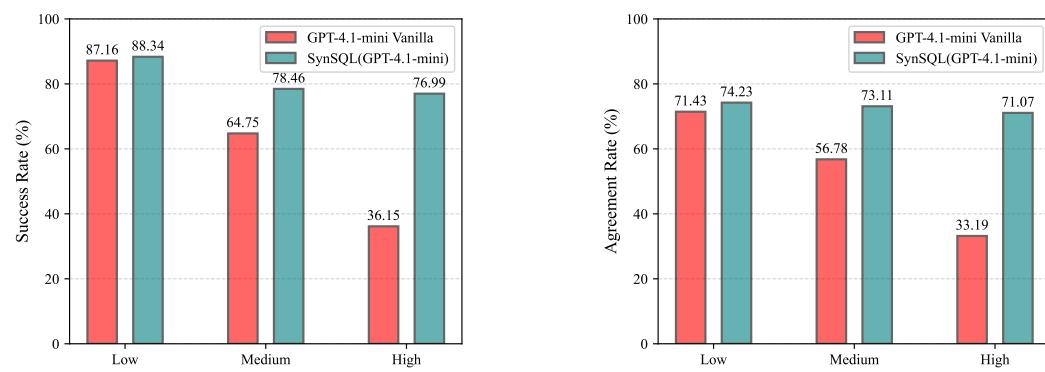


Figure 3: Success and agreement rates of SynSQL vs. GPT-4.1-Mini on BIRD dev set, broken down by schema complexity (Low: 1-15 columns, Medium: 16-60 columns, High: 61+ columns).

A.2 AGREEMENT RATE METRIC DETAILS

The Agreement Rate (AR) metric is inspired by Cohen’s Kappa score Cohen (1960), which measures inter-rater reliability. In our context, we treat the original human-curated database and each synthetic database as two “raters” that evaluate the correctness of SQL queries generated by text-to-SQL models. To compute AR, we first execute each model-generated SQL query on both the original and synthetic databases. Each execution yields a binary outcome: correct (the query produces the expected result) or incorrect (it does not). We then construct a confusion matrix based on these outcomes, counting the number of queries that fall into each of the four possible categories: True Positive (TP): Both databases evaluate the query as correct. True Negative (TN): Both databases evaluate the query as incorrect. False Positive (FP): The original database evaluates the query as incorrect, but the synthetic database evaluates it as correct. False Negative (FN): The original database evaluates the query as correct, but the synthetic database evaluates it as incorrect. Using these counts, we calculate the observed agreement P_o as:

$$P_o = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

Next, we compute the expected agreement P_e by considering the marginal probabilities of each database’s evaluations:

$$P_e = \left(\frac{(TP + FP)(TP + FN)}{(TP + TN + FP + FN)^2} \right) + \left(\frac{(TN + FN)(TN + FP)}{(TP + TN + FP + FN)^2} \right) \quad (4)$$

Finally, the Agreement Rate (AR) is calculated as:

$$AR = \frac{P_o - P_e}{1 - P_e} \quad (5)$$

An AR score of 1 indicates perfect agreement between the two databases, while a score of 0 indicates no better agreement than random chance. Negative values suggest systematic disagreement. By using AR, we can quantify how well the synthetic database preserves the evaluation characteristics of the original human-curated data on a query-by-query basis. There are ranges for interpreting AR scores:

- 0.81 - 1.00: Almost perfect agreement
- 0.61 - 0.80: Substantial agreement
- 0.41 - 0.60: Moderate agreement
- 0.21 - 0.40: Fair agreement
- 0.00 - 0.20: Slight agreement
- < 0.00: Poor agreement

Combining AR with success rate and execution accuracy offers a more complete assessment of synthetic database quality for text-to-SQL evaluation. While execution accuracy reflects overall model performance and preserves model ranking, it does not measure per-query consistency between databases. AR addresses this by quantifying agreement on individual queries, providing finer-grained insight and mitigating potential inflation of aggregate metrics.

A.3 ERROR ANALYSIS OF SUCCESS RATE

To better understand the performance of SynSQL in aligning with question intent, we performed error analysis on a random sample of 500 questions from the BIRD dataset. Of these, 84 questions returned an empty set as the result of the gold query, corresponding to an 83.2% success rate.

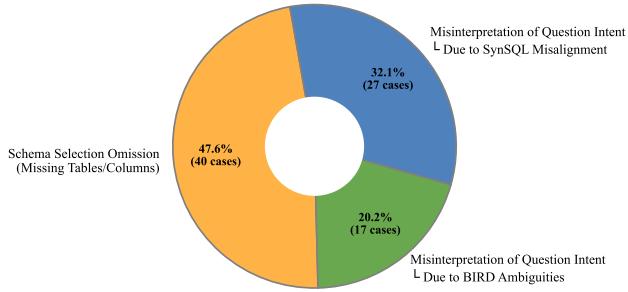


Figure 4: Breakdown of error cases in SynSQL success rate analysis (84 failures out of 500 BIRD dev questions).

Figure 4 summarizes the breakdown of failure cases and their causes. Of the 84 failed questions, 40 were due to schema selector failures. In these cases, schema reduction led to the omission of one or more tables or columns used in the gold query. This does not necessarily mean the generated data is meaningless; rather, the human annotator who wrote the gold query may have targeted different schema elements than the LLM. For example, in question 387 from the `card_games` database (Figure 5):

Question: What are the cards for set OGW? State the colour for these cards.

Evidence: set OGW refers to setCode = 'OGW'

Gold Query: `SELECT id, colors FROM cards WHERE id IN (SELECT id FROM set_translations WHERE setCode = 'OGW')`

Figure 5: An example of schema selection failure. The synthetic data omits the `setCode` column from `set_translations`, leading to a failed query.

SynSQL has generated data for the `setCode` column in `cards`, but omitted the `setCode` column from `set_translations` during schema selection. The gold query joins both tables on `setCode`, leading to failure. However, the synthetic data still contains valid `setCode` values, just not in the joined table. This highlights the challenge of schema selection in open-ended text-to-SQL tasks, where multiple valid interpretations exist.

The remaining 44 failures were due to misinterpretation of question intent. For example, in question 156 from the financial database (Figure 6):

810
811
812
813
814
815
816
817
818

Question: Who is the owner of the account with the largest loan amount?

Evidence: N/A

Gold Query: `SELECT T1.client_id FROM disp AS T1 INNER JOIN account AS T3 ON T1.account_id = T3.account_id INNER JOIN loan AS T2 ON T3.account_id = T2.account_id WHERE T1.type = 'OWNER' ORDER BY T2.amount DESC LIMIT 1`

Figure 6: Example of misinterpretation: the synthetic data contains values such as `owner` (lowercase) in the `type` column, while the gold query expects `OWNER` (uppercase). This case sensitivity mismatch leads to a failed query.

Here, the synthetic database reflects the casing found in the question or evidence, but the gold query expects a different case. Such mismatches between generated data and gold query expectations, especially regarding case sensitivity or value formatting, can result in lower success rates.

Another example is question 90 from the financial database (Figure 7):

Question: How many accounts who have region in Prague are eligible for loans?

Evidence: A3 contains the data of region

Gold Query: `SELECT COUNT(T1.account_id) FROM account AS T1 INNER JOIN loan AS T2 ON T1.account_id = T2.account_id INNER JOIN district AS T3 ON T1.district_id = T3.district_id WHERE T3.A3 = 'Prague'`

Figure 7: An example of misinterpretation due to synthetic data not matching gold query conditions. The synthetic data contains values that do not satisfy the gold query's `WHERE` clause, leading to failure.

The gold query expects `district.A3 = 'Prague'`, but the synthetic data contains values such as `Prague 1`, `Prague 2`, and `Prague 3`. Here, the LLM generated region names with appended numbers, resulting in a mismatch with the gold query's expected value.

Some misinterpretations are due to misalignment between the question and the gold query in the BIRD dev set, rather than errors by SynSQL. For example, in question 803 from the Superhero database (Figure 8):

Question: What is the power ID of cryokinesis?

Evidence: power ID refers to `superpower.id`; `cryokinesis` refers to `power_name = 'cryokinesis'`

Gold Query: `SELECT id FROM superpower WHERE power_name = 'Cryokinesis'`

Figure 8: An example of misinterpretation due to inconsistencies between question/evidence and gold query in the BIRD dev set. The synthetic data aligns with the question, but not the gold query, leading to failure.

In this case, the question and evidence refer to `cryokinesis` (lowercase), while the gold query expects `'Cryokinesis'` (capitalized). The synthetic database contains `power_name = 'cryokinesis'`, resulting in a mismatch with the gold query and subsequently lower success rate. Similarly, in question 758, the question and evidence specify `race = 'human'`, but the gold query expects `race = 'Human'`. In question 415, the question and evidence use `Status`

864 = 'legal', while the gold query expects Status = 'Legal'. The synthetic data generated
 865 by SynSQL reflects the casing found in the question, leading to mismatches with the gold query.
 866

867 In summary, among the 44 misinterpretation cases, 17 stem from insufficient or ambiguous infor-
 868 mation in the BIRD dev set, while 27 are attributable to SynSQL's generation errors. The following
 869 BIRD dev set questions could not be correctly handled by SynSQL due to a lack of necessary
 870 information in the dataset for generating appropriate synthetic data. Such cases are likely to be chal-
 871 lenging for any text-to-SQL system: 22, 73, 180, 309, 415, 758, 769, 803, 815, 818, 871, 1194,
 872 1336, 1472, 1491, 1499, and 1528.
 873

874 A.4 LIMITATIONS

875
 876 SynSQL demonstrates strong performance in generating synthetic databases for text-to-SQL eval-
 877 uation, but it has limitations. The schema selection process may omit relevant tables or columns,
 878 leading to gold queries returning empty results. This remains an active area of research in text-
 879 to-SQL evaluation. The challenge is amplified in our data synthesis setting, where the absence of
 880 actual database contents and value-based retrieval mechanisms makes high-recall schema selection
 881 inherently difficult. However, several practical extensions could improve robustness while main-
 882 taining the minimalist design principle. First, implementing multi-hop schema traversal guided by
 883 LLMs could recover essential join paths and connector tables in complex schemas, adding minimal
 884 columns while significantly boosting recall. Second, employing ensemble methods across multi-
 885 ple LLMs could reduce interpretation variance and yield more stable column predictions. These
 886 approaches offer promising directions for addressing the remaining failure cases while preserving
 887 SynSQL's core advantages.

888 Additionally, SynSQL relies on the assumptions made by the large language models used. If the
 889 LLMs misinterpret the question intent or generate inconsistent data, this can lead to lower success
 890 rates. Incorporating additional constraints or validation steps during data generation could help
 891 mitigate this.

892 893 A.5 ANALYSIS ON EFFECT OF CRITIC COMPONENT

894
 895 As shown in Figure 2, the critic component consistently improves SynSQL's success rate across dif-
 896 ferent LLMs on the BIRD dev set. To provide deeper insight into this improvement, Figure 9 breaks
 897 down the critic's impact across its five evaluation criteria: hint alignment, key integrity, schema cov-
 898 erage, data complexity, data variety, and relevance. The critic consistently enhances performance
 899 across all dimensions, indicating its effectiveness in generating more semantically coherent and di-
 900 verse synthetic databases.

901
 902 **Critic's Role in Avoiding Oversimplified Data Patterns** One potential limitation of LLM-based
 903 synthesis is the tendency to generate overly simplistic or repetitive data patterns, which could arti-
 904 ficially inflate success rates without providing meaningful evaluation coverage. The critic compo-
 905 nent addresses this by explicitly evaluating data complexity and variety as core quality dimensions.
 906 This improvement is reflected in the agreement rate (AR), which quantifies how well the synthetic
 907 database's query evaluations align with those of the original human-curated database on a per-query
 908 basis. As shown in Figure 10, the critic significantly improves AR across all three LLMs tested on
 909 the BIRD dev set. This indicates that the critic component enhances not just semantic alignment
 910 with question intent, but also the fundamental ability to differentiate between correct and incorrect
 911 SQL queries, the core objective of robust evaluation databases.

912
 913 **Feedbacks** Through detailed analysis of critic feedback across our experimental runs, we observed
 914 consistent patterns in how the critic identifies and addresses data quality issues. The critic provides
 915 targeted feedback such as: figure 11 shows an example where the critic highlights deficiencies in
 916 data complexity and variety, prompting the synthesizer to regenerate synthetic data that better aligns
 917 with the question intent and enhances evaluation robustness.

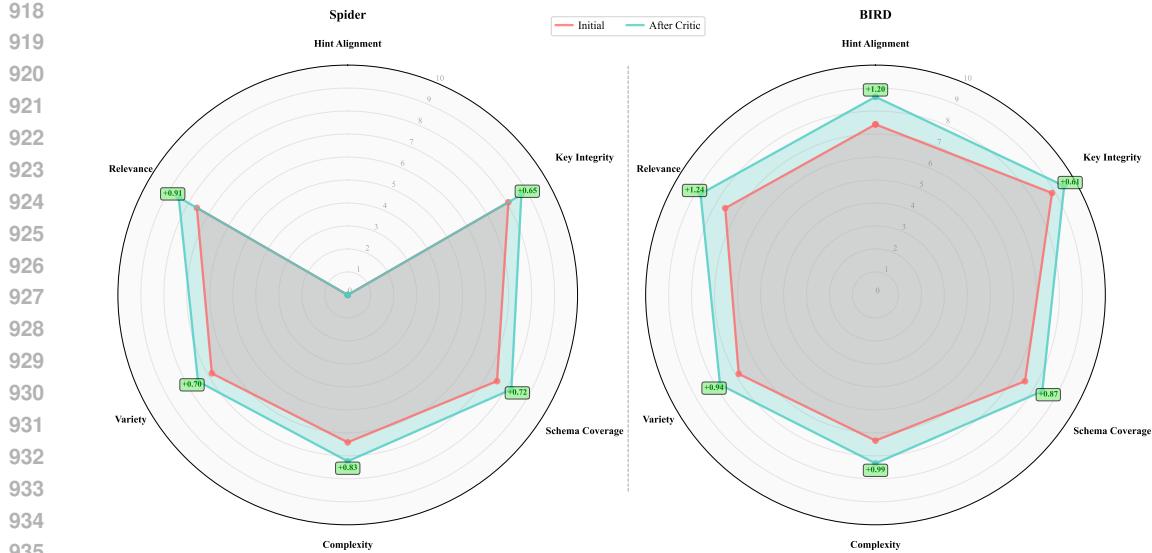


Figure 9: Impact of the critic component on each of the five data quality criteria in SynSQL, using Gemini-2.5-Flash on Spider and BIRD dev sets. Spider results exclude the Hint Alignment criterion as evidence/hint entries are not present in Spider.

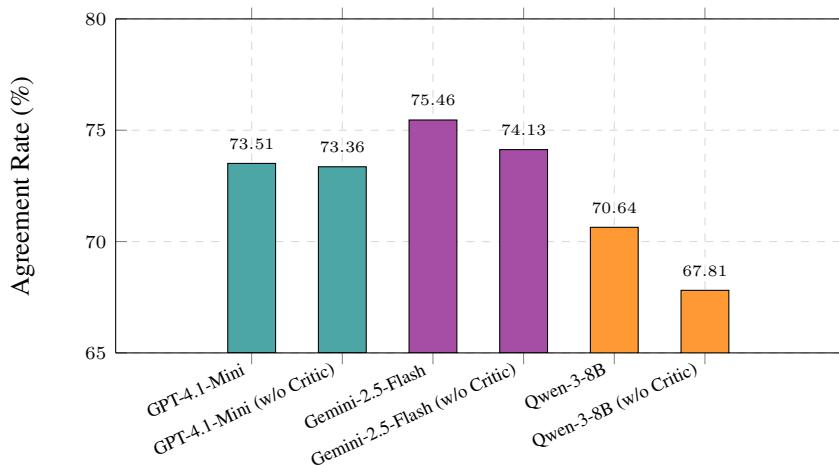


Figure 10: Impact of the critic component on agreement rate (%) of SynSQL with three different LLMs on the BIRD dev set. Showing that the critic improves the database’s ability to distinguish between correct and incorrect SQL queries.

Question: In the race a driver set the lap record for the Austrian Grand Prix Circuit, how long did he spent at the pit stop at that same race?

Evidence: lap record means the fastest time recorded which refers to time, how long spent at pitstop refers to duration

Feedback:

Add explicit marking or clearer linkage of the lap record in lapTimes, possibly by including multiple laps per driver and highlighting the fastest lap. Introduce edge cases such as multiple pit stops per driver and multiple drivers with close lap times to increase complexity. Increase variety by adding more races, circuits, and drivers to better simulate real-world data.

Figure 11: Example of critic feedback highlighting deficiencies in data complexity and variety, prompting regeneration of synthetic data to better align with question intent and evaluation robustness.

972
 973 The critic also frequently identifies key integrity violations, such as non-unique primary keys or
 974 referential integrity breaches. Figure 12 illustrates an example where the critic detects foreign key
 975 violations, leading to regeneration that enforces these constraints and ensures schema integrity.
 976
 977
 978
 979
 980
 981

982
 983 **Question:** What is the total amount of Community College District closure in 1989 in the city
 984 of San Francisco?
 985
 986

Evidence: N/A

Feedback:

Ensure all CDSCode values in "schools" exist in "frpm" and "satscores" to maintain foreign key
 989 integrity.
 990

Increase complexity by including more diverse DOCType values and closure scenarios.
 991

Add more variety in StatusType or closure reasons to enrich the dataset.
 992

993
 994 Figure 12: Example of critic feedback highlighting issues in foreign key integrity, leading to regeneration that
 995 enforces schema integrity.
 996
 997
 998
 999
 1000
 1001
 1002

1003 More examples of critic feedback are shown in Figures 13 and 14, demonstrating the critic's consis-
 1004 tent role in identifying and rectifying data quality problems.
 1005
 1006
 1007
 1008
 1009
 1010
 1011

1012
 1013 **Question:** How many patients with a normal level of complement 3 have a P pattern observed
 1014 in the sheet of ANA examination?
 1015

Evidence: normal level of complement 3 refers to $C3 > 35$; have a P pattern observed in the
 1016 sheet of ANA examination refers to ANA Pattern = 'P'; Should compute the number of distinct
 1017 ones

Feedback:

1019 Adjust the 'C3' values in the 'Laboratory' table to better reflect the "normal level of complement 3
 1020 refers to $C3 > 35$ " hint. Ensure a good mix of values both above and below 35, with a clear
 1021 distinction for "normal" cases. For example, include more values significantly above 35 for
 1022 "normal" cases, and values significantly below 35 for "abnormal" cases.
 1023

1024
 1025 Figure 13: An example of critic feedback

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

Question: Which country is the oldest driver from?

Evidence: date of birth refers to drivers.dob; The larger the birthday value, the younger the person is, and vice versa

Feedback:

Increase the variety and range of birth dates to better highlight the oldest driver and include edge cases such as multiple drivers born on the same day or very close dates. Add explicit foreign key relationships or at least ensure driverId is consistently referenced across tables to improve schema integrity. Expand the drivers table with more entries and nationalities to increase variety and complexity. Although lapTimes and results are less relevant, adding meaningful data or relationships could improve overall schema coverage and complexity.

Figure 14: An example of critic feedback

This systematic feedback mechanism ensures that subsequent iterations produce more robust test databases that can effectively distinguish between semantically correct and incorrect SQL queries. Overall, the critic’s feedback focuses on: (1) key integrity and schema coverage to ensure structural validity, (2) presence of edge cases and boundary values, (3) diversity in categorical attributes, (4) realistic distributions that reflect real-world data patterns while aligning with question intent, and (5) inclusion of potential query failure scenarios. This multi-dimensional assessment prevents the framework from converging on overly simplistic data that might mask SQL generation errors, ensuring that high success rates reflect genuine semantic alignment and increases the robustness of evaluation, rather than artificially accommodating weak queries.

A.6 EVALUATION WITH STRONGER LANGUAGE MODELS

In our main experiments, we focused on evaluating SynSQL with three small size LLMs: GPT-4.1-Mini, Gemini-2.5-Flash, and Qwen-3-8B. To further assess SynSQL’s performance, we conducted additional experiments using a more powerful model, GPT-4.1. The results, presented in Table 5, demonstrate that SynSQL continues to outperform the vanilla GPT-4.1 model across all metrics on the BIRD dev set.

Table 5: Performance comparison of SynSQL using GPT-4.1-Mini against GPT-4.1 on BIRD dev set.

Method	Success Rate (SR)	Agreement Rate (AR)	
		DAIL-SQL	DIN-SQL
BIRD (Original)	99.87	100.00	100.00
Vanilla GPT-4.1-Mini	69.43	58.57	61.88
SynSQL(GPT-4.1-Mini)	82.07	73.51	78.90
Vanilla GPT-4.1	76.09	66.52	70.58
SynSQL(GPT-4.1)	86.51	74.69	79.51

A.7 EVALUATION ON SPIDER 2.0

We also evaluated SynSQL on the Spider 2.0 dataset (Lei et al., 2024), which comprises 632 real-world text-to-SQL workflow problems derived from enterprise-level database use cases. Since most of these problems are based on Snowflake and BigQuery databases, we focused on the 135 questions that can be executed on SQLite to ensure compatibility with our synthetic database generation framework. We then evaluated all questions that have gold SQL queries provided.

As shown in Table 6, SynSQL outperforms the vanilla LLM baseline on success rate demonstrating its effectiveness in generating synthetic databases that align with question intent even in more complex, real-world scenarios.

1080

1081 Table 6: Performance comparison of SynSQL on Spider 2.0 dataset.

Method	Success Rate (SR)
Vanilla GPT-4.1-Mini	54.17
SynSQL(GPT-4.1-Mini)	58.33

1085

1086 A.8 EVALUATION ON ADVANCED TEXT-TO-SQL SYSTEMS

1087 To further validate SynSQL’s effectiveness, we evaluated its performance using three advanced text-
1088 to-SQL systems: Gemini-SQL (Multitask SFT + Gemini-2.5-Pro) (Pourreza & Kubik, 2025), which
1089 ranks at the top of the BIRD leaderboard in the single-model track at the time of writing, OmniSQL-
1090 32B (Li et al., 2025), and CSC-SQL-32B (Sheng & Xu, 2025). These models represent the state-
1091 of-the-art in text-to-SQL generation and provide a robust benchmark for assessing the quality of
1092 synthetic databases generated by SynSQL.1093 As shown in Table 7, SynSQL consistently outperforms vanilla LLM baselines across all three ad-
1094 vanced text-to-SQL systems on the BIRD dev set and achieves substantial agreement rates based
1095 on Cohen’s Kappa score range. Notably, SynSQL maintains the ranking of text-to-SQL models ob-
1096 served with the original human-curated databases, demonstrating its ability to preserve evaluation
1097 fidelity.1098 These results underscore SynSQL’s versatility and effectiveness in generating high-quality synthetic
1099 databases that facilitate robust evaluation of text-to-SQL systems, even when leveraging cutting-
1100 edge models.

1101

1102

1103 Table 7: Performance comparison of SynSQL and baseline methods on BIRD dev set using ad-
1104 vanced text-to-SQL systems: Gemini-SQL, OmniSQL, and CSC-SQL. AR: Agreement Rate (%),
1105 EX: Execution Accuracy (%), Δ : Difference in Execution Accuracy between original and synthetic
1106 databases.

Method	Gemini-SQL			OmniSQL			CSC-SQL		
	AR↑	EX	Δ ↓	AR↑	EX	Δ ↓	AR↑	EX	Δ ↓
BIRD (Original)	100.00	72.10	-	100.00	66.75	-	100.00	71.12	-
Vanilla GPT-4.1-Mini	48.67	84.68	12.58	47.72	80.51	13.76	47.70	82.20	11.08
Vanilla Gemini-2.5-Flash	56.82	82.86	10.76	55.71	78.42	11.67	55.11	80.70	9.58
SynSQL(Qwen-3-8B)	65.49	74.64	2.54	61.62	68.45	1.70	62.05	73.19	2.07
SynSQL(Gemini-2.5-Flash)	68.67	73.21	1.11	68.73	66.69	0.06	65.14	70.08	1.04
SynSQL(GPT-4.1-Mini)	65.58	73.14	1.04	65.61	65.71	1.04	63.98	68.12	3.00

1116

1117

1118 A.9 REALISM AND MINIMALISM OF SYNTHETIC DATABASES

1119

1120 SynSQL-generated databases are not only realistic but also minimal and lightweight. This property
1121 is particularly valuable in scenarios where gold queries are unavailable, not only during synthesis but
1122 also for evaluation. For example, in production or cold-start settings, it is crucial to inspect and val-
1123 idate the generated database, either through human-in-the-loop processes or by generating expected
1124 outputs via table reasoning. The compactness of SynSQL databases facilitates such inspection and
1125 validation, making them practical for robust evaluation even when large-scale or gold-standard an-
1126 notations are not accessible.

1127

1128 We saw in Figure 4 that in SynSQL we have an average of 8.37 columns for BIRD and 6.71 columns
1129 for Spider to answer a query, significantly fewer than the full schemas of 75.56 and 24.55 columns
1130 respectively. We see an example of this in question 1000 from the formula_1 database (see Fig-
1131 ure 15), SynSQL generates a minimal database with only 2 tables and 9 columns, compared to the
1132 original database’s 13 tables and 94 columns. The synthetic database sufficiently covers the question
1133 and relevant edge cases while being just 20KB in size, whereas the original is 21,836KB, making
SynSQL’s output much easier to inspect and validate. In contrast, synthetic databases generated
by prior work such as TestSuiteAccuracy (TSA) often contain random values from fuzzing and are

1134 typically as large as the original databases. As illustrated in Figures 16 and 17, which show the
 1135 entirety of data generated for this question by SynSQL, the synthetic data includes realistic values
 1136 that closely match the question intent.
 1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

Question: Which racetrack hosted the most recent race? Indicate the full location

Evidence: full location refers to location+country; most recent race = MAX(date)

Gold Query: `SELECT T1.location FROM circuits AS T1 INNER JOIN races AS T2 ON T1.circuitId = T2.circuitId ORDER BY T2.date DESC LIMIT 1`

Figure 15: An example from the `formula_1` database (question 1000).

Moreover, SynSQL ensures that values within each row are meaningfully related and contextually accurate. For example, if a row in the `races` table has the year set to 2024, all corresponding data in that row (such as race name or date) is consistent with that year. Similarly, in the `circuits` table, if the location is `Monza`, the country is set to `Italy`, reflecting the real-world fact that there is a Formula 1 Grand Prix held in Monza, Italy. This level of realism and consistency, both within rows and across related tables, is achieved by leveraging LLMs to generate data that maintains semantic coherence and factual alignment.

<code>raceID</code>	<code>year</code>	<code>circuitID</code>	<code>name</code>	<code>date</code>
101	2022	1	British Grand Prix	2022-07-03
102	2022	2	Monaco Grand Prix	2022-05-29
103	2022	3	Japanese Grand Prix	2022-10-09
104	2022	4	United States Grand Prix	2022-10-23
105	2022	5	Italian Grand Prix	2022-09-11
106	2023	1	British Grand Prix	2023-07-09
107	2023	2	Monaco Grand Prix	2023-05-28
108	2023	3	Japanese Grand Prix	2023-10-08
109	2023	4	United States Grand Prix	2023-10-22
110	2023	5	Italian Grand Prix	2023-09-10
111	2024	1	British Grand Prix	2024-07-07
112	2024	2	Monaco Grand Prix	2024-05-26
113	2024	3	Japanese Grand Prix	2024-10-13
114	2024	4	United States Grand Prix	2024-10-27
115	2024	5	Italian Grand Prix	2024-09-08

Figure 16: Generated synthetic table `races` for question 1000 from the `formula_1` database. The synthetic data contains realistic values that align with the question intent.

<code>circuitID</code>	<code>name</code>	<code>location</code>	<code>country</code>
1	Silverstone Circuit	Silverstone	United Kingdom
2	Circuit de Monaco	Monte Carlo	Monaco
3	Suzuka Circuit	Suzuka	Japan
4	Circuit of the Americas	Austin	USA
5	Autodromo Nazionale Monza	Monza	Italy

Figure 17: Generated synthetic table `circuits` for question 1000 from the `formula_1` database. The synthetic data contains realistic values that align with the question intent.

1182

1183

A.10 INCONSISTENCY EXAMPLES FROM SPIDER DEV SET

1184

1185

1186

1187

There are questions in the spider dev set that the gold query does not align with the content of original test databases. Below are some examples of such inconsistencies, which lead to the observed low success rates for the original Spider databases. SynSQL generates synthetic data that aligns with the

1188 question intent and recovers such inconsistencies. For example, in (Figure 18) the question asks for
 1189 the location and name for all stadiums with a capacity between 5000 and 10000. However, there are
 1190 no such stadiums in the original database, leading to the gold query returning empty results. SynSQL
 1191 generates synthetic data that includes stadiums within this capacity range. Another example is shown
 1192 in (Figure 19), where the question asks for the city and country of the Alton airport. However, the
 1193 original database `flight_2` has the airport name listed as `Alton` , with a trailing space, leading
 1194 to a mismatch with the gold query. SynSQL generates synthetic data that correctly matches the
 1195 airport name as specified in the question.

1196
 1197
 1198
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241

Question: Show location and name for all stadiums with a capacity between 5000 and 10000

Evidence: N/A

Gold Query: `SELECT LOCATION , name FROM stadium WHERE capacity BETWEEN 5000 AND 10000`

Figure 18: An example of inconsistencies between gold query and database contents in the Spider dev set.
 SynSQL aligns with the question, leading to recovery of such inconsistencies.

Question: Which city and country is the Alton airport at?

Evidence: N/A

Gold Query: `SELECT City, Country FROM AIRPORTS WHERE AirportName = "Alton"`

Figure 19: An example of inconsistencies between gold query and database contents in the Spider dev set.
 SynSQL aligns with the question, leading to recovery of such inconsistencies.

A.11 COLUMN SELECTION PROMPT

You are an expert data analyst. Your task is to carefully review the database schema, understand the question, and use the hint to determine which columns from which tables must be populated with synthetic data to fully support answering the question.

This task is for synthetic data generation, NOT for Text2SQL. In this context, RECALL IS MORE IMPORTANT THAN PRECISION. It is better to include more columns than to miss important ones.

Database Schema:
{DATABASE_SCHEMA}

This schema defines the database structure, including tables, columns, primary keys, foreign keys, and relevant relationships or constraints.
You can also rely on the following descriptions for the columns to better understand the nature of data that would be generated for them.

Column Descriptions:
{COLUMNS_DESCRIPTIONS}

Question:
{QUESTION}

Hint:
{HINT}

The hint is intended to guide your attention to the specific elements of the database schema that are essential for addressing the question accurately

Task:
Based on the database schema, question, and hint provided, your task is to determine the columns from tables that need to be populated with data to support the question.
You should also provide the foreign keys that are needed to potentially join the tables, in the context of the question.
For each of the selected columns, explain why exactly it is necessary to generate data for, in order to cover the question. Your explanation should be logical and concise, demonstrating a clear understanding of the database schema, the question, and the hint.

Please respond with a JSON object structured as follows:

```
```json
{
 "chain_of_thought_reasoning": "Your reasoning for selecting the columns, be concise and clear.",
 "table_name1": ["column1", "column2", ...],
 "table_name2": ["column1", "column2", ...],
 ...
 "foreign_keys": ["table_name1.column1 = table_name2.column2, ..."]
}
```

```

Make sure your response includes the table names as keys, each associated with a list of column names that are necessary for generating synthetic data that would be enough to support the question.

For foreign keys, make sure you include foreign keys within tables that are needed to cover the possibility of join, IN CONTEXT OF THE QUESTION AND THE HINT.

For each aspect of the question, provide a clear and concise explanation of your reasoning behind selecting the columns. Only output a json as your response.

Figure 20: The prompt template used for column selection in the schema selector component of SynSQL.

1296 A.12 COLUMN EXPANSION PROMPT

1297

1298

1299 You are an expert data analyst. Your task is to analyze the provided database
1300 schema and a list of already selected columns, and identify the most
1301 semantically similar columns to the selected ones.

1301

1302

1303

1304 Database Schema:
1305 {DATABASE_SCHEMA}

1306

1307 This schema defines the database structure, including tables, columns, primary
1308 keys, foreign keys, and relevant relationships or constraints.1309 You can also rely on the following descriptions for the columns to better
1310 understand the nature of the data that would be generated for them.

1310

1311

1312 Column Descriptions:
1313 {COLUMNS_DESCRIPTIONS}

1314

1315 Already Selected Columns:
1316 {SELECTED_COLUMNS}

1317

1318 Task:
13191320 Based on the database schema, column descriptions, and the already selected
1321 columns, your task is to identify, AT MOST 3 of the most semantically similar
1322 columns, that are:1323 1. Semantically similar to the selected columns but in a different table (e.g.
1324 if Country.id is selected, then Match.country_id would be a similar column)
1325 OR
1326 2. Likely to contain data that would complement the selected columns

1327

1328 Please respond with a JSON object structured as follows:

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

Figure 21: The prompt template used for column expansion in the schema selector component of SynSQL.

1350 A.13 DATA SYNTHESIZER PROMPT
1351

1352

1353 You are an expert data generator for SQL databases. Your task is to create
1354 realistic and challenging test data that will properly test a system's ability
1355 to answer complex questions.

1356

1357 For the given question, use the schema and hint to generate SQLite test data for
1358 the database. The schema identifies the specific tables and columns that are
1359 relevant to the question, and the hint provides guidance on how to structure the
1360 data to make the question answerable.

1361

1362 Use the following instructions for generating the test data:

1363 1- Pay attention to the primary key and foreign key constraints to ensure data
1364 integrity.
1365 2- Make sure the data includes edge cases and is challenging to answer the
1366 question.
1367 3- Include a variety of data that covers different scenarios related to the
1368 question.
1369 4- Generate enough data to make the question answerable but also challenging.
1370 5- The data should be realistic and diverse.
1371 6- Your response should follow the EXACT format of the example, where every line
1372 starts with INSERT. DO NOT group the insert statements and DO NOT put values on
1373 a different line than the INSERT statement.
1374 7- IMPORTANT: For each INSERT statement, ensure the number of values EXACTLY
1375 matches the number of columns in the table. Count the columns carefully in the
1376 CREATE TABLE statement and provide exactly that many values in each INSERT
1377 statement.

1378

1379 {FEEDBACK_INSTRUCTION}

1380

1381 {ONE_EXAMPLE}

1382

1383 Schema of the database with question and hint:

1384

1385 Database: {DB_NAME}

1386

1387 Schema: {DATABASE_SCHEMA}

1388

1389 Question: {QUESTION}

1390

1391 Hint: {HINT}

1392

1393 Figure 22: The prompt template used for data synthesis component of SynSQL.
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403

1404 A.14 DATA CRITIC PROMPT
 1405
 1406
 1407 You are a data critic agent designed to evaluate synthetic data for answering
 1408 natural language questions. Your task is to analyze the generated data and
 1409 determine if it is correct, sufficient, complex, and diverse enough to answer
 1410 the question.
 1411
 1412 You should evaluate the data based on the following criteria:
 1413 1. Hint Alignment: Does the data follow the intent and details of the question
 1414 hint?
 1415 2. Key Integrity: Does the data respect uniqueness and foreign key relationships
 1416 in the schema?
 1417 3. Schema Coverage: Does the data include the relevant columns and relationships
 1418 from the schema?
 1419 4. Complexity: Does the data include sufficient complexity and edge cases?
 1420 5. Variety: Is there enough variety in the data?
 1421 6. Relevance: Is the data directly related to answering the question?
 1422 {ONE_EXAMPLE}
 1423 Question: {QUESTION}
 1424 Database Schema: {DATABASE_SCHEMA}
 1425 Hint: {HINT}
 1426 Generated Data: {GENERATED_DATA}
 1427 Provide a detailed evaluation of the data based on the criteria of Hint
 1428 Alignment, Key Integrity, Schema Coverage, Complexity, Variety, and Relevance.
 1429 For each criterion, provide a score from 1-10 and specific feedback on what
 1430 aspects need improvement.
 1431 If there are issues such as incorrect data, violations of key integrity (e.g.,
 1432 non-unique or missing foreign keys), or other schema-related errors, provide
 1433 clear and actionable feedback to help address and resolve these problems.
 1434 When providing feedback, consider that having more data is usually more
 1435 beneficial, provided it does not distract from or obscure the key information
 1436 required to answer the question. Try not to recommend reducing the data.
 1437 Finally, determine if the data meets the minimum quality criteria to answer the
 1438 question effectively.

Figure 23: The prompt template used for data critic component of SynSQL.

A.15 THE USE OF LARGE LANGUAGE MODELS (LLMs)

1441 In accordance with the ICLR 2026 Code of Ethics, we acknowledge that large language models were
 1442 used to assist with the polishing of the writing in this paper.