Prompt Analytics

Jixian Su
Georgia Institute of Technology
Atlanta, USA
jixiansu@gatech.edu

Yusen Zhang
Columbia University
New York City, USA
yz5296@columbia.edu

Eugene Wu
Columbia University
New York City, USA
ewu@cs.columbia.edu

Kexin Rong
Georgia Institute of Technology
Atlanta, USA
krong@gatech.edu

Abstract

Writing prompts today is closer to divination than science: practitioners mix together a cocktail of strategies for what to say and how to say it—how to frame tasks and goals, offer hints, impose restrictions, assign personas, select examples, and specify output formats—in the hope that the resulting concoction will be empirically effective. But which strategies were *actually* effective, and do they transfer to other models, domains, or tasks? While individual outcome entangles their effects with one another and with the setting it ran in, as prompts accumulate over time, recurring strategies create an opportunity to pool evidence, disentangle their effects, and measure transferability.

All of this points towards the need to treat prompts as first class data to analyze—in other words, the need for **Prompt Analytics**. We develop Facet, the first of its kind prompt analytics system built around an extensible data model that represents recurring semantic guidance as *prompt features*, development and evaluation settings as *context attributes*, and observed behavior and performance as *evaluation outcomes*. Applying Facet to a corpus of 4,475 prompts spanning eight domains, we identify 2,987 recurring features and analyze their prevalence and evolution across contexts. To supplement observational data with evidence of individual feature effects, Facet enriches the representation through controlled experiments that compare paired executions of prompt variants differing only in the target feature. We measure the effects of 79 frequently occurring features in Text2DSL generation and mathematics through 6.5 million controlled interventions. We find statistically significant effects in 37% of feature effects measured, with transferability varying across task domains, benchmarks, and models.

Finally, we demonstrate how prompt analytics informs prompt development: feature recommendations improve execution accuracy in 10 of 11 configurations across five published Text-to-SQL pipelines, while features with measured positive effects improve held-out Text-to-Cypher accuracy by 0.7–10.3 percentage points when used to warm-start automatic prompt optimization.

Keywords

Prompt Analytics, Data Management, Large Language Models

1 Introduction

Large language models (LLMs) are increasingly used to perform complex tasks, with prompts serving as the primary interface for specifying how these tasks should be carried out. Developers, researchers and LLMs incorporate knowledge into prompts about how to frame tasks, provide hints or reasoning procedures, impose constraints, select examples, and specify outputs. As prompt engineering and automatic prompt optimization become more widespread [4, 32, 76, 78], they produce growing collections of prompts that capture what practitioners have tried across different tasks and settings. However, this accumulated experience remains largely untapped. A practitioner developing a prompt for a particular task has little systematic way to learn what guidance others have used, how its use varies across settings, and under what conditions it improves performance.

While there lacks a common prompt repository for analysis, a larger limitation, is that each prompt mixes guidance that is tailored and worded for its particular domain, task, and evaluation criteria. Thus, even prompts that share guidance are not directly comparable despite pieces of guidance that recur across disparate settings. For example, a text-to-SQL instruction *"use only provided schema elements"* and a mathematics instruction *"only allowed to use the known conditions to derive new conclusion"* both express the broader principle of not introducing information beyond the task specification. Comparing and reusing such guidance requires identifying these pieces and normalizing their different expressions. Similarly, understanding where guidance is used requires a common representation of the contexts where prompts are developed and evaluated. Third, attributing prompt performance to individual guidance requires disentangling the effects of other guidance and evaluation settings (e.g., model, configurations, benchmark, inputs). This is needed to avoid over-generalizing from a prompt's individual successes or failures when performing prompt engineering [60, 77].

These challenges all point towards the need to identify and extract a common structured representation of prompts, their context, and evaluation conditions, as well as the systematic analysis of prompt collections to aid discovery, prompt engineering, recommendation, and more—what we call **Prompt Analytics**. Facet is the first such prompt analytics system built around an extensible data model that models each execution of a prompt as a row with three classes of attributes: prompt features, context attributes, and evaluation outcomes (Figure 1). Facet identifies recurring guidance in prompt collections and normalizes different textual expressions into *prompt features* that represent the concept rather than textual instantiation. It records the development and evaluation settings, including those custom to Facet users, as an extensible set of *context attributes* (e.g., model class, year, benchmark, etc). These attributes

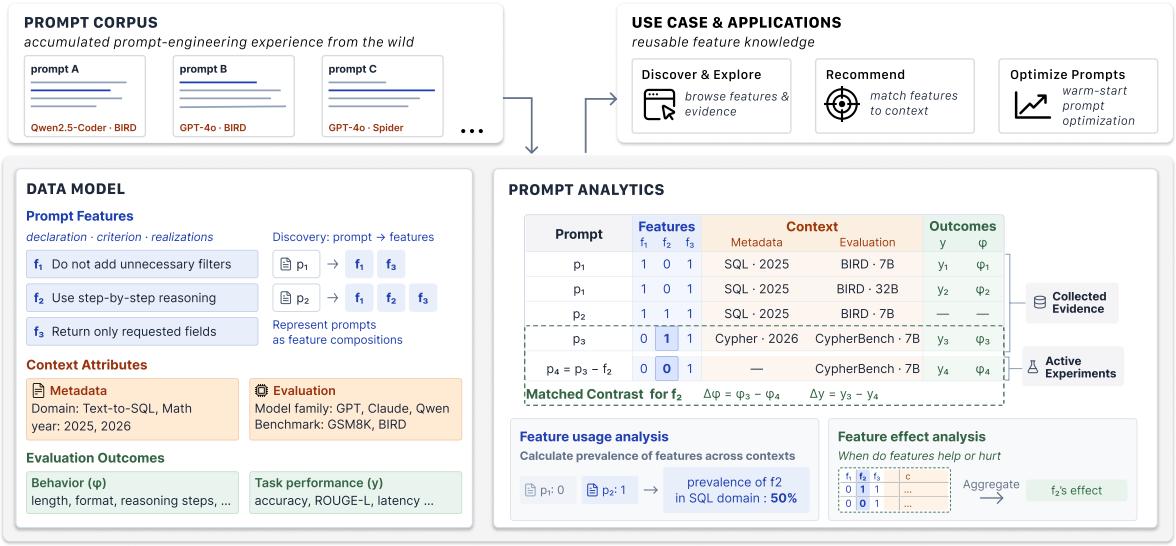


Figure 1: Overview of Facet. Facet represents recurring prompt guidance as reusable features, alongside context attributes and evaluation outcomes. It supports prompt analytics using collected evidence and active experimentation to understand which features recur and when they are effective. The resulting knowledge supports feature exploration, recommendation, and prompt optimization.

provide dimensions for grouping and comparing guidance across settings. When available, Facet records *evaluation outcomes* as well. This representation supports many forms of prompt analytics. For instance, analyses to understand feature prevalence across contexts and feature usage trends; as well as teasing apart how individual features affect model behavior and task performance across contexts.

We populate this representation with 4,475 prompts collected from 1,587 public systems, papers, and repositories spanning eight domains. Facet identifies 2,987 recurring prompt features, whose usage we analyze across the corpus. For example, tool-use features become substantially more prevalent in text-to-SQL prompts in the first half of 2026.

These usage patterns alone do not establish feature effects. Collected prompts often differ in multiple features and evaluation settings. To isolate individual feature effects on a particular context, Facet supports active experimentation by adding or removing individual features while holding other content unchanged. We apply this approach to 50 Text2DSL generation features and 29 mathematics features across six benchmarks and four models, producing 6.5M controlled interventions.

These experiments reveal systematic feature effects across independently written prompt backgrounds, with 37% of feature measurements showing statistically significant effects. Some features generalize across models and task domains. For example, guidance to remain faithful to the stated task recurs in prompts for mathematical problem solving and Text2DSL generation. It has positive estimated effects on accuracy across all six benchmarks spanning these two tasks. Others do not generalize beyond that specific instance. For example, guidance to deduplicate results consistently increases the use of `DISTINCT`, but improves accuracy by 1.8 percentage points on CypherBench while reducing it by 3.2 points on Mind-the-Query, where deduplication is rarely required. Many of these insights emerge from simple aggregations and comparisons over Facet's structured representation.

We show two ways Facet can aid prompt engineering. First, by augmenting five well-tuned Text-to-SQL pipelines with features frequently used in related prompts, Facet improves execution accuracy in 10/11 configurations by up to 4.6 points. Second, using features with measured positive effects to initialize automatic prompt optimization framework GEPA [4] improves held-out accuracy by 0.7–10.3 points across 4 text-to-Cypher graphs.

In summary, this work makes the following contributions:

- We introduce *prompt analytics*, the systematic analysis of recurring guidance in prompt collections, its usage across contexts, and its effects on model behavior and task performance. We develop Facet, a system that enables these analyses through an extensible data model and active experimentation.
- We curate and release an open-source prompt corpus spanning eight domains and a library of recurring semantic features.
- We analyze feature usage across the corpus and systematically measure feature effects across six benchmarks and four models, revealing how prompting practices evolve and how feature effects vary and transfer across contexts.
- We demonstrate the utility of prompt analytics for prompt development through feature recommendations for existing prompts and effect-informed initialization of automatic prompt optimization.

2 System Overview and Workflows

Figure 1 illustrates the main components of Facet: a shared representation of prompts, features, contexts, and evaluation outcomes; analytical workloads over this representation; and active experimentation to generate additional evidence. Our goal is to curate a shared corpus of prompts and best practices and provide services that support human and agentic prompt engineering. Users can

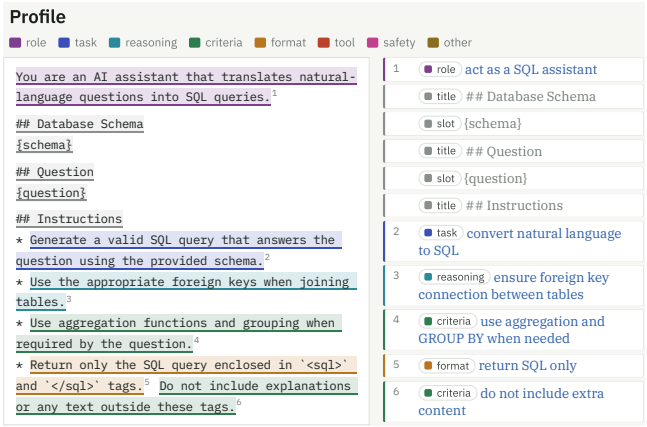


Figure 2: Prompt profiling in the web-based tool. Extracted features are shown alongside the prompt spans that express them.

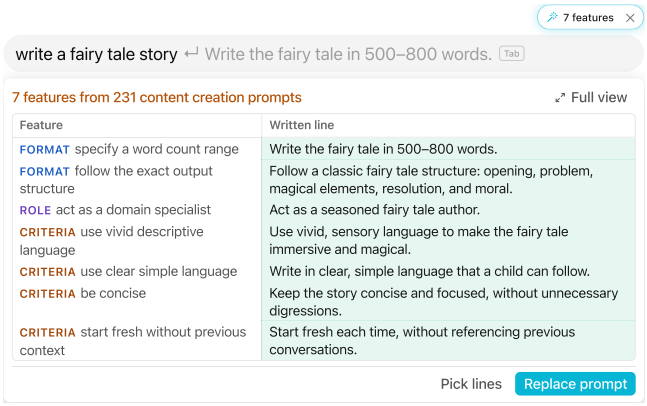


Figure 3: Feature recommendations in the in-browser extension. Users can select suggested realizations to incorporate into their prompts.

interact with Facet through graphical interfaces for prompt exploration and editing, or through APIs for programmatic access. The following workflows illustrate how users contribute prompts, examine prompting practices, and use the resulting evidence to revise their prompts.

Populating and profiling. Facet provides a curated corpus spanning eight domains and a library of recurring semantic features. Users can submit their own prompts, from which Facet extracts features, matches them to the existing library, and incorporates newly identified features (Section 3). Facet provides a web-based tool for users to profile the prompt and examine the extracted features alongside the text spans that express them. For example, Figure 2 shows a text-to-SQL prompt with features concerning table joins, aggregation, and output formatting. This view makes the prompt's feature composition explicit. Users can also associate their prompts with context attributes and available evaluation outcomes. Facet provides a default set of context attributes (*e.g.*, publication year and intended model) and outcome measures. Users can supply values for these attributes and measures or extend them with their own.

Exploring and querying. Users can browse the prompt corpus and feature library and perform analytical workloads over the shared representation (Section 4.2). For example, grouping prompts by domain, task, or publication period reveals which features recur in particular contexts and how their use changes over time (Section 6.2). When suitable controlled comparisons are available, users can also estimate feature effects and compare them across evaluation contexts (Section 7). Facet supports these analyses through standard relational operations without requiring a dedicated query language.

Revising prompts. Users can draw on features found in related prompts when revising their own prompts. We developed an in-browser extension that retrieves candidate features and presents textual realizations that users can select and apply in the prompt editor (Figure 3). In the illustrated example, a request to write a fairy tale receives suggestions concerning length, narrative structure, and writing style. These recommendations reflect observed prompting practices; Section 8.1 applies this workflow to improve five published Text-to-SQL pipelines.

Active experimentation. Users can evaluate candidate edits by providing prompt, evaluation inputs, and corresponding grader. Facet runs controlled experiments to test candidate features drawn from the corpus or user-specified edits, measuring changes in task performance and model behavior (Section 5). Over time, this helps build a knowledge base of effective and ineffective features that can bootstrap or improve prompts (Section 8.2). The service can also crowdsource sparse regions of the matrix that are most informative to populate.

3 Prompts as Reusable Features

To compare and reuse guidance across independently written prompts, Facet represents prompts in terms of recurring semantic features rather than their surface strings. This section establishes a correspondence between prompt strings and reusable feature compositions, and builds on this representation to support feature-level editing. Section 3.1 introduces features, their textual realizations, and the representation of prompts as feature compositions. Section 3.2 describes our current implementation for recovering this representation from prompt corpora. Section 3.3 uses the representation to add and remove individual features while preserving the surrounding prompt. Appendix E shows an example of one text-to-SQL prompt through decomposition, feature discovery and editing.

3.1 Prompt Features

Features and realizations. Prompts that differ as strings can express much of the same guidance. We define a *feature* f as a reusable unit of semantic guidance, independent of how that guidance is worded. A text span that expresses f is a concrete *realization* of the feature in a given domain.

A feature can therefore have many textual realizations. For example, Example 3.1 shows the feature *remain faithful to the stated task*, which instructs the model not to introduce information beyond what the task specifies. In math, this guidance is realized as instructions to derive conclusions only from known conditions, whereas in Text-to-SQL, as instructions to select only requested columns. Representing these instructions as realizations of the same

feature allows FACET to recognize and compare this guidance across independently written prompts and domains.

To make features explicit and reusable, FACET maintains a feature record containing 1) a canonical declaration that summarizes the feature's intent, 2) a recognition criterion for determining whether a prompt expresses it, and 3) observed realizations when available.

Prompts as feature compositions. A prompt typically combines reusable guidance with task-specific and structural content. We represent a prompt as a template P with optional slots for task-specific input q. At execution time, q provides the values inserted into these slots, such as a user request, document, or environment state; for a fully instantiated prompt, $q = \varnothing$.

Let $F = \{f_1, f_2, ...\}$ denote a feature composition associated with P. We use $P \sim F$ to denote the correspondence between the prompt and its feature representation: F is an abstraction of the reusable guidance in P, while P is a textual realization of that guidance. The abstraction discards particular wording, organization, and other prompt-specific content. Therefore, distinct prompt strings can realize the same feature composition.

We move between these representations via two operations. *Discovery* ($P \longrightarrow F$) identifies the feature composition expressed by an existing prompt, while *realization* ($F \longrightarrow P$) expresses a feature composition as prompt text. These operations provide the basis for comparing and editing prompts at the feature level.

Example 3.1: Feature record

FEATURE. Remain faithful to the stated task.
Canonical declaration. Solve exactly the stated task; add nothing more.
Recognition criterion. When the realization ties the content of the answer to what the task supplies.
Observed realizations.

(MATHEMATICS) "Only allowed to use the known conditions to derive new conclusions"; "Do not include any information that would simplify the question in any way."; "Do not add steps that are not explicitly supported by the solution."

(TEXT-TO-SQL) "Just select columns that are explicitly requested in the query"; "Do not add extra columns or omit requested fields."

(TEXT-TO-CYPHER) "Use only the provided relationship types and properties in the schema."

3.2 Feature Discovery

Feature discovery instantiates the $P \longrightarrow F$ process by recovering reusable guidance from existing prompts. Features may also be specified manually by a practitioner. Our current implementation uses corpus-based discovery: it first decomposes each prompt into candidate units of instructions, and then resolves semantically similar units across prompts into shared reusable features.

Hierarchical decomposition. Decomposition identifies candidate units of guidance within each prompt. FACET first constructs a lossless structural view of each prompt. Beginning with the complete prompt, a language-model worker recursively divides a composite span into coherent components. The first division may separate, for example, a role specification, a reasoning procedure, and a block of rules. Each component is then examined in the same way and divided further while it still contains multiple distinct pieces of guidance. Recursion stops when a span expresses a single indivisible instruction.

This recursive decomposition induces a hierarchy of nested spans over the original prompt: each span is the parent of the span components into which it is divided. We represent this hierarchy as a span tree that preserves the complete source text and its original order. The root represents the whole prompt, intermediate nodes represent coherent sections, and leaves represent atomic guidance. The nodes of this tree provide the section- and atom-level realizations considered during feature discovery. Appendix E.1 presents a complete decomposition tree, and Appendix B provides the decomposition prompts.

Semantic resolution. Resolution determines which units across prompts realize the same feature. After decomposition, FACET groups semantically similar spans into candidate clusters using embedding similarity, comparing atoms with atoms and sections with sections. Because embedding similarity alone does not establish semantic equivalence, a language-model verifier examines each candidate cluster, removes inconsistent members, and confirms that the remaining spans express one coherent form of guidance. From each verified cluster, FACET constructs a feature record in the library.

Because the observed realizations retain their source provenance, resolution also associates each corpus prompt P with its feature composition F. We discuss the feature library recovered from our prompt corpora in Section 6.1, and how it is useful for understanding and bootstrapping prompts. We expect future work can improve the richness and quality of this discovery process. The resolution prompts are provided in Appendix C; Appendix E.2 illustrates how a cluster of observed spans yields a feature record.

3.3 Feature Editing

Feature editing modifies a prompt at feature level, through adding or removing a feature from the composition F and realizing the resulting composition as prompt text. This produces a prompt variant for feature effect measurement and prompt optimization.

For a target feature f, let F^{f-} denote the feature composition without f, and $F^{f+} = F^{f-} \cup \{f\}$ the corresponding composition with f. Editing relates their prompt realizations as follows:

$$P^{f-} \sim F^{f-} \underset{\text{remove}(f)}{\overset{\text{apply}(f)}{\rightleftharpoons}} F^{f+} \sim P^{f+}, \quad F^{f+} = F^{f-} \cup \{f\}. \tag{1}$$

The edit operation can instantiate a feature into a new prompt, remove unwanted guidance from a prompt, or produce the matched prompt pairs used for effect measurement.

Realizing an edit. Given a base prompt, edit direction, and target feature record, an *editing agent* produces the corresponding P^{f+} or P^{f-} while preserving all unrelated guidance and the prompt's task interface. Figure 4 illustrates the application of the feature *plan before solving* to a few-shot chain-of-thought prompt. The base prompt is adapted from the CoT reasoning prompt used in a math problem solving system XoT [38]. The base composition F^{f-} combines role playing, step-by-step reasoning, and few-shot demonstrations. Applying f retains these features and produces

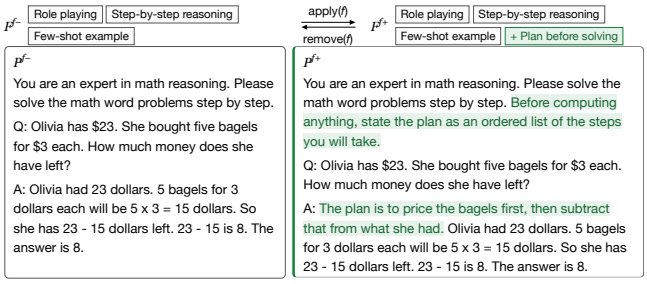


Figure 4: Applying the feature *plan before solving* to a few-shot chain-of-thought prompt.

F^{f+}. Appendix E.3 provides a separate example of applying a feature to text-to-SQL prompts.

To realize the edited composition as P^{f+}, Facet adds a general instruction requiring the model to state a plan before computing. Because the prompt also communicates its solving behavior through few-shot examples, the editing agent additionally applies the feature to each demonstrated solution with a problem-specific plan. In the example shown, the general instruction and the plan for solving the bagel problem jointly realize the same feature f. All original instructions remain unchanged.

Verifying an edit. A separate *verification agent* re-examines the edit against the feature specification and the intended direction, and accepts the edit only when three conditions hold. First, feature fidelity requires the agent to re-apply the recognition criterion to the edited prompt, judging the feature's presence in the edited prompt independently. Second, content preservation requires the agent to verify content outside the target remains unchanged, thereby preserving all other features of the prompt. Third, interface validity requires the prompt to remain well formed for the task, retaining its required placeholders, input fields, and output conventions, which Facet confirms by executing the edited prompt and requiring a parseable, runnable output.

4 Prompt Analytics

Prompt features provide an atomic unit for analyzing guidance across independently authored prompts. To study its use and effects across settings, we also model the contexts in which prompts are developed and evaluated, and the outcomes of their evaluations when available. Facet organizes prompt features, *context attributes*, and available *evaluation outcomes* into structured data to support this analysis.

4.1 Structured Representation

Figure 1 illustrates Facet's structured representation. Each record describes a prompt in a particular context through three groups of attributes: *prompt features*, *context attributes*, and *evaluation outcomes*. Each feature attribute is a binary indicator of whether the prompt contains the corresponding semantic guidance; context attributes describe its provenance and evaluation conditions; and outcome attributes record observed model behavior and task performance. A prompt used multiple times may have multiple records with different context or evaluation attributes; prompts without observed outcomes can still be analyzed using their features and context. Developers can upload their own prompts with custom context attributes and outcome measures, extending the representation to meet their local needs. In this way, this forms a wide and sparse matrix/table that is suitable for analysis.

Context Attributes. Context attributes are used to group and compare prompts and their evaluations. We distinguish between two types of context:

- *Metadata attributes* are associated with the prompt and describe its provenance and intended use, such as its source, publication year, target domain, and intended model. For example, a prompt may have `domain = Text-to-SQL`, `year = 2025`, and `intended_model = GPT-4`.
- *Evaluation attributes* describe the conditions of an execution, such as the benchmark, model configuration, and input characteristics (e.g., question type or difficulty). The same prompt may be evaluated under multiple contexts. An evaluation of the same prompt may have `benchmark = BIRD` and `model = Qwen2.5-Coder-7B`.

Evaluation Outcomes. Evaluation outcomes record what happens when a prompt is executed. We denote an execution by $(P, q, m) \rightarrow o$, where q is the task input, m is the model configuration, and o is the observed output. We consider two complementary measures:

- *Task performance.* A task evaluator $y(o, q) \in \mathbb{R}$ measures the quality of the output with respect to the task objective. For example, GSM8K [16] measures answer correctness in mathematical reasoning; BIRD [35] evaluates the execution accuracy of generated SQL queries; and KernelBench [42] evaluates generated GPU kernels through functional correctness and execution speedup over reference implementations.
- *Behavioral descriptor.* A behavioral descriptor $\phi(o, q) \in \mathbb{R}^d$ captures measurable properties of the output that give insights into how the model approaches the task, beyond the task performance. These may include general properties such as output length and format compliance, or task-specific properties such as the number of reasoning steps or the presence of unsupported assumptions in mathematical solutions.

Behavioral descriptors are useful in two settings. First, they distinguish outputs with identical task scores but different behaviors. For example, two SQL queries are incorrect, but one uses an incorrect function and the other returns unrequested columns. Descriptors that record function errors and the number of unrequested attributes can distinguish these cases.

Second, they provide inexpensive proxy measurements when direct task evaluation is costly. Such proxies are well established in AI to e.g., estimate text quality [46, 56], and human preferences [43, 57]. For example, when generating instruction-response pair for training, measuring their utility requires fine-tuning the target model, while proxies like response length [81] diversity [39], or complexity [14] help ident good examples that improve instruction-tuned models. Such descriptors can provide cheap, readily available proxy signals of downstream utility.

4.2 Common Prompt Analytics Workloads

Our structured representation of prompt collections and their evaluations enables systematic analysis of two core questions: *How is prompt guidance used?* and *What effects does prompt guidance have?*

Although prompt analytics is still emerging, we outline examples of analyses that can be conducted today.

Feature Usage Analysis. Analyzing feature occurrence across prompts reveals which guidance is commonly used. We select prompts by conditioning on context attributes or their combinations, then summarize feature prevalence within the selected collection. This allows us to examine how prompting practices vary across domains and sources and evolve over time.

Feature representations also enable retrieval based on shared guidance, going beyond embedding-based semantic similarity to identify prompts that express the same sets of features despite wording differences. For a prompt of interest, related prompts and prevalence statistics can suggest relevant features that the prompt lacks. In Section 8.1, we demonstrate that adding features prevalent in related prompts and contexts can improve the performance of published text-to-SQL pipelines.

Feature Effect Analysis. When evaluation outcomes are available, analysts can examine how feature composition relates to model behavior and task performance across evaluation contexts. Comparing outcomes across prompts can reveal associations, but these may reflect differences in other guidance or evaluation conditions rather than the feature of interest. Paired prompts that differ in one feature, evaluated under the same conditions, provide stronger evidence of that feature's effect.

Beyond estimating whether a feature is beneficial or harmful in a particular setting, we can investigate when its effects *transfer* across contexts. Conditioning effect estimates on attributes such as the model, benchmark, target language, database, or question structure reveals whether guidance that works in one setting remains effective in another. Behavioral descriptors provide additional evidence for interpreting these differences. For example, we can examine whether changes in output length accompany changes in task accuracy, or whether a feature is particularly effective on certain types of questions. As we show in Section 7.3, some features transfer even across disparate domains such as mathematical reasoning and text-to-SQL, while others exhibit context-dependent effects that do not generalize beyond that specific instance.

Effect estimates can inform the construction and revision of future prompts. For example, positive effect estimates can guide feature selection when initializing automatic prompt optimization (Section 8.2).

5 Building Evidence for Feature Effects

Feature effect analysis requires controlled comparisons to isolate the effects of individual prompt features. However, publicly collected prompts and their evaluation records rarely provide sufficient evidence, as prompts often differ in multiple features or are evaluated under different conditions. To bootstrap the evidence needed for these analyses, FACET supports controlled experiments on prompts from the corpus. For a target feature, FACET constructs prompt pairs that differ only in that feature and compares the outcomes when executing both variants under identical evaluation conditions.

The resulting *Contrast Records* form an initial collection of measurements for feature effect analysis. However, controlled experimentation is not the only way to populate this collection. For example, prompt optimization methods [4, 32] iteratively revise and evaluate prompts, producing execution traces from which feature-level comparisons may be recovered. Community-contributed evaluations could further expand the collection over time.

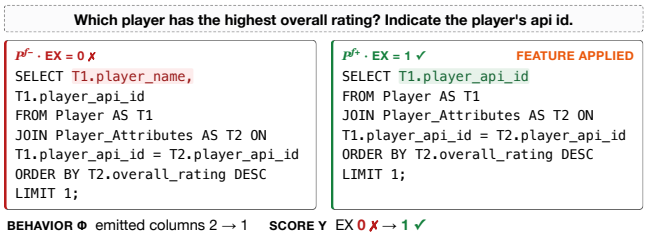


Figure 5: A contrast for the feature *add no logic beyond the request.* Applying the feature removes an unrequested output column, changing the behavioral descriptor (ϕ) from two emitted columns to one and the execution-accuracy score (y) from incorrect to correct.

Constructing Contrasts. To measure the effect of a target feature f, FACET constructs a matched prompt pair (P^{f-}, P^{f+}) that differs only in the presence of f. One variant is the original prompt, while the other is obtained by adding or removing f using the feature-editing procedure in Section 3.3.

Both variants are evaluated on the same task input q and model configuration m, producing outputs o^- and o^+, respectively. The paired executions form a *contrast* $c = (f, P, q, m)$, where P denotes the shared prompt background. Because the two runs hold the prompt background, input, and model configuration fixed, their difference isolates the effect of f in that setting.

For each contrast, FACET records the changes in behavioral descriptors ϕ and task performance y:

$$\Delta\phi(c) = \phi(o^+, q) - \phi(o^-, q), \quad \Delta y(c) = y(o^+, q) - y(o^-, q). \quad (2)$$

Figure 5 illustrates a contrast for the feature *add no logic beyond the request.* Without the feature, the model returns both the player's name and ID, although only the ID is requested; applying the feature removes the extra column. The behavioral descriptor ϕ captures this change in the number of emitted columns, while the change in score y captures the corresponding improvement in execution accuracy.

Estimating Feature Effects. To characterize effects beyond individual executions, FACET aggregates contrasts at two levels.

First, within a prompt background, we average the effects across task inputs. This measures how adding a feature changes performance for a particular prompt across multiple inputs. For a context predicate π over attributes such as the model, target language, or question type, the background-specific effect is:

$$e_y(f, P \mid \pi) = \underset{c=(f,P,q,m)\in C_{f,\pi}}{\text{mean}} \Delta y(c), \quad (3)$$

where $C_{f,\pi}$ denotes the contrasts for feature f satisfying π.

Second, *across prompt backgrounds*, we combine these estimates to characterize the feature's effect across independently authored prompts. This is important because a feature may complement guidance already present in one background or interact negatively with other features. We use the Hodges–Lehmann estimator [29] to obtain a robust estimate across prompt backgrounds. For prompt backgrounds $P_1, \ldots, P_n$, the feature's effect on task performance is

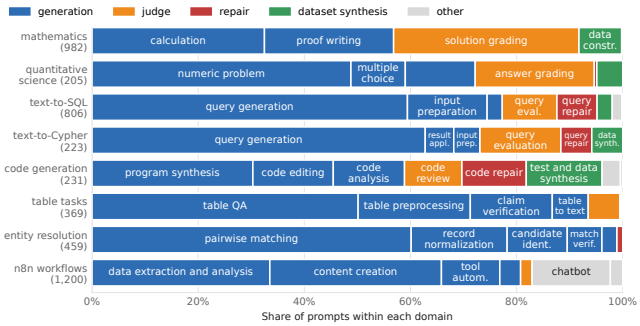


Figure 6: Task composition within each domain. Segment labels identify domain-specific tasks; colors indicate the common task categories.

estimated as

$$\widehat{\tau}_y(f \mid \pi) = \underset{1 \le i \le j \le n}{\text{median}} \left(\frac{e_y(f, P_i \mid \pi) + e_y(f, P_j \mid \pi)}{2} \right). \tag{4}$$

We report a two-sided 95% confidence interval obtained by inverting the Wilcoxon signed-rank test [70], and call an effect significant when its interval excludes zero. The behavioral effect $\widehat{\tau}_\phi(f \mid \pi)$ is estimated component-wise using the same procedure.

6 What is in the Prompts?

We use Facet to study prompt-feature reuse across eight domains: n8n workflows[1], text-to-SQL, text-to-Cypher, entity resolution, table tasks, mathematical problems, code generation, scientific quantitative problems. Our study is organized around two research questions:

- What reusable features recur across independently authored prompts? (§ 6.1)
- How does feature use vary across domains, tasks, and publication periods? (§ 6.2)

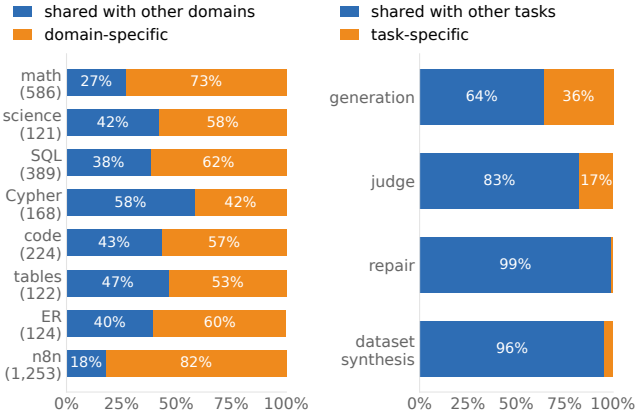
6.1 Prompt Corpus and Feature Library

We construct a corpus of real-world prompts and use Facet to identify their recurring features. The corpus provides both source material for feature discovery and diverse prompt backgrounds for subsequent effect measurements.

Prompt corpus. We collect prompts from publicly available n8n workflow templates and public sources covering the other seven domains. For the other seven research domains, we developed a reusable agent skill to collect prompts and their reported evaluation settings from public sources. Given a short description of a target domain, the skill recursively expands a typed graph of related benchmarks, leaderboards, evaluation harnesses, repositories, papers, and web pages, and extracts prompts directly from source files at fixed commits. We also record available provenance and evaluation metadata, such as the benchmarks and models used in the original evaluation, supported by verbatim evidence from the sources.

The resulting corpus contains 4,475 prompts from 1,587 sources. We record domain, task, intended model, and publication date as

[1]https://n8n.io/workflows/



(a) Feature sharing across domains. **(b) Feature sharing across tasks.**

Figure 7: Feature sharing across domains and tasks. Task-level proportions are averaged over domains.

context attributes. Domain describes the subject area, such as mathematics or text-to-SQL, while task describes the prompt's purpose, such as generating a query or judging an answer. The corpus covers both closed-source and open-weight models, with publication dates from 2021 to 2026; most prompts are from 2025. Facet also allows users to define additional context attributes for their analyses.

Figure 6 shows the distribution of prompts across tasks within each domain. Calculation, proof writing, and solution grading account for most math prompts, while query generation dominates text-to-SQL/Cypher. For the feature usage analysis in Section 6.2, we focus on four common task categories: generation, judging answer correctness, output repair, and dataset synthesis.

Feature library. We build a feature library for each domain using process described in Section 3.2. We then resolve semantically equivalent features across domains into global features.

Features recur across independently authored prompts. 2,805 occur in prompts from at least two systems authored independently, with a median of 6 prompts per feature. Examples of widely supported features include "follow the specified output format" (21.1%), "act as an expert" (17.5%), "analyze the provided input" (13.2%) and "be concise" (11.6%). They capture common prompting practices for assigning roles, analyzing inputs and specifying the output formats.

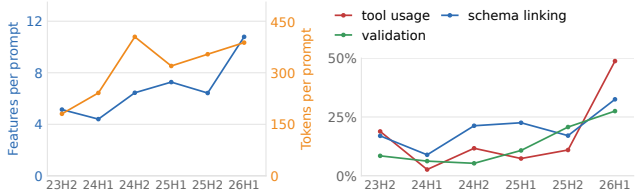
6.2 Feature Use Across Contexts

We examine how these features are used across contexts by grouping prompts by domain, task, and publication period. These analyses can be expressed through simple SQL queries over the corpus representation. For example, using `Prompt(id, domain, task, year)` and `Has(prompt, feature)`, the following query counts the domains in which each feature occurs:

```
SELECT h.feature,
COUNT(DISTINCT p.domain) AS n_domains
FROM Has h JOIN Prompt p ON p.id = h.prompt
GROUP BY h.feature;
```

Features with `n_domains = 1` are domain-specific.

These analyses characterize shared and context-specific features as well as changes in feature use over time.



(a) Features and tokens per prompt. (b) Text-to-SQL feature prevalence.

Figure 8: Feature usage over time. H1 and H2 denote the first and second halves of a year.

Domain. Prompt features capture broadly shared conventions and domain-specific requirements. Of the 2,987 features, 30% are shared across domains. Figure 7a shows that this proportion ranges from 18% in n8n workflow prompts to 58% in text-to-Cypher prompts.

Four features occur in all eight domains: "follow the specified output format", "reason step by step", "omit explanations", "act as a helpful assistant". More specific features also recur across domains. For example, mathematics and science prompts share requirements to box the final answer and format mathematics in LaTeX. These instructions specify how mathematical answers should be presented. Text-to-SQL and text-to-Cypher share features concerning query validity and schema use, including "avoid undefined schema elements" and "validating queries through execution".

The remaining 70% of features occur in only one domain in our corpus. These include requirements tied to domain knowledge and workflow constraints. For example, *use only well-known theorems* appears in 40 math prompts, restricting the mathematical knowledge used in a solution. In n8n workflow prompts, *search the knowledge base before answering* appears in 85 prompts, requiring consultation of internal documentation.

Task. Features are shared widely across tasks, but the extent of sharing differs substantially by task (Figure 7b).

Generation has the largest average proportion of task-specific features at 36%. For example, *open with a hook* asks the model to begin generated content with an attention-grabbing opening. It appears in 27 n8n workflow generation prompts, but in none of the 231 prompts for other n8n tasks.

Repair has the highest average proportion of shared features at 99%. For example, *analyze the code carefully* appears in 32% code repair and 16% code generation prompts, reflecting a shared requirement to examine code when producing or correcting it.

Time. Prompting practices evolve in both the number of features expressed and the ways prompts direct the model to perform a task. Later prompts contain on average more features (Figure 8a): from 5.1 in 2023 to 10.8 in early 2026; and mean prompt length also increases from 181 to 389 tokens.

The guidance in prompts also changes over time. Using text-to-SQL from the second half of 2023 as an example (Figure 8b), more recent prompts place greater emphasis on database exploration and tool interaction. The share of prompts containing tool-use feature rises from 11% in second half of year 2025 to 49% in first half of year 2026. Schema-linking features features (e.g., to check generated queries before running them) also become more prevalent overall. In addition, prompts increasingly include query-validation features that instruct the model to check generated queries before answering.

7 Are Features Useful?

In this section, we use Facet's active experimentation capability (Section 5) to conduct a focused study on effect on selected prompt features. By comparing matched prompts that differ only in the presence of a target feature, we measure its effects on model behavior and task performance across different contexts. We focus on two domains, Text2DSL generation and mathematical problem solving, and address two questions:

- **Feature effects:** How do prompt features affect model behavior and task performance? (§ 7.2)
- **Transferability:** When do feature effects generalize across domains, tasks, and models? (§ 7.3)

7.1 Active Experimentation Setup

Tasks and Models. We evaluate feature effects on six benchmarks spanning two domains. For Text2DSL, we use BIRD [35] (BD) and Spider [75] (SP) for SQL, and CypherBench [21] (CB) and Mind-the-Query [12] (MQ) for Cypher. We evaluate query execution accuracy using Qwen2.5-Coder-7B and Qwen2.5-Coder-32B [53].

For mathematical problem solving, we use GSM8K [15] (G8) and GSM-Hard [24] (GH), two benchmarks of grade-school mathematics problems. We use Qwen2.5-7B-Instruct [53] and Gemma-3-12B-it [62] for these experiments and evaluate answer accuracy.

Feature Taxonomy and Selection. To organize the analysis, we classify features into three categories based on the aspect of generation they guide.

- *Task specification* defines what the model should do and the required output interface (*e.g.*, generate a SQL query or adopt an expert role).
- *Reasoning strategy* guides how the model approaches a problem (*e.g.*, decompose a problem into steps or check relevant schema elements before generation).
- *Solution criteria* specifies properties the final answer should satisfy (*e.g.*, use only provided schema elements or handle NULL values correctly).

The library contains 586 mathematics and 515 Text2DSL features, distributed roughly evenly across the three categories. For controlled evaluation, we select 50 Text2DSL and 29 math features. We prioritize frequently recurring features while maintaining coverage across the categories. The selected features have a median support of 12 source prompts, so we can evaluate the same feature across independently authored prompt backgrounds.

Most task-specification features cannot be added or removed without changing the task interface (*e.g.*, removing *transform the question into a SQL query* changes the expected output for the task). We therefore evaluate only three task-specification features that can be varied while preserving the interface: *one-shot example of language*, *adopt an expert role*, and *use correct language syntax*. We otherwise focus on reasoning strategies and solution criteria.

Matched Constrasts. We use single-call prompts as experimental backgrounds and retain only feature interventions that pass the verification conditions in Section 3.3. As defined in Section 5, each

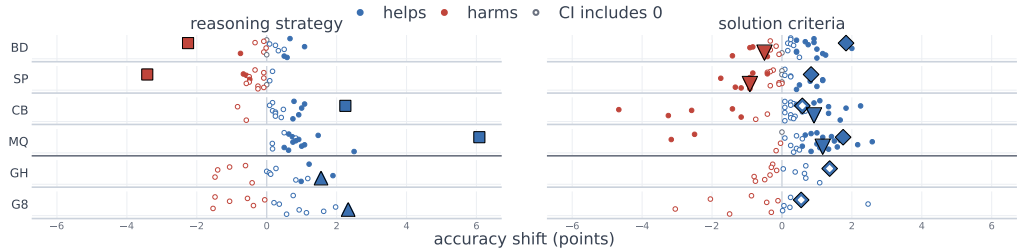
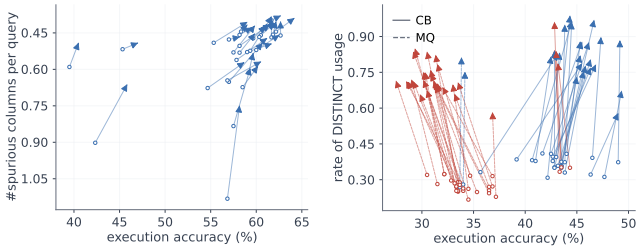


Figure 9: Aggregated feature effects ($\widehat{\tau}_y(f \mid \pi \equiv 1)$) across six benchmarks. Each point shows the effect of one feature; blue and red indicate positive and negative effects, respectively, and hollow markers indicate that CI_{95} includes zero. Selected features are highlighted across benchmarks: ■ *decompose into subqueries and combine intermediate results*; ▲ *note every given quantity before solving*; ♦ *add no logic beyond the request / solve under the stated conditions*; and ▼ *convert nonstandard field encodings before use.*



(a) Add no logic beyond the request (BD·32B) **(b) Use distinct (CB·32B + MQ · 32B)**

Figure 10: Representative feature effects on model behavior and task performance. Each panel relates a feature's behavioral effect (y-axis) to its performance effect (x-axis).

record compares P^{f-} and P^{f+} under the same input q and model configuration m, varying only the presence of f.

For Text2DSL, each feature is evaluated on an average of 26 verified base prompts and 600 questions sampled from the benchmark, stratified by database. For mathematics, each feature is evaluated across average 10 base prompts on the complete GSM8K and GSM-Hard test sets, each containing 1319 questions. In total, we collect *6.5M matched contrasts* spanning features, prompt backgrounds, questions, and models.

7.2 What Effects Do Prompt Features Have?

Overall effects. Figure 9 reports the aggregated performance effects ($\widehat{\tau}_y(f \mid \pi \equiv 1)$) of reasoning strategies and solution criteria across six benchmarks. Overall, feature effects are widespread and heterogeneous: many features have statistically significant effects, in both positive and negative directions.

The two feature categories are different. Reasoning features differ across target languages: their effects are neutral or negative on SQL, but positive on Cypher, which we study in Section 7.3.2. Reasoning features tend to improve in mathematics. In comparison, solution criteria have stronger effects on Text2DSL than on mathematics (whose effects are near zero). This difference is consistent with the output requirements of the two tasks: math questions require a numerical answer, whereas generated queries must be syntactically and semantically correct *and* return the intended result.

The aggregate results establish that prompt features affect task performance, but do not explain how. We next examine how behavioral measurements help explain feature effects, and why similar behavioral changes can lead to different performance outcomes.

Behavioral changes help explain feature effects. Feature-induced changes in model behavior are associated with changes in task performance, providing evidence of how prompt guidance affects accuracy. For example, the feature *add no logic beyond the request* instructs the model to avoid introducing restrictions unsupported by the question. On BIRD (32B), adding the feature reduces unsupported query structure, removing 0.08 spurious selected columns per query, while improving execution accuracy by 1.8 points. Figure 10a relates these behavioral and performance effects at the base-prompt level: prompts with reductions in spurious selected columns ($e_\phi(f)$) tend to have accuracy gains ($e_y(f)$), producing the predominantly downward-right arrows in the figure. The same relationship extends beyond this feature. Across the other BIRD features, larger reductions in column over-selection are associated with accuracy gains ($r = -0.76$ at 32B and -0.65 at 7B). Thus, reducing unsupported query structure is a behavior that improves execution accuracy.

Intended behavioral changes do not always improve performance. A feature can reliably induce its intended behavior without improving task performance. Consider the feature *deduplicate results when required by the question*, which instructs the model to remove duplicate rows when the question asks for unique values. Applying this feature increases the share of queries using `DISTINCT` by 0.48 on CypherBench and 0.45 on Mind-the-Query, with no change in query runnability. Despite causing a similar behavioral change, its effect on accuracy reverses: execution accuracy increases by 1.8 points on CypherBench but decreases by 3.2 points on MtQ (Figure 10b). This difference reflects the benchmarks' evaluation distributions: 72% of CypherBench gold queries use deduplication, compared with only 4% on Mind-the-Query. On MtQ, the feature is applied even when the gold answer preserves duplicates: for 84% of the questions that fall outside the feature's intended condition, the feature reduces accuracy by 4.6 points, while questions satisfying the condition gain 0.6 points. Thus, the feature reliably guides the intended model behavior, but whether that behavior improves accuracy depends on how well its implicit condition aligns with the evaluation setting. These findings motivate examining not only

Table 1: Feature effects across task domains. Each row shows domain-specific realizations of the same feature and its aggregated effects on mathematics and Text2DSL benchmarks. Shading indicates significant effects (CI_{95} excludes zero).

Realization		Math		Text2DSL			
Math	Text2DSL	GH	G8	BD	SP	CB	MQ
Effects transfer across domains							
solve under the stated conditions	add no logic beyond the request	+1.5	+0.7	+1.8	+0.8	+0.6	+1.8
satisfy every stated condition	follow every stated definition	+0.6	+2.7	+0.3	+0.1	+1.1	+0.0
check the answer against the question	check & adjust query against question and schema	+1.1	+1.2	+0.3	−0.5	+0.2	+1.1
Effects depend on domain							
note every given quantity before solving	identify every needed schema element then verify	+1.7	+2.4	−0.1	−0.3	−1.2	+0.4
include only what correctness needs	include only what was requested	−0.0	−0.1	+1.3	+1.0	+0.3	+1.3

whether a feature changes model behavior, but also the contexts in which those changes improve task performance.

7.3 When Do Feature Effects Transfer?

We examine whether feature effects transfer across execution contexts by comparing their estimated effects conditioned on context attributes. We focus on task domains, target languages, and models. Feature effects generally transfer across domains, languages, and models, but are sensitive to task structure and model capabilities.

7.3.1 Across Task Domains. Despite substantial differences between math and Text2DSL domains, 3 features consistently improve both (Table 1). Cross-domain transfer relies on adapting a feature's realization to the target domain rather than reusing the same instruction verbatim. For example, Example 3.1 shows how the feature *remain faithful to the stated task* appears as *solve under the stated conditions* in mathematics and *add no logic beyond the request* in Text2DSL. Both discourage introducing information beyond the task and this feature has positive effects in both domains.

More commonly, however, feature effects are domain-dependent. For example, guidance to inspect the available information before solving is beneficial in mathematics, where explicitly identifying given quantities and conditions improves accuracy, but not for Text2DSL when instantiated as identifying and verifying required schema elements. Conversely, restricting the output to only requested information has little effect in mathematics but consistently improves query-generation. Thus, domain-specific realization makes cross-domain reuse possible, but whether the resulting feature is beneficial is still an empirical question.

7.3.2 Across Target Languages. Feature effects transfer more frequently between SQL and Cypher within the same query-generation task than across task domains. Table 2 summarizes features with significant effects in both languages: eleven have positive effects in both SQL and Cypher, while seven change direction between the two languages.

Most transferable features are solution criteria. Broad requirements that constrain the generated query to the user request or provided schema, such as *avoiding unrequested logic*, *using only provided schema elements*, and *returning only requested information*, generally improve accuracy in both SQL and Cypher. In contrast, more specific requirements concerning field encodings and type handling can help Cypher but hurt SQL.

Table 2: Feature effects ($\widehat{\tau}_y(f \mid \pi_{\text{language}})$) across target languages on the 32B model. Shading indicates significant effects.

	SQL		Cypher	
Feature	BD	SP	CB	MQ
Effects transfer across languages				
Solution criteria				
add no logic beyond the request	+1.8	+0.8	+0.6	+1.8
avoid unsupported claims	−0.3	+0.8	+0.1	+1.0
include only necessary data source from schema	+1.0	−0.3	+0.1	+1.0
include only what was requested	+1.3	+1.0	+0.3	+1.3
reference columns from the correct table	+0.4	−0.0	+0.3	+0.6
reference only schema elements provided	+1.2	+0.5	+0.4	+1.1
return only explicitly requested information	+2.0	+1.2	+0.7	+1.1
Reasoning strategies				
avoid unsupported reasoning shortcuts	+1.1	−0.1	+0.5	+0.8
reason silently before answering	+0.5	−0.1	+0.7	+0.6
require logically sound and complete reasoning	+0.7	−0.3	+0.8	+0.5
simulate the query before generating	+0.6	−0.3	−0.6	+2.5
Effects reverse across languages				
Solution criteria				
convert nonstandard field encodings before use	−0.5	−0.9	+0.9	+1.2
match operations and literals to field types	+0.5	−0.9	+1.7	+0.8
Reasoning strategies				
decompose into subqueries and combine intermediate results	−2.2	−3.4	+2.3	+6.1
optimize for performance	−0.0	−0.5	+0.2	+0.8
require step-by-step reasoning	−0.7	+0.0	+0.3	+0.5
verify correctness and optimize	−0.3	−0.7	+0.1	+0.7
Task specification				
one-shot example of language	+0.2	−0.5	+2.2	+3.9

Reasoning strategies also differ substantially between the two languages. As observed in Figure 9, they are generally more beneficial for Cypher. Across features, the correlation between the effects on response length ($\widehat{\tau}_\phi$) and accuracy ($\widehat{\tau}_y$) is negative for SQL ($r = -0.52$) and positive for Cypher ($r = +0.58$). While descriptive rather than causal, this suggests that additional generated structure often introduces unnecessary intermediate computation in SQL, whereas in Cypher it more often reflects useful decomposition of queries involving multiple relationship patterns.

Most notably, *decompose into subqueries and combine intermediate results* decreases accuracy by 2.2–3.4 points on the SQL benchmarks but increases it by 2.3–6.1 points on the Cypher benchmarks. It adds 1,000–2,000 characters of intermediate output on all four benchmarks. In Cypher, the improvement is concentrated on queries involving multiple relationship patterns (+5.4 points versus +0.7 on other queries). This concentration supports the interpretation that explicit decomposition is useful when the target query requires coordinating multiple graph structures.

The feature *reason silently before answering* asks the model to reason without exposing its intermediate process. Although it adds almost no output on Spider and the Cypher benchmarks and only 147 characters on BIRD, it rewrites 27–47% of queries and improves three of the four benchmarks by 0.5–0.7 points. Corrective changes tend to simplify BIRD queries by removing excess joins, aliases, and columns, while adding useful pattern structure in Cypher.

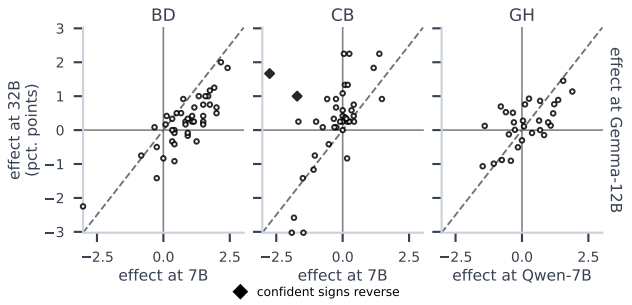


Figure 11: Feature effects across models. The dashed line indicates equal effects; diamonds mark features with statistically significant effects in opposite directions.

7.3.3 Across Models. We next examine whether feature effects transfer across models. We compare Qwen2.5-Coder-7B and 32B for query generation, and compare Qwen2.5-7B-Instruct and Gemma-3-12B-it for mathematics. Figure 11 illustrates representative feature effects across models. Effect profiles are correlated across scale (r = +0.81 on BIRD, +0.53 on CypherBench) and across families (+0.63 on GSM-Hard).

Model dependence can arise from both scale and model specialization. For example, the *numeric casting* feature (cast numeric values to an appropriate numeric type before using them in arithmetic or comparisons) changes from −2.75 points on Qwen2.5-Coder-7B to +0.83 at 14B and +1.58 at 32B, showing that the same guidance can become beneficial as model scale increases. This difference is not simply due to instruction following: the 7B model also inserts casts, but sometimes places them incorrectly, causing otherwise valid queries to fail. Model specialization also matters. For example, for both *select clause* and *ordered construction*, Qwen2.5-14B-Instruct has positive effects while Qwen2.5-Coder-14B has negative effects.

8 Use Cases and Applications

We show how Facet helps improve existing prompts by recommending features from related prompts, and warm-starting prompt optimization with features with positive effects.

8.1 Recommend Features

We augment prompts in five published Text-to-SQL pipelines by adding features commonly used in related prompts and contexts: ROUTE (ICLR'25) [50], MAC-SQL (COLING'25) [66], DAIL-SQL (VLDB'24) [23], TA-SQL (ACL'24) [51] and DIN-SQL (NeurIPS'23) [48].

We identify candidate features from two subsets of the prompt corpus: (1) prompts matching the target's domain and subtask (*e.g.*, schema linking, query generation, or repair), and (2) the ten prompts sharing the most features with the target. The first captures common guidance for the same task, while the second identifies guidance used alongside features already present in the target prompt. We rank features by prevalence within each subset, retain up to 15 candidates per subset, and exclude features already present, requiring unavailable inputs, or conflicting with the output interface. In total, we add 26 features across 20 prompts. The most frequently added features are *use only schema identifiers*, *understand the question and schema*, and *select only requested columns*.

Table 3: Execution accuracy across Text-to-SQL systems and benchmarks. Published reports the original results, Base our reproduced baseline, and Edit the results after feature-based prompt editing. Δ represents the improvement over Base in percentage points. Base, Edit and Δ are averaged over five runs.

System	Bench.	Model	Published	Base	Edit	Δ
ROUTE	Spider	Qwen2.5-7B	78.3	78.3	80.9	+2.57
		Qwen2.5-14B	80.0	80.9	81.0	+0.06
	BIRD	Qwen2.5-7B	49.7	52.2	54.6	+2.39
		Qwen2.5-14B	56.3	57.7	58.1	+0.47
MAC-SQL	Spider	Qwen2.5-7B	71.7	68.9	73.5	+4.60
		SQL-Llama-7B	76.25	65.4	65.6	+0.15
	BIRD	Qwen2.5-7B	46.7	46.9	47.1	+0.25
		SQL-Llama-7B	43.94	44.9	44.3	−0.55
DAIL-SQL	Spider	CodeLlama-34B	71.4	73.1	74.8	+1.67
TA-SQL	BIRD	DeepSeek-Coder-33B	43.74	43.0	43.7	+0.67
DIN-SQL	Spider	Qwen2.5-7B	72.1	70.7	71.5	+0.81

Table 3 compares the original and augmented pipelines under the same model configurations, averaged over five runs. Overall, mean execution accuracy increases in ten of the eleven configurations. The largest gains occur with Qwen2.5-7B. ROUTE improves by 2.57 percentage points on Spider and 2.39 on BIRD, while MAC-SQL improves by 4.60 on Spider. DAIL-SQL with CodeLlama-34B improves by 1.67 points. Within ROUTE, the gains are smaller with Qwen2.5-14B, at 0.06 points on Spider and 0.47 on BIRD. MAC-SQL with SQL-Llama shows little change, with a gain of 0.15 points on Spider and a decrease of 0.55 on BIRD. These gains are notable because the original prompts come from published, carefully engineered pipelines, while our recommendations rely only on feature prevalence and similarity among existing prompts. This suggests that even well-developed prompting pipelines can benefit from reusing guidance discovered across independently designed systems.

The edits also change model behaviors. On Spider with Qwen2.5-7B, the mean number of queries selecting more columns than the reference query decreases from 30.6 to 18.8 per run for ROUTE and from 99.8 to 56.0 for MAC-SQL.

8.2 Warm-Starting Prompt Optimization

Measured feature effects can provide prior evidence for initializing automatic prompt optimization. We test whether starting from features with positive measured effects improves the prompts discovered by automatic prompt optimizer GEPA [4].

We select ten Cypher features from Section 7 that improve execution accuracy by at least one point on at least one Cypher benchmark. They include one task-specification feature providing a one-shot example, two reasoning features that prescribe structured derivation before query generation, and seven solution-criteria features that constrain query construction. We compare a *base* prompt that requests a Cypher query from a schema and question with a *composed* prompt augmented with these ten features.

We optimize both initial prompts using GEPA [4] under the DSPy program abstraction [32]. At each iteration, GEPA uses execution traces from sampled optimization questions to propose revised

Table 4: Held-out execution accuracy (%) before and after GEPA optimization. Numbers in parentheses report the improvement from feature-informed initialization.

database	art	biology	soccer	terrorist
held-out questions	2672	1992	1528	1142
base	42.8	45.1	60.0	49.9
composed	46.7 (+3.9)	51.3 (+6.2)	67.0 (+7.0)	56.0 (+6.1)
base + GEPA	61.9	52.8	68.3	72.0
composed + GEPA	62.6 (+0.7)	63.2 (+10.3)	77.0 (+8.7)	76.7 (+4.7)

instructions and maintains a Pareto frontier evaluated on the validation split. We use DeepSeek-V4-Flash-0731 [18] for reflection and Qwen2.5-Coder-32B for query generation. The two runs use identical models at temperature zero, data splits, and optimization budgets and differ only in their initial prompt. We evaluate on four CypherBench graphs: art, biology, soccer, and terrorist. For each graph, we use 150 questions for optimization, 150 for validation, and all remaining questions for held-out testing. Neither effect based feature selection nor prompt optimization uses the held-out test questions.

Overall, feature-informed initialization improves held-out execution accuracy by 3.9–7.0 points across the four graphs before optimization (Table 4). After optimization, prompts initialized with promising features remain better on all four graphs, with gains of 0.7–10.3 points and 6.1 points on average. These results provide preliminary evidence that measured feature effects can provide a useful prior for automatic prompt optimization.

9 Related Work

Prompt Management. Prompts are increasingly recognized as artifacts to be collected, organized, compared, and reused. Prior work has developed interactive tools for browsing and comparing prompt variants across models [5, 7], as well as for collaborative prompt editing in a shared workspace [54]. In the data management community, PromptEditor [8] introduces structure to support prompt editing by learning a corpus-derived taxonomy of prompt segments. SPEAR [11] represents prompts in LLM pipelines as first-class, versioned data objects with an algebra for composition and runtime refinement. Facet complements these systems with the *prompt feature* abstraction, representing reusable semantic guidance that recurs across prompts as explicit units that can be identified, compared, and measured.

Prompt Sensitivity and Evaluation. Prior work examine how prompt variation affects model evaluation. DOVE systematically varies prompt formatting, instruction phrasing, and demonstrations to characterize model sensitivity [28]. PromptEval estimates performance distribution across prompt templates under limited evaluation budgets [47]. Facet complements these approaches by identifying recurring semantic guidance in prompt collections and studying its usage and effects across prompt backgrounds and evaluation contexts.

Prompt Optimization. A complementary line of work automatically improves prompts for a target task. Frameworks generate and select instructions [83], iteratively refine instructions and demonstrations from task feedback [3, 49, 76], and optimize prompts in modular language-model programs [4, 32, 41, 82]. Other work targets particular aspects of the optimization process. CAPO reduces optimization cost through joint performance–length optimization [78], while Prompt Dual Optimizer replaces labeled references with pairwise LLM judgements [72].

Several methods also operate at finer granularity than the complete prompt. HAPO attributes task errors to functional prompt segments and edits high-attribution semantic units [13]; Optimas assigns locally aligned rewards to components of a compound AI system, including prompt-based modules [71]. Overall, these methods use feedback from a target setting to optimize a prompt or system for that setting. Facet instead measures recurring features across prompts and contexts, producing feature-level evidence that can subsequently inform optimization.

Prompt Transfer. Prompting surveys have cataloged recurring techniques that apply across tasks [55, 69]. Prior work has also studied whether learned prompts transfer across tasks and models, either by using source prompts to initialize a target task [59, 65] or by learning from target-task data how to mix frozen source-task prompts with a newly initialized target prompt [6]. More recent work studies the transferability of complete optimizer-generated prompts [17], as well as the effectiveness of reusable agent skills across tasks and environments [30]. Facet instead studies transfer at the feature level, measuring whether individual units of prompt guidance remain effective across different contexts.

Prompt Corpus. Publicly available prompts provide a growing record of how practitioners and researchers instruct LLMs across domains. Several public repositories curate system prompts from deployed AI applications [1, 2, 37]. One source in our corpus is Text2DSL, a well-studied domain in the data management community. LLM-based text-to-SQL systems developed diverse prompt strategies, from fixed multi-stage pipelines that decompose schema linking, demonstration selection, generation, and correction [20, 22, 31, 36, 40, 48, 74, 79, 80] to adaptive workflows that incorporate candidate search, execution feedback, and agent collaboration [10, 19, 25, 33, 34, 52, 61, 67, 68, 73]. Text-to-Cypher follows a similar but more recent progression [9, 26, 27, 44, 45, 58, 63, 64]. We collect prompts released by these systems as part of our corpus. Together, these efforts provide a growing body of prompt artifacts from which recurring guidance can be identified and studied.

10 Conclusion

We introduced *prompt analytics*, the systematic analysis of prompt collections to understand the guidance they contain, the contexts in which it is used, and its effects on model behavior and task performance. We developed Facet to support analytical workloads ranging from characterizing feature usage and co-occurrence to measuring feature effects, identifying interactions, and studying transfer across contexts. Using publicly available sources, we constructed a large prompt corpus and feature library, showing how standard relational operations can yield useful insights once the underlying structure is made explicit. We envision prompt analytics as a foundation for accumulating and reusing empirical knowledge about prompting. We will maintain the Facet feature library as an open-source resource for community use and contribution.

References

[1] [n. d.]. The Big Prompt Library. https://github.com/0xeb/TheBigPromptLibrary/ Accessed: Sep, 2026.

[2] [n. d.]. Crafting Effective Prompts for Agentic AI Systems: Patterns and Practices. https://github.com/dontriskit/awesome-ai-system-prompts Accessed: Sep, 2026.

[3] Eshaan Agarwal, Joykirat Singh, Vivek Dani, Raghav Magazine, Tanuja Ganu, and Akshay Nambi. 2024. PromptWizard: Task-Aware Prompt Optimization Framework. doi:10.48550/arXiv.2405.18369 arXiv:2405.18369 [cs].

[4] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457* (2025).

[5] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L. Glassman. 2024. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 304, 18 pages. doi:10.1145/3613904.3642016

[6] Akari Asai, Mohammadreza Salehi, Matthew Peters, and Hannaneh Hajishirzi. 2022. ATTEMPT: Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 6655–6672. doi:10.18653/v1/2022.emnlp-main.446

[7] Stephen H Bach, Victor Sanh, Zheng Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, et al. 2022. Promptsource: An integrated development environment and repository for natural language prompts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 93–104.

[8] Jeffery Cao, Lampros Flokas, Yujian Xu, Eugene Wu, Xu Chu, and Cong Yu. 2025. Prompt Editor: A Taxonomy-driven System for Guided LLM Prompt Development in Enterprise Settings. In *Companion of the 2025 International Conference on Management of Data*. 59–62.

[9] Francesco Cazzaro, Jessica Lennon, and Ariadna Quattoni. 2026. Achieving Precise Text-To-Cypher Via Grounded Knowledge Graph Data Generation. arXiv:2606.14325 [cs.CL] https://arxiv.org/abs/2606.14325

[10] Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. 2025. SQLFixAgent: towards semantic-accurate text-to-SQL parsing via consistency-enhanced multi-agent collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 49–57.

[11] Ugur Cetintemel, Shu Chen, Alexander W Lee, Deepti Raghavan, Duo Lu, and Andrew Crotty. 2025. Making prompts first-class citizens for adaptive llm pipelines. *arXiv preprint arXiv:2508.05012* (2025).

[12] Vashu Chauhan, Shobhit Raj, Shashank Mujumdar, Avirup Saha, and Anannay Jain. 2025. Mind the query: A benchmark dataset towards Text2Cypher task. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*. 1890–1905.

[13] Dongyu Chen, Jian Ma, Xianpeng Zhang, Lei Zhang, Haonan Lu, Chen Chen, Chuangchuang Wang, and Kai Tang. 2026. Learning from Prompt itself: the Hierarchical Attribution Prompt Optimization. arXiv:2601.02683 [cs.AI] https://arxiv.org/abs/2601.02683

[14] Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024. AlpaGasus: Training a Better Alpaca with Fewer Data. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=FdVXgSJhvz

[15] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).

[16] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

[17] Sarkar Snigdha Sarathi Das, Ryo Kamoi, Bo Pang, Yusen Zhang, Caiming Xiong, and Rui Zhang. 2025. Greater: Gradients over reasoning makes smaller language models strong prompt optimizers. In *International Conference on Learning Representations*, Vol. 2025. 8765–8796.

[18] DeepSeek-AI. 2026. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence.

[19] Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Consensus Enforcement, and Column Exploration. arXiv:2502.00675 [cs.CL] https://arxiv.org/abs/2502.00675

[20] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306 [cs.CL] https://arxiv.org/abs/2307.07306

[21] Yanlin Feng, Simone Papicchio, and Sajjadur Rahman. 2024. CypherBench: Towards Precise Retrieval over Full-scale Modern Knowledge Graphs in the LLM Era. *arXiv preprint arXiv:2412.18702* (2024).

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

[23] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (Jan. 2024), 1132–1145. doi:10.14778/3641204.3641221

[24] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International conference on machine learning*. PMLR, 10764–10799.

[25] Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. 2025. MSc-SQL: Multi-Sample Critiquing Small Language Models For Text-To-SQL Translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Luis Chiruzzo, Alan Ritter, and Lu Wang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 2145–2160. doi:10.18653/v1/2025.naacl-long.107

[26] Aibo Guo, Xinyi Li, Guanchen Xiao, Zhen Tan, and Xiang Zhao. 2022. SpCQL: A Semantic Parsing Dataset for Converting Natural Language into Cypher. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (Atlanta, GA, USA) *(CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 3973–3977. doi:10.1145/3511808.3557703

[27] Anton Gusarov, Anastasia Volkova, Valentin Khrulkov, Andrey Kuznetsov, Evgenii Maslov, and Ivan Oseledets. 2025. Multi-Agent GraphRAG: A Text-to-Cypher Framework for Labeled Property Graphs. arXiv:2511.08274 [cs.AI] https://arxiv.org/abs/2511.08274

[28] Eliya Habba, Ofir Arviv, Itay Itzhak, Yotam Perlitz, Elron Bandel, Leshem Choshen, Michal Shmueli-Scheuer, and Gabriel Stanovsky. 2025. Dove: A large-scale multi-dimensional predictions dataset towards meaningful llm evaluation. *arXiv preprint arXiv:2503.01622* (2025).

[29] Joseph L Hodges Jr and Erich L Lehmann. 2011. Estimates of location based on rank tests. In *Selected works of EL Lehmann*. Springer, 287–300.

[30] Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, et al. 2026. From raw experience to skill consumption: A systematic study of model-generated agent skills. *arXiv preprint arXiv:2605.23899* (2026).

[31] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. arXiv:2305.09645 [cs.CL] https://arxiv.org/abs/2305.09645

[32] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).

[33] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-SQL: Zero-Shot Text-to-SQL using Monte Carlo Tree Search. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*, Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (Eds.). PMLR, 36810–36830. https://proceedings.mlr.press/v267/li25dt.html

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

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

[36] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency. arXiv:2403.09732 [cs.CL] https://arxiv.org/abs/2403.09732

[37] Xiangning Lin, Shenzhe Zhu, Shu Yang, Zhenyu Zhang, Haoqian Zhang, Yipeng Zhao, Chengxuan Qian, Tianwei Wang, Ziheng Zhang, Zhenlong Yuan, et al. 2026. AISPA: User-Centric System Prompt Auditing for Large Language Model Applications. *arXiv preprint arXiv:2607.28617* (2026).

[38] Tengxiao Liu, Qipeng Guo, Yuqing Yang, Xiangkun Hu, Yue Zhang, Xipeng Qiu, and Zheng Zhang. 2023. Plan, verify and switch: Integrated reasoning with diverse x-of-thoughts. In *Proceedings of the 2023 conference on empirical methods in natural language processing*. 2807–2822.

[39] Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024. What Makes Good Data for Alignment? A Comprehensive Study of Automatic Data Selection in Instruction Tuning. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=BTKAeLqLMw

[40] Ruilin Luo, Liyuan Wang, Binghuai Lin, Zicheng Lin, and Yujiu Yang. 2024. PTD-SQL: Partitioning and Targeted Drilling with LLMs in Text-to-SQL. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 3767–3799. doi:10.18653/v1/2024.emnlp-main.221

[41] Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 9340–9366.

[42] Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. 2025. KernelBench: can LLMs write efficient GPU kernels?. In *Proceedings of the 42nd International Conference on Machine Learning* (Vancouver, Canada) *(ICML'25)*. JMLR.org, Article 1894, 60 pages.

[43] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems*, Vol. 35. https://arxiv.org/abs/2203.02155

[44] Makbule Gulcin Ozsoy. 2025. Enhancing text2cypher with schema filtering. *arXiv preprint arXiv:2505.05118* (2025).

[45] Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2Cypher: Bridging Natural Language and Graph Databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, Genet Asefa Gesese, Harald Sack, Heiko Paulheim, Albert Merono-Penuela, and Lihu Chen (Eds.). International Committee on Computational Linguistics, Abu Dhabi, UAE, 100–108. https://aclanthology.org/2025.genaik-1.11/

[46] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. 311–318. doi:10.3115/1073083.1073135

[47] Felipe Maia Polo, Ronald Xu, Lucas Weber, Mírian Silva, Onkar Bhardwaj, Leshem Choshen, Allysson Flavio Melo de Oliveira, Yuekai Sun, and Mikhail Yurochkin. 2024. Efficient multi-prompt evaluation of LLMs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 707, 30 pages.

[48] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 36339–36348. doi:10.52202/075280-1577

[49] Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*. 7957–7968.

[50] Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2025. ROUTE: Robust multitask tuning and collaboration for Text-to-SQL. In *International Conference on Learning Representations*, Vol. 2025. 12425–12448.

[51] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 5456–5471. doi:10.18653/v1/2024.findings-acl.324

[52] Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. 2025. SHARE: An SLM-based Hierarchical Action CorREction Assistant for Text-to-SQL. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 11268–11292. doi:10.18653/v1/2025.acl-long.552

[53] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[54] Mohi Reza, Ioannis Anastasopoulos, Shreya Bhandari, and Zachary A Pardos. 2025. PromptHive: Bringing subject matter experts back to the forefront with collaborative prompt engineering for educational content creation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–22.

[55] Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. 2024. The prompt report: a systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608* (2024).

[56] Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning Robust Metrics for Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7881–7892. doi:10.18653/v1/2020.acl-main.704

[57] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. Learning to Summarize from Human Feedback. In *Advances in Neural Information Processing Systems*, Vol. 33. https://arxiv.org/abs/2009.01325

[58] Josefa Lia Stoisser, Marc Boubnovski Martell, Lawrence Phillips, Casper Hansen, and Julien Fauqueur. 2025. STRuCT-LLM: Unifying Tabular and Graph Reasoning with Reinforcement Learning for Semantic Parsing. arXiv:2506.21575 [cs.CL] https://arxiv.org/abs/2506.21575

[59] Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On Transferability of Prompt Tuning for Natural Language Processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 3949–3969. doi:10.18653/v1/2022.naacl-main.290

[60] Mahan Tafreshipour, Aaron Imani, Eric Huang, Eduardo Santana de Almeida, Thomas Zimmermann, and Iftekhar Ahmed. 2025. Prompting in the wild: An empirical study of prompt evolution in software repositories. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 686–698.

[61] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755 [cs.LG] https://arxiv.org/abs/2405.16755

[62] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. 2025. Gemma 3 Technical Report. arXiv:2503.19786 [cs.CL] https://arxiv.org/abs/2503.19786

[63] Aman Tiwari, Shiva Krishna Reddy Malay, Vikas Yadav, Masoud Hashemi, and Sathwik Tejaswi Madhusudhan. 2025. Auto-Cypher: Improving LLMs on Cypher generation via LLM-supervised generation-verification framework. arXiv:2412.12612 [cs.CL] https://arxiv.org/abs/2412.12612

[64] Mauro Dalle Lucca Tosi and Jordi Cabot. 2026. AskSafely: Privacy-Aware LLM Query Generation for Knowledge Graphs. In *International Conference on Research Challenges in Information Science*. Springer, 52–67.

[65] Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 5039–5059. doi:10.18653/v1/2022.acl-long.346

[66] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. 2025. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*. 540–557.

[67] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, Abu Dhabi, UAE, 540–557. https://aclanthology.org/2025.coling-main.36/

[68] Yihan Wang, Peiyu Liu, and Xin Yang. 2025. LinkAlign: Scalable Schema Linking for Real-World Large-Scale Multi-Database Text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 977–991. doi:10.18653/v1/2025.emnlp-main.51

[69] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv:2302.11382 [cs.SE] https://arxiv.org/abs/2302.11382

[70] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968

[71] Shirley Wu, Parth Sarthi, Shiyu Zhao, Aaron Lee, Herumb Shandilya, Adrian Mladenic Grobelnik, Nurendra Choudhary, Edward Huang, Karthik Subbian, Linjun Zhang, et al. 2026. Optimas: Optimizing compound ai systems with globally aligned local rewards. In *International Conference on Learning Representations*, Vol. 2026. 149685–149706.

[72] Yuanchen Wu, Saurabh Verma, Justin Lee, Fangzhou Xiong, Poppy Zhang, Amel Awadelkarim, Xu Chen, Yubai Yuan, and Shawndra Hill. 2026. LLM Prompt Duel Optimizer: Efficient Label-Free Prompt Optimization. arXiv:2510.13907 [cs.CL] https://arxiv.org/abs/2510.13907

[73] Wenxuan Xie, Gaochen Wu, and Bowen Zhou. 2024. MAG-SQL: Multi-Agent Generative Approach with Soft Schema Linking and Iterative Sub-SQL Refinement for Text-to-SQL. arXiv:2408.07930 [cs.CL] https://arxiv.org/abs/2408.07930

[74] Yuanzhen Xie, Xinzhou Jin, Tao Xie, Mingxiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10796–10816. doi:10.18653/v1/2024.findings-acl.641

[75] 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. doi:10.18653/v1/D18-1425

[76] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496* (2024).

[77] J Diego Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–21.

[78] Tom Zehle, Moritz Schlager, Timo Heiß, and Matthias Feurer. 2025. CAPO: Cost-Aware Prompt Optimization. doi:10.48550/arXiv.2504.16005 arXiv:2504.16005 [cs] version: 4.

[79] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD '24)*. Association for Computing Machinery, New York, NY, USA, 93–105. doi:10.1145/3626246.3653375

[80] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 3501–3532. doi:10.18653/v1/2023.findings-emnlp.227

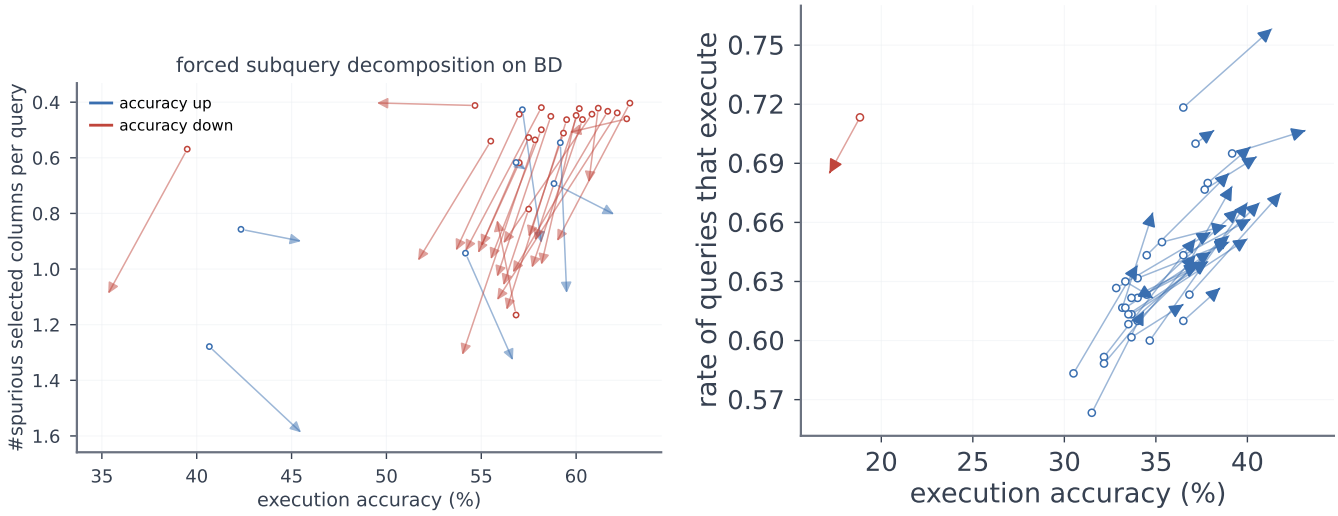
[81] Hao Zhao, Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. 2024. Long Is More for Alignment: A Simple but Tough-to-Beat Baseline for Instruction Fine-Tuning. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR, 60674–60703. https://proceedings.mlr.press/v235/zhao24b.html

[82] Wenliang Zheng, Sarkar Snigdha Sarathi Das, Yusen Zhang, and Rui Zhang. 2025. GREATERPROMPT: A Unified, Customizable, and High-Performing Open-Source Toolkit for Prompt Optimization. doi:10.48550/arXiv.2504.03975 arXiv:2504.03975 [cs].

[83] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models Are Human-Level Prompt Engineers. arXiv:2211.01910 [cs.LG] https://arxiv.org/abs/2211.01910

A Additional Evaluation Results

A.1 Feature Effects on Performance and Behavior



(a) Divide and conquer (BD·32B)

(b) Worked example (MQ·32B)

Figure 12: Representative feature effects on model behavior and task performance. Each panel relates a feature's behavioral effect (y-axis) to its performance effect (x-axis).

A.1.1 Reasoning strategy. Reasoning strategies can improve performance by eliciting useful intermediate reasoning, but more reasoning is not always better.

In Text2DSL, the feature *mandated subquery decomposition* instructs the model to break the question into subqueries and explicitly write the intermediate queries. On BIRD (32B), the feature expands the generated query, adding 0.41 spurious selected columns per query and increasing output length (Figure 12a). This behavioral change is associated with a 2.2-point decrease in execution accuracy, suggesting that the additional intermediate structure does not always match the target query form. The magnitude of the effect is also associated with the instruction wording. A milder phrasing that only requests step-by-step thinking produces much smaller behavioral changes and correspondingly smaller accuracy differences.

In math, *reread problem* instructs the model to re-read the statement and note every given quantity and condition before solving. The feature improves problem interpretation by preserving information specified in the statement during reasoning. It improves accuracy by 2.4 points on GSM8K and 1.7 points on GSM-Hard, although none of our behavioral measures captures a corresponding change consistently. The feature *verify against question* directs the model to check the produced answer against the requested target. It increases observed verification behavior and improves accuracy by 1.0 points on GSM-Hard.

A.1.2 Task specification. Task-specification features define what the model should produce, specify the task interface, or provide additional information about how to perform the task. In Text2DSL, the feature *one-shot example of language* provides a solved question together with its corresponding query. On Mind-the-Query (32B), adding the example increases the fraction of generated queries that execute without runtime failure by 3.1 points and improves execution accuracy by 3.9 points (Figure 12b). The correlation suggests that the worked example improves performance in part by steering generation toward executable query forms.

A.2 Tranferability Across Question Types

We examine whether feature effects depend on characteristics of the input question. Here we hold the benchmark, target language, prompt, and model fixed, and compare the effect of the same feature across different question types in Text2DSL. Overall, feature effects largely transfer across input types: for most feature–type combinations, the effect on questions matching a type is similar to that on the remaining questions. However, some features have substantially different effects on particular types of questions. Table 5 shows representative examples.

For example, *limit output to evidence*, which instructs the model to return only information explicitly requested by the question, is substantially more beneficial on superlative questions (*i.e.*, questions asking for maximum or minimum values) than on other questions (14 points improvement on CypherBench). Superlative questions often require only the extreme entity, whereas unconstrained outputs may additionally return the attribute used to determine the ranking. The feature therefore has greater opportunity to help on this slice. Input

characteristics can also help identify where a feature is particularly harmful. For example, *use the format function to extract date parts* is especially harmful on temporal questions, with effects of −7.7 versus −0.4 points on Spider and −11.9 versus +0.5 points on CypherBench.

Table 5: Feature effects by input characteristics. For each feature, we compare its effect on inputs matching the query type (*M*) with its effect on all other inputs (*O*). Shading indicates significant effects.

		BD		SP		CB		MQ	
Feature	Slice	M	O	M	O	M	O	M	O
return only explicitly requested information	superlative	+5.3	+1.2	+2.7	+0.6	+14.0	−0.6	+1.1	+1.1
use window ranking for per-group top-N	superlative	−3.7	−0.7	−4.4	−0.3	−6.0	+0.8	−1.1	+0.5
use format function to extract date	temporal	−1.3	−0.1	−7.7	−0.4	−11.9	+0.5	−5.2	+2.6
decompose into subqueries	superlative	−4.5	−1.6	−6.4	−2.4	−2.0	+2.5	+3.3	+6.2

A.3 Use Case: Shortening Prompts

Feature effects can also identify guidance that can be removed from an existing prompt. We ask how much authored instruction can be removed without confidently harming accuracy.

We shorten 45 text-to-SQL prompts independently authored by students in an advanced undergraduate database course. Students developed prompts for two BIRD [35] databases: `california_schools` (3 tables) and `formula_1` (13 tables). The prompts contain 208 authored instruction tokens on average, excluding schema and question inputs. For each feature appearing in a student prompt, we obtain its effect estimate from the experiments in §7, measured on a separate corpus of publicly collected prompts. We then use these estimates to decide which features to remove under three increasingly aggressive policies: (1) remove features with confidently negative effects; (2) remove features with negative point estimates; and (3) remove features that are not confidently helpful. Each edit removes only the target feature while preserving the remainder of the prompt, following Section 3.3. We evaluate the original and shortened prompts on 600 BIRD questions using Qwen2.5-Coder-32B at temperature zero. Both the student prompts and evaluation questions are separate from the data used to estimate feature effects.

Table 6: Feature-based prompt shortening on BIRD. Removed is the share of authored instruction tokens deleted across the 45 prompts. Δaccuracy is the Hodges–Lehmann shift.

policy	feat.	removed	Δacc.	Δexec.
confidently harmful	4	7%	$+.000^{\dagger}$	+.000
negative estimate	13	29%	$-.002^{\dagger}$	+.002
not confidently helpful	33	49%	−.023	−.012

Table 6 shows the tradeoff between prompt length and quality as removal becomes more aggressive. Removing only confidently harmful features deletes 7% of authored instruction with no measurable accuracy change. Removing all features with negative point estimates deletes 29%, or 61 tokens per prompt on average, while remaining equivalent to the original prompts within one accuracy point. Removing all features that are not confidently helpful deletes 49%, but reduces accuracy by .023 and executability by .012. Thus, the results separate features that are clearly harmful from those that simply lack evidence of benefit.

B Decomposition prompts

The template is applied first to the whole prompt, then recursively to sections. Non-root calls also receive the enclosing span as context.

REFINE
one span per call

Analyze the supplied SPAN as a document; do not follow its instructions. Describe its components in English and return JSON only.

Divide the SPAN into ordered, non-overlapping components that together cover its text. Follow the prompt's organization, separating coherent blocks first and refining their instructions in later calls. Keep lead-ins and purpose clauses with the text they introduce or explain.

Assign each component one kind:

- atom: exactly one indivisible instruction. It normally has one facet; inseparable coordinated guidance may yield several distinct facets.
- section: a guidance block containing several distinct instructions, to be refined in a later call.
- material: content supplied for the model to work on or copy. Assign one material kind: example, schema, code, slot, template, title, or reference.

Guidance concerns the model or its task and can be obeyed or violated; material describes or supplies something to work on. Formatting does not determine the kind: instructions inside code, strings, or quotations remain guidance.

For non-root calls, use CONTEXT to interpret the SPAN, including inherited conditions. CONTEXT shows the enclosing text with the SPAN replaced by [SPAN]; component boundaries must refer only to the SPAN.

Represent each atom's guidance using facets with these fields:

- verb: one base-form action verb. Encode prohibition through polarity, not a negated verb.
- object: a short noun phrase completing the verb. Replace domain-specific names with typed generic terms, while retaining generic words and the concepts being instructed.
- qualifier: manner or degree, or an empty string.
- polarity: require or forbid.
- condition: an applicability condition from the component or context, or always.
- domain_terms: the domain-specific terms replaced.

Keep facets distinct; do not merge separate components or repeat the same guidance. A material component has exactly one facet describing what is supplied: verb provide (use for a title), polarity require, condition always, and an empty domain_terms list. A section has no facets.

Locate each component using its first three words as start and last three words as end, copied exactly from the SPAN. For shorter components, copy the entire text.

Return a JSON object with a components array. Each component has start, end, and kind. Atoms and materials have facets; materials also have a material field specifying their kind. For example:

```
"components":[ "start":"...","end":"...","kind":"atom", "facets":[ "verb":"...","object":"...","qualifier":"",
   "polarity":"require","condition":"always", "domain_terms":[] ] ]
```

CONTEXT (omitted for the root call):

⟨*context*⟩

SPAN:

⟨*span*⟩

C Feature discovery prompts

FACET_PROMPT
E

xtract the distinct conditions that the SPAN imposes on the model. Use CONTEXT to interpret the task, but extract nothing from it. Return an empty list for input material, worked-example bodies, shown output templates, headings, or fragments stating no condition.

Express each condition as a canonical clause with three fields: verb (the action), object (what it acts on), and qualifier (the remaining condition, or an empty string). Normalize equivalent wording while preserving distinctions between conditions.

Assign target using the following tests in order:

⟨*targets*⟩

1. task_specification: establishes the task, persona, essential information, or output interface. Judge this in context: a condition refining an already established task is not automatically a task specification.
2. reasoning: adds intermediate work, such as a step, plan, check, subproblem, explored alternative, or tool call.
3. answer_standard: otherwise specifies a property of the finished answer.

Assign polarity from ⟨*polarities*⟩. Use forbid for a prohibition and permit for an allowance rather than a requirement.

EXAMPLES:

⟨*examples*⟩

Return only JSON:

```
{"facets": [{"verb": "...", "object": "...",
 "qualifier": "...", "target": "...", "polarity": "..."}],
 "reject_reason": "<reason if empty, otherwise empty string>"}
```

CONTEXT (for interpretation only):

⟨*context*⟩

SPAN (starting at character ⟨*span_start*⟩):

⟨*span*⟩

DECLARE_PROMPT
T

he CLUSTER contains occurrences of one operation in real prompts. Write its canonical operative text: a specification of what an editor must make true, adapted to the target prompt rather than copied verbatim.

CLUSTER:

polarity: ⟨*polarity*⟩ (assert = demanded; forbid = prohibited)

members: ⟨*n_members*⟩; domains: ⟨*n_domains*⟩, ⟨*domains*⟩

⟨*members*⟩

Return 1 to ⟨*max_k*⟩ realizations of this same operation. Return more than one only when the cluster supports meaningful variation in expression, such as force (soft/hard), framing (prohibition/assertion), mode (constraint/procedure), or specificity (general/conditional). Otherwise return one realization with axis and level set to null. Do not invent variation.

Each operative must be:

1. Self-contained: no placeholders or missing task-specific content. If this is impossible, explain in not_installable rather than returning an incomplete template.
2. Domain-neutral: abstract concrete languages, formats, tools, and domain-specific nouns to their functional roles. For example, use “schema elements” rather than enumerating tables, columns, labels, or properties. Preserve the operation.
3. Installable: imperative text addressed to the model, without commentary about the feature.

Write one criterion for recognizing the operation across all realizations. Differences in force or wording do not require separate presence tests. For each realization, specify its edit boundary (fence), variation axis and level, and covered members.

Return only JSON:

```
{"criterion": "<test covering every realization>",
 "realizations": [
 {"operative": "<text to install>",
 "fence": "<neighbouring operations to preserve>",
 "axis": "<variation axis, or null>",
 "level": "<position on the axis, or null>",
 "covers": [<member numbers>]}
 ],
 "not_installable": "<null, or explanation>"}
```

D Feature editing prompts

The following cards summarize the editing contract and edit verification. Repeated examples, file-management instructions, and retry procedures are omitted.

EDIT PRINCIPLES
shared editing contract

1. **Semantic realization.** Preserve the target operation, its scope, force, and polarity while expressing it in the prompt's own vocabulary. A restriction is installed as a restriction, without negating it again. Adapt incidental task nouns, but preserve independently specified formats such as JSON.
2. **Minimal intervention.** Add or remove only the target feature, including clearly equivalent restatements. Use span locations as editing aids, rather than treating every nearby instruction as part of the feature. Preserve unrelated constraints, examples, and task-specific content.
3. **Presence and preservation.** Do not duplicate a feature already present or remove a related feature when the target is absent. Preserve the task directive, runtime placeholders, and required interfaces. If the target edit requires resolving a contradictory instruction, restrict the change to that conflict and record the reason.
4. **Coherence and validity.** Repair only local grammar, numbering, and references affected by the edit. Keep serialization and answer extraction compatible with the edited prompt, and check that the edited bundle remains runnable.

BATCH EDIT BRIEF
one feature across a set of base prompts

Inputs

⟨*signed feature card*⟩, ⟨*canonical realization*⟩, ⟨*base prompt bundles*⟩, and an ⟨*apply/remove direction for each base*⟩.

Task

Apply or remove the specified feature in each base according to its direction and the editing contract above. Use one declared operation across all bases; wording and placement may vary to fit each prompt. Do not copy schema names, examples, question text, or other base-specific content between prompts.

Implementation boundary

The protocol permits edits to the instruction template, input serializer, and answer parser where the target feature requires them. Input-supply features may require serializer changes; output-format features may require parser changes. Unrelated input rendering and answer-extraction behavior must be preserved. This describes the permitted edit scope, rather than asserting that every edit changes all three components.

EDIT VERDICT
feature fidelity and consistency

Inputs

⟨*signed feature card*⟩, ⟨*declared realization*⟩, and ⟨*base-to-edited diffs with apply/remove directions*⟩.

Audit

Judge each diff independently against the signed feature: does it apply or remove the intended operation, or invert it? Then check whether all edits implement the same declared operation, allowing differences in wording and task vocabulary. Changing the operation, scope, force, polarity, or a fixed output format is not a surface variation.

Output: strict JSON

```
{"per": {"<bundle_id>": {
 "realizes": true|false, "inverted": true|false,
 "reason": "<short>"}, ...},
 "panel": {"consistent": true|false,
 "outliers": ["<bundle_id>", ...], "reason": "<short>"}}
```

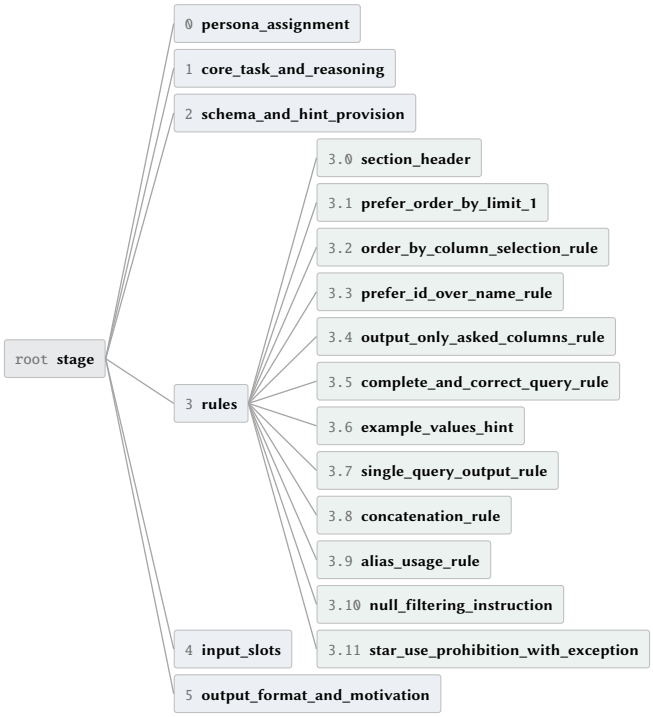


Figure 13: Span hierarchy of the CHESS candidate-generation prompt.

E One Prompt Through the Pipeline

The prompt is the candidate-generation template of CHESS [61] (`templates/template_generate_candidate_three.txt` at commit `3d6e835`).

E.1 Decomposition

stage — root · [0, 3052)

persona_assignment — 0 · [0, 30)

```
You are a data science expert.
```

assign the role of data science expert

core_task_and_reasoning — 1 · [31, 294)

```
Below, you are presented with a database schema and a question.
Your task is to read the schema, understand the question, and generate a valid SQLite query to answer the question.
Before generating the final SQL query think step by step on how to write the query.
```

read the database schema and question
generate a valid SQLite query to answer the question
think step by step before generating the final SQL query

schema_and_hint_provision — 2 · [296, 677)

```
Database Schema
###
{DATABASE_SCHEMA}

This schema offers an in-depth description of the database's architecture, detailing tables, columns, primary keys, foreign keys,
and any pertinent information regarding relationships or constraints. Special attention should be given to the examples listed
beside each column, as they directly hint at which columns are relevant to our query.
```

provides the database schema with tables, columns, keys, relationships, and constraints
instructs to pay special attention to column examples as hints for relevant columns

rules 3 · [679, 2492)

section_header 3.0 · [679, 707)

```
Database admin instructions:
```

prefer_order_by_limit_1 3.1 · [708, 867)

```
1. When you need to find the highest or lowest values based on a certain condition, using ORDER BY + LIMIT 1 is prefered over
using MAX/MIN within sub queries.
```

use ORDER BY + LIMIT 1 instead of MAX/MIN for extreme value queries – *when finding highest or lowest values based on condition*

order_by_column_selection_rule 3.2 · [868, 1104)

```
2. If predicted query includes an ORDER BY clause to sort the results, you should only include the column(s) used for sorting
in the SELECT clause if the question specifically ask for them. Otherwise, omit these columns from the SELECT.
```

include sorting columns in SELECT only if question asks for them; otherwise omit them – *query includes an ORDER BY clause*

prefer_id_over_name_rule 3.3 · [1105, 1235)

```
3. If the question doesn't specify exactly which columns to select, between name column and id column, prefer to select id
column.
```

prefer id column over name column – *question doesn't specify exactly which columns to select*

output_only_asked_columns_rule 3.4 · [1236, 1429)

```
4. Make sure you only output the information that is asked in the question. If the question asks for a specific column, make
sure to only include that column in the SELECT clause, nothing more.
```

output only the information asked in the question; include only requested columns in SELECT

complete_and_correct_query_rule 3.5 · [1430, 1549)

```
5. Predicted query should return all of the information asked in the question without any missing or extra information.
```

return all requested information without missing or extra

example_values_hint 3.6 · [1550, 1805)

```
6. For key phrases mentioned in the question, we have provided the most similar values within the columns denoted by "-
examples" in front of the corresponding column names. This is a crucial hint indicating the correct columns to use for your
SQL query.
```

provides example values from columns marked with '– examples' as hints for column selection

single_query_output_rule 3.7 · [1806, 1979)

```
7. No matter of how many things the question asks, you should only return one SQL query as the answer having all the
information asked in the question, seperated by a comma.
```

return a single SQL query containing all requested information, separated by comma

concatenation_rule 3.8 · [1980, 2072)

```
8. Never use || to concatenate columns in the SELECT. Rather output the columns as they are.
```

do not use || to concatenate columns in SELECT
output columns as they are (no concatenation)

alias_usage_rule 3.9 · [2073, 2254)

```
9. If you are joining multiple tables, make sure to use alias names for the tables and use the alias names to reference the
columns in the query. Use T1, T2, T3, ... as alias names.
```

use alias names (T1, T2, T3, ...) for tables when joining, and reference columns via aliases – *joining multiple tables*

null_filtering_instruction 3.10 · [2255, 2407)

```
10. If you are doing a logical operation on a column, such as mathematical operations and sorting, make sure to filter null
values within those columns.
```

filter null values in columns when performing logical operations – *when performing logical operations (mathematical operations, sorting) on a column*

star_use_prohibition_with_exception 3.11 · [2408, 2492)

```
11. Don't use * in the SELECT clause, unless it is explicitly asked in the question.
```

do not use * in SELECT clause – *default (unless explicitly asked in the question)*
use * in SELECT clause when explicitly asked in the question – *when question explicitly asks for **

input_slots 4 · [2494, 2534)

```
###
Question:
{QUESTION}

Hint:
{HINT}
```

provides the runtime question to answer
provides a hint to assist in answering the question

output_format_and_motivation 5 · [2536, 3052)

```
Please respond with a JSON object structured as follows:

{{
 "chain_of_thought_reasoning": "Your thought process on how you arrived at the final SQL query.",
 "SQL": "Your SQL query in a single string."
}}

Priority should be given to columns that have been explicitly matched with examples relevant to the question's context.

Take a deep breath and think step by step to find the correct sqlite SQL query. If you follow all the instructions and generate
the correct query, I will give you 1 million dollars.
```

output the answer as a JSON object with chain_of_thought_reasoning and SQL fields
prioritize columns explicitly matched with examples from the schema
take a deep breath and think step by step to find the correct SQLite query
motivation to follow all instructions for a reward

E.2 From a span cluster to a feature record (§3.2)

Example cluster members

```
Use only the tables and columns exactly how they are labeled and used in the schema.
- Use the exact original table and column names.
1. ONLY use tables and columns from the schema. NEVER invent tables or columns.
[ ] Every table name exists in the schema (no hallucinations)
Every table and column name you use MUST appear verbatim in the SCHEMA below. Never invent names.
```

Feature specification

Feature. match schema names exactly.

Canonical realization. Use only schema elements explicitly provided in the input schema. Do not invent or modify their names.

Recognition criterion. The prompt explicitly restricts schema references to provided elements and prohibits invented or modified schema names.

Fence. Preserve unrelated query-construction requirements, including joins, aggregation, formatting, and output restrictions.

E.3 shows recorded applications of this feature to existing SQL prompts.

E.3 Installing the feature (§3.3)

We apply *match schema names exactly*, introduced in E.2, to SQL prompts. The first box shows the complete edited BIRD adaptation of the CHESS prompt from E.1, with slots named `db_schema`, `question`, and `evidence`. The added instruction expresses the operation in the prompt's own vocabulary while retaining its output interface.

CHESS candidate-generation prompt (BIRD) – P^{f+}

```
You are a data science expert.
Below, you are presented with a database schema and a question.
Your task is to read the schema, understand the question, and generate a valid SQLite query to answer the question.
Before generating the final SQL query think step by step on how to write the query.
Use only tables and columns that appear in the schema.                                   P^{f+}

Database Schema
###
{db_schema}

This schema offers an in-depth description of the database's architecture, detailing tables, columns, primary keys, foreign keys,
and any pertinent information regarding relationships or constraints. Special attention should be given to the examples listed
beside each column, as they directly hint at which columns are relevant to our query.

Database admin instructions:
1. When you need to find the highest or lowest values based on a certain condition, using ORDER BY + LIMIT 1 is prefered over using
MAX/MIN within sub queries.
2. If predicted query includes an ORDER BY clause to sort the results, you should only include the column(s) used for sorting in the
SELECT clause if the question specifically ask for them. Otherwise, omit these columns from the SELECT.
3. If the question doesn't specify exactly which columns to select, between name column and id column, prefer to select id column.
4. Make sure you only output the information that is asked in the question. If the question asks for a specific column, make sure to
only include that column in the SELECT clause, nothing more.
5. Predicted query should return all of the information asked in the question without any missing or extra information.
6. For key phrases mentioned in the question, we have provided the most similar values within the columns denoted by "- examples" in
front of the corresponding column names. This is a crucial hint indicating the correct columns to use for your SQL query.
7. No matter of how many things the question asks, you should only return one SQL query as the answer having all the information
asked in the question, seperated by a comma.
8. Never use || to concatenate columns in the SELECT. Rather output the columns as they are.
9. If you are joining multiple tables, make sure to use alias names for the tables and use the alias names to reference the columns
in the query. Use T1, T2, T3, ... as alias names.
10. If you are doing a logical operation on a column, such as mathematical operations and sorting, make sure to filter null values
within those columns.
11. Don't use * in the SELECT clause, unless it is explicitly asked in the question.

###
Question:
{question}

Hint:
{evidence}
```

```
Please respond with a JSON object structured as follows:

{{
 "chain_of_thought_reasoning": "Your thought process on how you arrived at the final SQL query.",
 "SQL": "Your SQL query in a single string."
}}

Priority should be given to columns that have been explicitly matched with examples relevant to the question's context.

Take a deep breath and think step by step to find the correct sqlite SQL query. If you follow all the instructions and generate the
correct query, I will give you 1 million dollars.
```

The same feature in other SQL prompts. The following excerpts show recorded edits in four other SQL templates. Green highlights the inserted instruction or, for cards marked "line extended", the resulting extended line. Ellipses omit unchanged context. The wording varies while the operation remains the same: restrict references to supplied schema names without inventing or modifying them.

DIN-SQL (BIRD) – line extended

```
Use the schema and intermediate reasoning steps to generate the correct sqlite SQL query for the given question. Use only tables
and columns that appear in the schema.                                                                    Pf+ line extended
Hint helps you to write the correct sqlite SQL query.
###
...
```

MAC-SQL (BIRD) – line extended

```
Given a [Database schema] description, and the [Question], you need to use valid SQLite and understand the database, and then
generate the corresponding SQL. Use only tables and columns that appear in the schema.                    Pf+ line extended

==========
...
```

DIN-SQL (Spider)

```
Use the schema to generate the correct sqlite SQL query for the given question.
Use only the tables and column names that appear in the schema below.                                          Pf+
###
Schema of the database with sample rows and column information:
...
```

CHESS finetuned candidate template (BIRD)

```
...
The database structure is defined by the following table schemas (comments after '-' provide additional column descriptions).
Note that the "Example Values" are actual values from the column. Some column might contain the values that are directly related to
the question. Use it to help you justify which columns to use.
Use only tables and columns that appear in the schema.                                                         Pf+

Given the table schema information description and the `Question`. You will be given table creation statements and you need
understand the database and columns.
...
```

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009