ECQL: An Executable Benchmark for Auditable Corpus Query Semantic Parsing

Yuxi Zheng[1] **Yeqing Liu**[2] **Ziming Zhou**[1]
Xiaodong Wang[3] **Yongwei Zhang**[4] **Jingsong Yu**[2]
[1]Department of Chinese Language and Literature
Peking University, Beijing 100871, China
[2]School of Software and Microelectronics
Peking University, Beijing 102600, China
[3]School of Electronic and Computer Engineering
Peking University, Shenzhen 518055, China
[4]Institute of Linguistics, Chinese Academy of Social Sciences
Beijing 100732, China
{zhengyuxi,liuyeqing}@stu.pku.edu.cn
{zhouziming,wangxiaodong21s}@stu.pku.edu.cn
zhangyw@cass.org.cn YJS@ss.pku.edu.cn

Abstract

Corpus Query Language (CQL) supports precise corpus analysis but requires expertise in annotations, regular expressions, structural scopes, and platform-specific syntax. Natural-language interfaces simplify access, but their answers can be difficult to inspect, reproduce, or revise. We study Text-to-CQL as executable semantic parsing, mapping intents to inspectable queries. We introduce ECQL, a Sketch-Engine-oriented benchmark of 3,572 natural language–CQL pairs comprising 811 tutorial-derived and 2,761 synthetic examples, with a default split of 3,215 training and 357 test examples. ECQL focuses on Sketch Engine normalization, CQL Diagnostic Profile (CQL-DP) annotations, controlled semantic contrasts, audit-oriented metadata, and generalization protocols, alongside an evaluation suite covering syntactic validity, exact match, canonical match, and execution match. The full 357-example headline test set is independently double-annotated and adjudicated, with 96.9% agreement for intent clarity, 94.7% agreement for CQL–intent alignment, and a 3.1% post-adjudication intent-match error rate. Experiments with prompting, retrieval-augmented prompting, syntax repair, DeepSeek-R1 reasoning prompting, a GPT-4o Agent/API alternative, and supervised fine-tuning reveal persistent difficulties with corpus-query semantics. The best fine-tuned model reaches 85.6% execution accuracy on the random split, but drops to 68.9% on template-disjoint evaluation. ECQL provides a reproducible testbed for auditable semantic parsing over specialized linguistic query languages.

1 Introduction

Semantic parsing has made substantial progress in mapping natural language to executable programs, with Text-to-SQL as the dominant reference task (Zhong et al., 2017; Yu et al., 2018; Wang et al., 2020). However, many linguistic research workflows do not query relational databases. They query tokenized, lemmatized, and structurally annotated corpora through specialized corpus query languages. In these settings, a query may need to constrain word forms, lemmas, part-of-speech (POS) tags, regular expressions, sentence boundaries, paragraph scopes, and relations among matched tokens.

Corpus Query Language (CQL) is a common formalism for such corpus searches. It is supported, with dialect differences, by platforms such as Sketch Engine (Kilgarriff et al., 2014), CQPweb (Hardie, 2012), BlackLab (de Does et al., 2017), and the IMS Open Corpus Workbench (Evert and The CWB Development Team, 2022). CQL is powerful because it exposes linguistically meaningful constraints. It is also difficult for many users because it combines corpus-specific annotation schemes, regular-expression syntax, and structural operators.

This paper treats Text-to-CQL not as a way to make users memorize CQL, but as an auditable intermediate representation for corpus analysis. The goal is to produce a formal query that users can inspect, edit, save, share, execute, and rerun. This framing is important because corpus-linguistic claims often depend on the exact retrieval condi-

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 35–44
November 6-10, 2026 ©2026 Association for Computational Linguistics

tion used to collect evidence. A natural-language answer may be useful for exploration, but it is less suitable as a durable research artifact.

1.1 Why Generate CQL Instead of Direct Answers?

Direct natural-language interfaces and agentic corpus tools are useful alternatives, and ECQL should be evaluated against them. We nevertheless argue that CQL generation is valuable for three reasons. First, it supports reproducibility: a generated query can be stored with a paper, dataset, or lab notebook and rerun under a declared corpus setting. Second, it supports auditability: researchers can check whether the system searched a word form, lemma, POS tag, regular expression, structural scope, or labeled relation. Third, it supports precision: many corpus questions are about exact linguistic constraints rather than broad topical relevance.

For example, the query `[word="dog"] []{0,3} [lemma="bark"] within <s/>` is not merely asking for text about dogs. It specifies a token sequence, a bounded gap, a lemma condition, and a sentence-level scope. Such constraints are exactly what corpus query languages are designed to express.

1.2 Contributions

Building on the Text-to-CQL task introduced by Lu et al. (2024), ECQL[1] makes four complementary contributions:

1. We normalize 3,572 natural language–CQL pairs to the Sketch Engine dialect across 12 operator categories: 811 tutorial-derived and 2,761 synthetic examples, with a default split of 3,215 training and 357 test examples.

2. We provide CQL-DP annotations and audit-oriented metadata that connect query operators, evaluation-risk flags, provenance, and annotation/adjudication records.

3. We provide 96 controlled semantic contrasts to diagnose CQL-specific failures that aggregate execution scores can obscure.

4. We provide generalization protocols for corpus-query evaluation, including template-disjoint and source-disjoint splits, and apply them to prompting, retrieval augmentation, repair, reasoning prompting, an agent/API alternative, and supervised fine-tuning.

[1]Dataset: https://github.com/yuzicx/ECQL.

2 Related Work

Natural-language interfaces for corpus analysis. Corpus tools have long balanced formal query expressiveness and usability. Query-language design has been studied as a tradeoff among expressiveness, learnability, and system fit (Jarke and Vassiliou, 1985). CQPweb emphasizes accessible corpus querying while retaining CQP/CQL power (Hardie, 2012); Sketch Engine supports large multilingual corpus workflows for lexicography and linguistic research (Kilgarriff et al., 2014); BlackLab provides indexed corpus search infrastructure (de Does et al., 2017). Prior usability work notes that query languages can be a barrier for non-expert users (Gärtner and Kuhn, 2018). ECQL targets this interface problem while preserving a formal, inspectable query artifact.

Text-to-CQL and corpus query generation. Lu et al. (2024) introduced the Text-to-CQL task and a bilingual Chinese–English dataset, evaluated LLM prompting and fine-tuned model baselines, and established exact match (EM), valid accuracy (VA), execution accuracy (EX), and CQL-BLEU metrics. Their contribution encompasses the task, data, methods, and evaluation; template-based generation is a dataset-construction technique, not the scope of their approach. Zhang et al. (2026) also explore LLM-based corpus query generation. ECQL does not claim to introduce Text-to-CQL, these baseline families, or these established metrics. Its complementary contribution focuses on normalization to the Sketch Engine dialect, CQL-DP annotations, controlled semantic contrasts, audit-oriented metadata, and corpus-query generalization protocols. These components support focused analysis of lexical and annotation constraints, regexes, gaps, structural scopes, and relational/global conditions.

Executable semantic parsing beyond SQL. Text-to-SQL work provides useful metrics and modeling patterns, including cross-domain generalization, execution-guided evaluation, schema linking, and constrained decoding (Finegan-Dollak et al., 2018; Yu et al., 2018; Scholak et al., 2021; Liu et al., 2025). However, corpus query generation differs from SQL in important ways. The target program is typically compact, token-centered, and strongly tied to annotation schemes

and corpus structures. This makes CQL closer to a specialized domain-specific language than to a general relational query language.

Evaluation of executable queries. Exact match can undercount semantically equivalent programs, while execution accuracy can overcount programs that happen to return the same result in one environment. Prior Text-to-SQL evaluation work shows that split design and query equivalence strongly affect measured generalization (Finegan-Dollak et al., 2018). ECQL therefore separates string matching, canonical string matching, syntactic validity, live execution, controlled semantic tests, and expert human intent match. This separation is central to the benchmark design.

3 Task and Benchmark Design

3.1 Task Definition

Given a natural-language corpus-search request x, a system outputs one Sketch Engine CQL query y. The intended output is a compact, executable, auditable query, not an explanatory answer. Systems may use prompting, retrieval-augmented prompting, repair loops, supervised fine-tuning, or agentic strategies, but evaluation records the final CQL and, where available, the retrieved result set.

3.2 Scope

ECQL targets Sketch Engine CQL syntax. This scope is deliberate: CQL dialects differ across platforms, and a benchmark must declare one execution setting before reporting executable metrics. Cross-platform CQL conversion is treated as a limitation and future direction.

3.3 Benchmark Design Requirements

We design ECQL around four requirements. First, outputs must be *executable*: a prediction should be a query that can be sent to a declared corpus system, not merely a paraphrase of the user's intent. Second, outputs must be *inspectable*: the linguistic constraints over word forms, lemmas, tags, regular expressions, gaps, structures, and labels should be visible to users and evaluators. Third, evaluation must be *contrastive*: a benchmark should include cases where small formal differences, such as `word` versus `lemma`, change the intended retrieval set. Fourth, splits must test *generalization*: systems should not receive credit for memorizing recurring tutorial templates.

Interface	Evidence exposed	Main risk
Direct NL answer	Final prose answer	Hard to audit or rerun
Agent/API retrieval	Logs and result sets	Often verbose and non-portable
CQL generation	Formal executable query	Requires validation of syntax and semantics

Table 1: Interface comparison motivating ECQL as auditable executable semantic parsing rather than direct question answering.

View	Subset	Count	Share
Provenance	Tutorial-derived	811	22.7
Provenance	Synthetic	2761	77.3
Default split	Train	3215	90.0
Default split	Test	357	10.0

Table 2: ECQL composition by provenance and default split. Shares are percentages within the 3,572-example dataset.

These requirements distinguish CQL generation from direct answer generation. A direct natural-language answer can be helpful for exploration, but it often hides the exact retrieval operation. An agentic interface can expose intermediate logs, but such logs are usually longer, less portable, and harder to revise than a single corpus query. CQL occupies a middle position: it is less flexible than arbitrary tool use, but it is compact, domain-native, shareable, and executable. Table 1 summarizes this design choice.

4 ECQL Dataset

4.1 Composition

ECQL contains 3,572 natural language–CQL pairs: 811 tutorial-derived examples and 2,761 synthetic examples. The default split contains 3,215 training examples and 357 test examples. For analysis, we report results by operator family and evaluation-risk flags, both of which are extracted from explicit CQL operators and alignment cues. Table 2 summarizes the dataset.

4.2 Construction

Tutorial-derived CQL examples were collected from 21 CQL tutorial and documentation sources covering Sketch Engine, CQP/CWB, CQPweb, BlackLab, and related corpus-query materials. Non-Sketch-Engine examples were normalized to

Sketch Engine style where possible. Synthetic examples were produced by controlled expansion over operator categories and natural-language styles, then filtered for syntax, intent alignment, and naturalness. This construction follows the broader finding that synthetic query generation can improve executable semantic parsing only when paired with careful filtering, split design, and quality control (Guo et al., 2018; Wu et al., 2021; Weir et al., 2020; Yang et al., 2024; Li et al., 2025).

We make the audit trail explicit. Each artifact row includes provenance, source type, diagnostic fields, validation status, execution-inclusion status, and fields for double annotation and adjudication.

4.3 CQL Diagnostic Profile

CQL-DP is a descriptive profile for slicing evaluation results and flagging cases where execution metrics may be misleading. It does not rank examples as easy or hard. The profile records three field groups:

- **Operator structure**: token constraints, attributes, Boolean operators, gaps, structural scopes, labels, global operators, and nesting.
- **NL-to-CQL mapping cues**: natural-language cues that require lemma, POS tag, scope, negation, gap, regex, or relation mapping.
- **Evaluation-risk flags**: cases where execution equality can be misleading, such as broad queries, empty results, optional gaps, regexes, and labeled conditions.

The CQL-DP profiler assigns 433 examples to C1 atomic, 537 to C2 local constraint, 1,660 to C3 compositional pattern, 853 to C4 structural scope, and 89 to C5 relational/global. These bins identify the dominant operator family used for analysis slices. The profiler also marks 72 examples with evaluation-risk flags.

CQL-DP fields are extracted by rule-based parsing of normalized CQL and rule-derived NL/CQL alignment cues. The extraction rules are fixed before model evaluation and provided in the supplement, so each slice can be traced to explicit CQL operators rather than to a hidden scoring assumption.

4.4 Data Quality and Leakage Checks

The supplemental artifact includes annotation guidelines, row-level audit fields, and a QA agreement script. The full 357-example headline test set was independently annotated by two CQL-familiar annotators and adjudicated. The same annotators also independently checked a separate stratified audit sample of 420 synthetic training examples. The audit judged whether the natural-language intent was clear, whether the gold CQL was syntactically valid, and whether the CQL expressed the intended search. Each dimension used *yes*, *no*, and *unclear* labels. Syntax validity required parseable Sketch Engine style CQL, while intent match required the query to preserve lexical choice, annotation level, token order, scope, and relational constraints.

Subset / dimension	Agr.	κ	Err.
Test intent clarity	96.9	0.84	1.4
Test CQL validity	98.6	0.88	0.8
Test intent match	94.7	0.81	3.1
Synthetic intent clarity	94.5	0.78	2.6
Synthetic CQL validity	97.6	0.82	1.9
Synthetic intent match	91.7	0.76	6.4

Table 3: Two-annotator QA results for the full 357-example test set and a separate 420-example synthetic training audit sample. Agreement (Agr.) and error rate (Err.) are percentages; κ is Cohen's kappa. Test intent-match Err. is the post-adjudication error rate; synthetic rows refer only to the training audit sample.

Disagreements were adjudicated by discussion and recorded in the released audit fields. Across the full test audit, 35 cell-level disagreements out of 1,071 annotation decisions required adjudication. The post-adjudication intent-match error rate on the full test set is 3.1% (11 of 357 examples). The audit also records 5 syntax corrections and 6 cases where the natural-language request was rewritten for clarity while preserving the CQL. In the synthetic training audit, 68 cell-level disagreements out of 1,260 annotation decisions were adjudicated; most involved POS granularity or optional-gap wording. The 6.4% synthetic intent-match error rate is measured only on this 420-example training audit sample; it is not the error rate of the 357-example headline test set. Table 3 reports agreement and error estimates, and Table 4 reports automatic leakage and validity checks.

Check	Count	Rate
Static-valid gold CQL	3572 / 3572	100.0
Default test examples	357 / 3572	10.0
Canonical CQL test overlap	23 / 357	6.4
NL-signature test overlap	4 / 357	1.1
Evaluation-risk flagged examples	72 / 3572	2.0

Table 4: Automatic validation and leakage checks. Overlap counts compare default test examples against the default training split.

Condition	Score policy	Reason
Invalid CQL	VA=0, EX=0	No executable output
Prediction timeout	EX=0; logged separately	Query not reliably executable
Gold/API outage	Live EX unavailable	Infrastructure failure
Both outputs empty	Requires contrast or audit	Avoids vacuous correctness
Different hit counts	EX=0	Result counts do not match

Table 5: Execution failure handling used by ECQL. The protocol separates model failures from infrastructure failures and guards against empty-result artifacts.

5 Evaluation Suite

5.1 Metrics

ECQL reports multiple metrics rather than treating one score as semantic truth:

- **VA**: generated CQL passes static or live syntax validation.
- **EM**: generated CQL exactly matches the reference after whitespace normalization.
- **CEM**: generated CQL matches after conservative canonicalization of quotes, operator spacing, and top-level attribute order.
- **Live EX**: generated and reference CQL return matching results under a declared Sketch Engine corpus, date, result cap, and query configuration.
- **Controlled EX**: generated and reference CQL match on a contrastive controlled semantic suite.

Empty-result matches are not automatically counted as semantic correctness. If both queries return empty results, the example requires controlled-suite evidence or human intent judgment before being treated as correct.

5.2 Execution Protocol and Failure Handling

Live execution is reported only under a declared execution record. For each system output, the evaluator stores the normalized prediction, validation status, corpus identifier, execution date, result cap, timeout status, hit count, and the first 100 concordance identifiers when available. The default live setting uses Sketch Engine-style execution over enTenTen21 with a 30-second timeout and two automatic retries for transient API failures. Multi-corpus sanity checks rerun the same query pair on enTenTen21, the British National Corpus, and an English web sample when corpus access is available. These settings are part of the metric definition: an EX number without corpus, date, cap, and failure policy is not treated as reproducible evidence.

Execution failures are separated from model errors. A syntactically invalid prediction receives VA=0 and EX=0 because the system failed to produce an executable query. A timeout on the prediction after retries is counted as EX=0 but kept separate in the execution reliability report. A corpus API outage that affects both gold and predicted queries is marked as unavailable and excluded from live EX, but the example remains eligible for EM, CEM, and controlled EX. If the gold query returns an empty result and the prediction also returns empty, the example is not automatically correct; the evaluator requires either controlled contrast evidence or adjudicated semantic agreement. This policy prevents broad or malformed queries from receiving accidental credit in sparse corpus states.

5.3 Controlled Contrast Construction

Single-corpus execution can miss semantic errors when two different queries happen to return the same concordance lines. ECQL therefore includes a controlled contrast suite in which each item has a small diagnostic corpus state and one or more minimally different distractor queries. The suite is not used to replace live execution; it is used to test distinctions that live execution may underspecify. For example, if a corpus contains no inflected forms of a lemma, `word` and `lemma` queries may appear equivalent under live EX even though they encode different search intentions.

Controlled cases are built by first selecting an operator family, then creating a minimal set of

Contrast family	Diagnostic purpose
Word vs. lemma	Separates surface-form search from inflectional search
Exact POS vs. wildcard POS	Tests tagset granularity and overbroad matches
Gap and quantifier scope	Checks optional tokens, distance, and token counting
`within` vs. `containing`	Distinguishes search unit from structural containment
Label/global condition	Tests cross-token relations and global constraints
Empty-result trap	Prevents accidental credit for vacuous execution matches

Table 6: Controlled contrast families used to diagnose semantic distinctions that may be invisible under single-corpus execution.

sentences that separates the reference query from plausible confusions. Each case stores the natural-language intent, reference CQL, distractor CQL, expected match identifiers, and the semantic contrast being tested. Empty-result traps are scored by whether the system avoids treating two empty outputs as evidence of equivalence. Table 6 lists the diagnostic families.

5.4 Generalization Splits

The default random split is useful as a compatibility baseline, but it is not sufficient to show generalization, a concern also raised in Text-to-SQL evaluation methodology (Finegan-Dollak et al., 2018; Yu et al., 2018). ECQL therefore provides three additional splits:

- **Template-disjoint**: CQL values are abstracted into skeletons, and train/test skeleton overlap is zero.
- **Source-disjoint**: selected tutorial sources are held out for testing.
- **Real-only**: tutorial-derived examples are evaluated under a source-disjoint setting.

The generated artifact contains 664 template-disjoint test examples with zero skeleton overlap, 193 source-disjoint tutorial-derived test examples with zero source overlap, and 193 real-only source-disjoint test examples.

Split	Train	Test	Skel. ov.	Src. ov.
Random	3215	357	127	1
Template-disjoint	2908	664	0	22
Source-disjoint	3379	193	43	0
Real-only	618	193	38	0

Table 7: Generalization split diagnostics. Skeleton overlap counts are for CQL skeletons; source overlap counts are source identifiers shared between train and test.

6 Experiments

6.1 Systems

We evaluate documentation prompting (DP), manual five-shot prompting (DP+5shot), retrieval-augmented prompting, retrieval plus syntax repair, DeepSeek-R1 reasoning prompting, a GPT-4o Agent/API alternative, and supervised fine-tuning. The retrieval nearest-neighbor baseline is intentionally lightweight: it retrieves the five most similar training requests by lexical overlap and emits the top neighbor's CQL. Retrieval-augmented prompting inserts retrieved examples into the prompt, following evidence that demonstrations shape format and task recognition in in-context learning (Min et al., 2022). The repair setting gives validator feedback for at most two rounds, motivated by constrained decoding and validation methods for formal outputs (Scholak et al., 2021). Main-table Live EX is computed on enTenTen21 under one query configuration; multi-corpus sanity checks are diagnostic and not used for model selection.

All prompting systems use the same output contract: return one CQL query and no prose unless the method is explicitly the Agent/API alternative. Retrieved examples are selected from the training split only using BM25 over lowercased requests and normalized CQL operator mentions. The repair baseline receives only the error class and invalid CQL, never the gold query or result set. DeepSeek-R1 is evaluated with the same prompt and logs only the final query. The GPT-4o Agent/API baseline receives the same request and corpus metadata, may issue at most three API-style retrieval calls, and has no access to gold CQL or gold results. It is scored by result-set agreement but not credited for EM or CEM because it does not necessarily produce a reusable query artifact.

6.2 Main Results

Table 9 reports the main results on the default split. The retrieval nearest-neighbor baseline reaches perfect static validity because it copies gold CQL from training, but only 5.9% EM and 5.9% EX. This confirms that lexical retrieval alone is insuf-

Method	Interface	Reported evidence
DP / DP+5shot	prompt	CQL output
Retrieved 5/10-shot	prompt + retrieval	CQL output
Retrieved + repair	validator loop	repaired CQL output
DeepSeek-R1	reasoning prompt	CQL output
GPT-4o Agent/API	tool use	result-set output
Fine-tuned Qwen/BART	SFT	CQL output

Table 8: Evaluation coverage included in the artifact.

ficient. Retrieval-augmented prompting substantially improves over manual five-shot prompting, and repair mainly improves validity and execution robustness. The GPT-4o Agent/API alternative does not produce a reusable CQL artifact for all cases. The fine-tuned Qwen2.5-72B model remains strongest on the random split, but later split analysis shows a large generalization gap.

6.3 Generalization Results

Table 10 reports execution accuracy across the additional splits. All methods degrade substantially on template-disjoint examples, showing that the random split overestimates generalization to unseen CQL skeletons. The fine-tuned model remains strongest overall, but its 16.7-point drop from random to template-disjoint evaluation shows that specialized query generation is not solved by supervised tuning alone.

6.4 Controlled Semantic Diagnostics

The controlled suite contains 96 contrastive cases covering word/lemma contrast, exact word forms, POS granularity, regex prefixes, gap quantification, within/containing scope, case sensitivity, negation, Boolean alternation, labeled relations, and empty-result traps. On the controlled suite, the best fine-tuned model reaches 78.4% C-EX, while the reasoning model reaches 68.5%. Empty-result traps account for 8.3% of controlled cases; these are scored by controlled fingerprints.

Performance varies sharply by contrast family. The fine-tuned model is strongest on lexical-form contrasts (86.7% C-EX) and case-sensitive matching (84.6%), where the output mostly depends on selecting the correct attribute. It is weaker on gap and quantifier contrasts (72.9%), structural scope contrasts (69.4%), and labeled/global conditions (62.5%). The GPT-4o Agent/API alternative's outputs are less compact and less reusable because they do not always expose a stable formal query.

7 Analysis

7.1 Feature-Level Errors

Coarse categories such as POS wildcard errors or broad query-form errors hide several distinct failure modes. ECQL instead uses feature-level labels: word/lemma confusion, POS granularity, regex escaping, gap or quantifier errors, structural-scope reversal, label/global-condition errors, over-constrained queries, under-constrained queries, and ambiguous natural language. Table 11 reports the distribution for the best fine-tuned model's incorrect examples. POS granularity is the largest category, reflecting mismatches among broad tags such as `V.*`, lexical-verb tags such as `VV.*`, and exact tags such as `VVD`. Structural-scope and label/global-condition errors are less frequent but more damaging because they change the retrieval unit or token relation.

7.2 Representative Failure Cases

Table 12 shows a representative failure that would be easy to miss if evaluation only checked whether a query executes. The natural-language request asks for an inflectional class and an exact plural-noun tag within a sentence. The prediction is syntactically valid and often returns plausible concordance lines, but it changes lemma search into surface-form search and broadens the noun tag. This error is not cosmetic: it excludes forms such as *gave* and *given* while admitting singular and proper-noun matches. A user can inspect the predicted CQL and notice the error, which is precisely why ECQL treats CQL as an auditable intermediate representation.

Two broader patterns appear in the audited errors. First, models often translate natural-language grammatical terms into the nearest broad tag instead of the platform-specific tag needed by Sketch Engine. This explains why POS granularity remains the largest error class even when VA is high. Second, models under-specify structural scope when the request contains ordinary phrases such as "in the same sentence" or "inside a noun phrase." These phrases are easy for humans to treat as background wording, but in CQL they determine whether `within`, `containing`, or a labeled condition is required.

Model / method	Setting	VA	EM	EX	C-EX
Nearest-neighbor retrieval	retrieved 5-copy	100.0	5.9	5.9	18.8
GPT-4o	DP	98.5	36.3	52.7	43.6
GPT-4o	DP+5shot	98.5	34.8	55.7	45.1
Gemini 2.0 Flash	DP+5shot	98.0	38.8	62.7	50.8
GPT-4o	retrieved 5-shot	99.0	43.1	68.9	61.5
GPT-4o	retrieved 10-shot	99.2	44.8	70.3	63.0
GPT-4o	retrieved + repair	99.7	45.4	72.1	65.2
DeepSeek-R1	DP + reasoning	99.4	47.6	73.4	68.5
GPT-4o Agent/API	tool-use result	–	–	76.2	69.7
Qwen2.5-14B-Instruct	supervised fine-tuning	95.5	49.8	64.7	58.3
Qwen2.5-72B-Instruct	4-bit LoRA SFT	98.0	67.7	85.6	78.4
BART-large	supervised fine-tuning	85.6	43.8	58.7	50.4

Table 9: Main results on the default split. C-EX is controlled semantic execution accuracy. The GPT-4o Agent/API alternative is not scored by VA or EM because it need not emit CQL.

Method	Rand.	Temp.	Source	Real
Retrieved 10-shot	70.3	54.8	58.6	61.1
Retrieved + repair	72.1	57.9	61.4	64.0
DeepSeek-R1	73.4	60.2	63.7	65.5
Qwen2.5-72B SFT	85.6	68.9	72.5	75.1

Table 10: Execution accuracy across random, template-disjoint, source-disjoint, and real-only splits.

Error type	Share
POS granularity / tagset mismatch	24.8
Word vs. lemma confusion	16.3
Regex or escaping error	12.8
Gap / quantifier error	11.6
Within / containing scope error	10.5
Over-constrained query	9.3
Label / global-condition error	8.1
Under-constrained query	6.6

Table 11: Feature-level error distribution for incorrect predictions by the best fine-tuned model. Values are percentages over errors.

Field	Example
Intent	Find any form of *give* followed within two tokens by a plural noun in the same sentence.
Gold	`[lemma="give"] []{0,2} [tag="NN2"] within <s/>`
Prediction	`[word="give"] []{0,2} [tag="N.*"] within <s/>`
Diagnosis	Lemma-to-word confusion plus POS over-generalization.

Table 12: Representative valid-but-wrong prediction. The query executes, but it changes the intended linguistic constraints.

7.3 Operator-Family Slices

CQL-DP complements aggregate scores with slices that correspond to explicit query operators. The slices are descriptive rather than hierarchical: they identify whether a prediction must handle lexical attributes, POS tags, regexes, gaps, structural scopes, labels, or global conditions. This makes the analysis auditable because each slice can be verified directly from the reference CQL.

The resulting slices point to a consistent pattern. Atomic lexical constraints are comparatively reliable, while examples involving POS granularity, optional gaps, structural scopes, and labeled or global conditions account for most valid-but-wrong predictions. This analysis is intentionally descriptive: a reader can inspect the extracted operators for each row and verify why an example belongs to a slice. It also suggests that practical Text-to-CQL interfaces should expose generated queries and high-risk operators to users rather than returning opaque answers only.

Interface implications. These findings suggest a conservative deployment model. A Text-to-CQL system should draft a query, display a short paraphrase of the formal constraints, and highlight fragile operators before execution. In particular, POS wildcards, optional gaps, structural scopes, labels, and global conditions should be visible because they account for many valid-but-wrong predictions. This does not require users to become expert CQL programmers; it gives them an inspectable object for correction, rerun, and citation. The same design also supports benchmark evaluation: systems can be compared not only by whether they return plausible concordance lines, but by whether they expose a stable, editable query that preserves the intended linguistic constraints.

8 Conclusion

ECQL reframes Text-to-CQL as a benchmark for auditable executable semantic parsing over a specialized corpus-query DSL. The central claim is not that fine-tuning one model solves CQL generation, but that reliable corpus-query generation requires public data, execution-aware evaluation, controlled semantic diagnostics, generalization splits, and transparent error analysis. The supplemental artifact provides the data, schemas, scripts, and reporting protocol needed to evaluate whether systems generate CQL that is not only syntactically plausible, but inspectable, reusable, and semantically faithful.

Limitations

ECQL is Sketch-Engine-oriented and does not solve dialect transfer across CQPweb, BlackLab, CQP/CWB, and other CQL variants. Synthetic examples may contain artifacts despite filtering and expert audit, especially in rare relational operators. Execution accuracy is not formal semantic equivalence; one corpus may miss contrastive cases, and empty-result matches are especially fragile. The controlled suite mitigates this risk but cannot cover every corpus state. Live corpus execution also depends on Sketch Engine access and corpus licenses. ECQL does not include a full user study; it evaluates whether systems generate auditable queries, not whether a deployed interface improves end-to-end research productivity.

References

Jesse de Does, Jan Niestadt, and Katrien Depuydt. 2017. Creating research environments with BlackLab. In Jan Odijk and Arjan van Hessen, editors, *CLARIN in the Low Countries*, pages 245–257. Ubiquity Press, London.

Stephanie Evert and The CWB Development Team. 2022. *The IMS Open Corpus Workbench (CWB) CQP Interface and Query Language Manual*. CWB version 3.5.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Markus Gärtner and Jonas Kuhn. 2018. Making corpus querying ready for the future: Challenges and concepts. In *Proceedings of the 14th Conference on Natural Language Processing (KONVENS 2018)*, pages 158–164, Vienna, Austria.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from SQL queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1597–1607, Brussels, Belgium. Association for Computational Linguistics.

Andrew Hardie. 2012. CQPweb: Combining power, flexibility and usability in a corpus analysis tool. *International Journal of Corpus Linguistics*, 17(3):380–409.

Matthias Jarke and Yannis Vassiliou. 1985. A framework for choosing a database query language. *ACM Computing Surveys*, 17(3):313–340.

Adam Kilgarriff, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlý, and Vít Suchomel. 2014. The Sketch Engine: Ten years on. *Lexicography*, 1(1):7–36.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing high-quality text-to-SQL data at scale. *Proceedings of the VLDB Endowment*, 18(11):4695–4709.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025. A survey of text-to-SQL in the era of LLMs: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 37(10):5735–5754.

Luming Lu, Jiyuan An, Yujie Wang, Liner Yang, Cunliang Kong, Zhenghao Liu, Shuo Wang, Haozhe Lin, Mingwei Fang, Yaping Huang, and Erhong Yang. 2024. From text to CQL: Bridging natural language and corpus search engine. *Preprint*, arXiv:2402.13740.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

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

Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Cetintemel, and Carsten Binnig. 2020. DBPal: A fully pluggable NL2SQL training pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2347–2361. ACM.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical SQL-to-question generation for cross-domain text-to-SQL parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8974–8983, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, Bangkok, Thailand. Association for Computational Linguistics.

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

Chenghui Zhang, Jing Luo, Xinhui Tu, and Yulin Chen. 2026. Automatic corpus query generation method based on large language model. *Computer Engineering*, 52(2):404–412.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *Preprint*, arXiv:1709.00103.