Example-Driven Intent Synthesis for Constrained Data Bundle Retrieval: Focused Text Snippet Extraction and Beyond

Whanhee Cho
University of Utah
whanhee.cho@utah.edu

Kuangfei Long
Boston University
longkuangfei@outlook.com

Mahmood Jasim
Louisiana State University
mjasim@lsu.edu

Matteo Brucato
OSM Data
matteo@osm-data.com

Alexandra Meliou
UMass Amherst
ameli@cs.umass.edu

Peter J. Haas
UMass Amherst
phaas@cs.umass.edu

Anna Fariha
University of Utah
afariha@cs.utah.edu

ABSTRACT

Selecting a *bundle* of items that collectively satisfies constraints is a fundamental task across databases, recommender systems, and text summarization. Unlike traditional retrieval that returns individual or top-k items, bundle retrieval is inherently combinatorial and, in general, NP-hard. Although package queries can efficiently retrieve bundles given a well-formed query, two key user-centric challenges remain: (1) expressing and tuning multi-dimensional bundle intent through a user-friendly interface, and (2) ensuring feasibility when the query yields empty results. We introduce Ex2Bundle, an *Example-driven Bundle retrieval* framework that enables users to specify their intent through *example bundles* and automatically synthesizes package queries that capture the intent implicit in those example bundles via aggregate constraints. Ex2Bundle also addresses a challenge unique to bundle retrieval: when inferred aggregate constraints are infeasible over the target data, our *data-aware constraint relaxation* minimally adjusts the constraint bounds while preserving alignment with user intent. We instantiate a specific application of *focused text snippet extraction by example* to demonstrate the efficacy of the Ex2Bundle framework. Extensive experiments over real-world datasets and a user study demonstrate that Ex2Bundle improves usability and consistently returns intent-aligned bundles even under distributional shifts of the target database.

Artifact Link: https://github.com/kuangfei-long/ex2bundle.

1 INTRODUCTION

Data *bundle* retrieval is a fundamental task across multiple domains: package queries retrieve packages (sets of tuples) from relational databases under aggregate constraints [10]; recommender systems suggest playlists (sets of songs) or combo offers (sets of products) tailored to user preferences [4]; and extractive text summarization systems retrieve subsets of sentences from documents that form coherent summaries based on user queries [19, 53, 75]. Unlike traditional retrieval that returns individual or top-k items, bundle retrieval involves selecting a *set* (or package [10]) whose elements must collectively optimize global objectives while satisfying set-level constraints that align with user preferences. Consequently, the inclusion or exclusion of any single item affects the feasibility and overall quality of the bundle, making the problem inherently combinatorial and NP-hard [10]. While systems exist for the efficient retrieval of data bundles from this combinatorial search space [10], two user-centric challenges remain as primary obstacles: (1) the *interface* challenge—how can humans easily express their intent of the desired data bundles—and (2) the *feasibility* challenge—how to ensure non-empty answers for the user queries.

Challenge 1: Usable interface. The first challenge is communicating the user intent of the desired data bundle. Like SQL, the Package Query Language (PaQL) [10] allows specification of a bundle query through linear constraints and objective functions over data attributes, but the users must know the exact query parameters: attribute names and their bounds as numerical values. This is inherently difficult for human users—especially non-experts—who must (i) learn the PaQL [10] syntax, (ii) know the database schema, (iii) determine the attributes of interest, (iv) have a good sense of the value distributions, and (v) translate their intent precisely to parameterized constraints and global objectives of a PaQL query. We show in Example 1 that even for experts, step (v) is particularly difficult for PaQL because constraints are over *bundle aggregates* (SUM, AVG, COUNT): users must reason at the bundle level—predicting what a SUM should be over a bundle of unspecified size—rather than at the tuple level as in SQL. For certain applications, such as extractive document summarization [75], an alternative is to specify the intent in natural language. However, as prior work shows, natural language is too generic to express specific user intents [19] and subsequent conversational tuning of intent is imprecise and frustrating for humans [78].

Example 1 (Synthesizing a package query). *Morpheus, a CS department Chair, is seeking to fill two tenure-track assistant professor positions to strengthen AI and Databases. Since he plans for the two new hires to jointly teach an "AI in Databases" course, he wants them to have reasonable collective teaching experience, but not too much, as that may indicate teaching-focused or senior candidates who are unlikely to accept a tenure-track entry-level position. He also wants to maximize the total recommendation score of the new hires based on their letters. This is a bundle retrieval problem: Morpheus must pick a set of two out of* 200 *candidates that best satisfy his criteria, as shown via the following under-specified package query:* [1]

```
Q1:   SELECT PACKAGE(*) FROM CANDIDATES
      SUCH THAT COUNT(*) = 2
      AND SUM(ai_score) is high
      AND SUM(db_score) is high
      AND SUM(teaching_score) is reasonable
      MAXIMIZE SUM(reco_score);
```

[1] For ease of exposition, we use a toy example, albeit somewhat unrealistic, of 2 hires over 4 simple criteria; in practice, faculty searches involve more candidates (5–10) over more complex criteria (15+). The search space grows combinatorially with the #candidates. E.g., choosing 5 out of 200 yields $\binom{200}{5} \approx 2.5$ billion possible bundles.

	AI	Databases	Teaching	Recommendation
	`ai_score`	`db_score`	`teaching_score`	`reco_score`
Smith	0.8	0.4	0.1	0.4
Jones	0.4	0.7	0.2	0.9
Neo	0.4	0.5	0.4	0.8
Brown	0.5	0.4	0.4	0.9
…	…	…	…	…

Table 1: Partial list of candidates for Example 1.

However, at this point, all Morpheus has are the raw application materials (resumes, statements, letters, etc.) of these candidates. Before querying the candidate database, he must first map each candidate's application materials to a 4-dimensional vector space: `ai_score`*,* `db_score`*,* `teaching_score`*, and* `reco_score`*. Morpheus decides to use an off-the-shelf embedding technique to obtain numeric scores for each of these dimensions for every candidate (Table 1). However, he now faces another problem: he does not know what constitutes an appropriate replacement for the underspecified values high and reasonable in* `Q1`*. Is* 0.7 *considered a high score for AI expertise? What value represents the reasonable range for teaching score? He can examine the value distributions of the embeddings, but still cannot determine which corresponding values would accurately capture his intended criteria.*

Example 1 highlights two key struggles in formulating package queries to express bundle query intent: (1) the lack of means to accurately map a database to a vector database with appropriate dimensions of interest, which primarily affects non-experts; and (2) the lack of knowledge of the correct parameters for the PaQL query, which affects experts and non-experts alike. In Example 1, Morpheus was interested in only 4 criteria; however, his struggle would be even worse if there were a larger number of criteria.

Our solution to Challenge 1: bundle-query by example. To overcome the usability challenge in the interface of bundle query intent specification, we build on the *Query by Example* (QbE) paradigm [82], which has seen significant success in traditional SQL querying [21] and other domains [19, 27, 28, 57] by lowering the barrier to task specification. We extend QbE to bundle queries (*BQbE*): the user provides a few exemplar data bundles to convey their query intents *implicitly*, thereby bypassing the construction of a precisely parameterized PaQL query. BQbE is particularly useful when users cannot articulate precise criteria but can instead provide representative examples of what they seek. For a large number of use cases, BQbE significantly lowers the barrier to bundle querying (§2).

Example 2 (Parameterizing a package query). *Continuing from Example 1, Morpheus recalls that strong new hires resemble recent hires at top universities, such as {Trinity, Cypher} (University X) and {Link, Niobe, Seraph} (University Y). These exemplars excel in Databases and AI with reasonable collective teaching experience. Morpheus computes their scores across 4 dimensions using the same embedding of Example 1 (Table 2). Guided by the aggregated scores, he parameterizes the underspecified query* `Q1` *to obtain:*

```
Q2:   SELECT PACKAGE(*) FROM CANDIDATES
      SUCH THAT COUNT(*) = 2
      AND SUM(ai_score) BETWEEN 0.8 AND 1.2
      AND SUM(db_score) BETWEEN 1.0 AND 1.3
      AND SUM(teaching_score) BETWEEN 0.7 AND 1.3
      MAXIMIZE SUM(reco_score);
```

	AI	Databases	Teaching	Recommendation
	`ai_score`	`db_score`	`teaching_score`	`reco_score`
Example 1: University X hires				
Trinity	0.6	0.5	0.3	0.7
Cypher	0.2	0.8	0.4	0.9
SUM	**0.8**	**1.3**	**0.7**	**1.6**
Example 2: University Y hires				
Link	0.9	0.3	0.4	0.6
Niobe	0.2	0.3	0.6	0.8
Seraph	0.1	0.4	0.3	0.7
SUM	**1.2**	**1.0**	**1.3**	**2.1**
Morpheus' (implicit) intent				
	[0.8, 1.2]	**[1.0, 1.3]**	**[0.7, 1.3]**	**Maximize**

Table 2: Example bundles help discover query parameters.

bundle		`ai`	`db`	`teaching`	`reco`	valid?
b_1	{Smith, Jones}	1.2	1.1	0.3 (Fail)	1.3	**no**
b_2	{Smith, Neo}	1.2	0.9 (Fail)	0.5 (Fail)	1.2	**no**
b_3	{Smith, Brown}	1.3 (Fail)	0.8 (Fail)	0.5 (Fail)	1.3	**no**
b_4	{Jones, Neo}	0.8	1.2	0.6 (Fail)	1.7	**almost**
b_5	{Jones, Brown}	0.9	1.1	0.6 (Fail)	1.8	**almost**
b_6	{Neo, Brown}	0.9	0.9 (Fail)	0.8	1.7	**almost**

Table 3: No candidate bundle fully satisfies the constraints of `Q2`.

Example 2 highlights a scenario where a user cannot directly specify precise criteria via PaQL query parameters for bundle retrieval but can provide valid *examples* to implicitly convey the criteria. These examples can be used to automatically learn user preferences and translate them to query parameters. However, another key challenge remains, which we highlight in Example 3.

Example 3 (Ensuring query feasibility to obtain results.). *Morpheus issues* `Q2` *to a package query execution engine [10], but it returns no result. A close inspection reveals that indeed none of the 2-candidate bundles fully satisfies* `Q2` *(Table 3). However, the last three bundles "almost" satisfy the constraints. If the* `db_score` *constraint had a slightly relaxed lower bound (0.9 instead of 1.0), then b_6 would be a valid bundle. Similarly, slightly relaxing the lower bound of the* `teaching_score` *constraint (0.6 instead of 0.7) would result in two additional valid bundles: b_4 and b_5. Then following the goal of maximizing the* `reco_score`*, the bundle b_5 would be the final result.*

Challenge 2: ensuring feasibility. While BQbE addresses the first challenge, it brings forth a second one that is particular to bundle retrieval and absent from traditional single-tuple QbE: constraints inferred from example bundles may yield queries that are *infeasible* over the target data, returning empty answers [47, 48, 52], because aggregate constraints can be jointly unsatisfiable even when individually plausible, especially under distribution shift between the example and target domains. In Example 3, even with a fully formed PaQL query, with the desired constraints and specific parameters, it turned out to be *infeasible* over the target data (Morpheus' University). This happened because the data distributions differed between the example domains (Universities X and Y) and the target domain (Morpheus' University), a common scenario in practice [35, 58]. Existing package query execution systems [10] provide no further insight into how the query parameters interact with the target data, leaving users uncertain about how to tune the parameters to make the query feasible such that valid results are retrieved.

Our solution to Challenge 2: data-aware relaxation of query parameters. To ensure feasibility even when the original query is infeasible, we introduce *data-aware constraint relaxation techniques* that adapt the synthesized PaQL query to the target data by adjusting constraint bounds. By analyzing the data distribution of the target database, we identify which constraints act as bottlenecks that prevent "almost-valid" bundles from being retrieved. This analysis guides us to relax the appropriate constraints, while minimizing deviation from the user's original intent.

Why existing approaches fall short. Three classes of relevant approaches exist, but all of them fail to support BQbE.

PaQL and QbE systems. PaQL engines [10] require fully parameterized queries and return empty results when infeasible; QbE systems [21, 82] reduce the parameter burden for single-tuple retrieval but do not extend to bundle queries with aggregate constraints.

Top-k retrieval over vector DBs. For BQbE, an alternative solution is to treat the entire example bundle as a single query vector and retrieve top-k similar tuples based on vector similarity [46, 61] to assemble the result bundle. However, this approach can miss globally optimal bundles and violate intended constraints. The main issue is that top-k retrieval evaluates items independently using an implicit scoring function—without guarantees of global optimality—whereas bundle retrieval requires reasoning over combinations of tuples under globally enforced [81], multi-dimensional aggregate constraints. RAG systems inherit the same limitation, as they perform top-k nearest-neighbor search over individual tuples (but not their combinations) before passing results to a generative model. Another alternative is to match each tuple in the example bundle independently and combine their top matches to form the result bundle; yet this approach breaks down when the example and target bundle sizes differ and still offers no optimality guarantees. In summary, bundle retrieval is fundamentally an NP-hard combinatorial optimization problem, and similarity-based greedy top-k search is ill-suited to address it.

Conversational LLMs. An LLM prompted with examples (few-shot learning) could produce the bundle directly, but this requires reasoning over the entire target data—infeasible for large data—and solving a combinatorial constrained problem, which LLMs handle unreliably [70]. An LLM could instead produce code that extracts the bundle. Text-to-SQL is well-studied [23], but SQL operates on individual tuples and cannot natively express aggregate-constrained set selection. PaQL closes this gap; an LLM could plausibly synthesize PaQL queries similarly, though text-to-PaQL is less established. Even so, the bounds would reflect an opaque LLM-internal policy mapping the user's examples to constraint values (e.g., element-wise min/max or mean$\pm\sigma$), with no verification. The LLM also has no native mechanism to detect infeasibility or relax bounds in an intent-preserving way; outputs are non-deterministic across versions, and per-query latency limits interactive use. In a pure NL interface, without examples to anchor the bounds, the user must rely on conversational refinement, which is imprecise and tedious [78].

Desiderata. Examples 1–3 and the limitations of alternative approaches motivate the desiderata of an ideal BQbE system:

D1 Ensure a user-friendly interface for specifying intents, where users can provide examples to implicitly convey their bundle query intents, without requiring any knowledge of the database schema and complex query syntax and parameters.

D2 Accurately model the user intent implicit in the examples into a transparent intermediate representation with explicit parameters, such as a PaQL query.

D3 Ensure feasibility of the synthesized PaQL query over the target database, yielding a non-empty result even when no bundle perfectly matches the user's intent.

D4 Provide an intuitive interface for iterative intent refinement, allowing users to adjust query constraints in a way that guarantees a valid result.

Ex2Bundle. We introduce Ex2Bundle, an example-driven data bundle retrieval system that (i) enables users to specify bundle query intent through example bundles, addressing D1; (ii) synthesizes a package query from these examples to transparently capture user intent, addressing D2; (iii) applies data-aware constraint relaxation to ensure feasibility of the synthesized query w.r.t. the target data domain—particularly when its distribution differs from that of the example data domain—addressing D3; and (iv) provides an intuitive interface for interactive intent refinement, addressing D4.

In the context of text summarization, BQbE arises as *focused text snippet extraction*: given a "focus" in the form of several pairs of source document and corresponding example summary—expressed as a set of extracted sentences from the document—the goal is to select a set of sentences from a target document that collectively forms an extractive summary consistent with those examples. Our preliminary work SuDocu (summarizing documents by example), published as a demonstration paper at VLDB [19], involved a simple solution for focused text snippet extraction. In this work, we significantly extend it to build a generalized and more robust framework to support bundle retrieval tasks across multiple domains.

Contributions. We make the following contributions:

- We introduce a novel paradigm for specifying *bundle retrieval intents through examples*, and highlight its broad applicability across real-world applications in several domains (§2).
- We formalize the problem of *example-driven bundle retrieval* and show how it translates to the package query framework (§3).
- We present Ex2Bundle, an end-to-end framework for *example-driven bundle retrieval* that synthesizes PaQL query constraints from user examples. We contribute a *data-aware constraint-bound relaxation technique* that ensures query feasibility. We further introduce design principles for an interactive slider-based interface for intent refinement, along with effective techniques to translate between sliders and constraint bounds (§4).
- We evaluate Ex2Bundle on two use cases—package queries over the TPC-H dataset [69] and focused text snippet extraction over real-world datasets [30, 76]—showing that Ex2Bundle achieves 100% constraint satisfaction while maintaining competitive objective scores, and outperforms retrieval and extractive baselines. Notably, Ex2Bundle remains competitive with LLM-based approaches without incurring any additional token or inference cost, while scaling to realistic workloads (§5).
- Our user study shows that Ex2Bundle achieves high user satisfaction, due to improved ease of use and customizable sliders, for the task of focused text snippet extraction (§6).

2 EXAMPLE USE CASES

We now present three real-world applications where example-driven intent specification lowers the barrier for data bundle retrieval: *supplier selection* for business, *focused snippet extraction* from text documents, and *playlist recommendation* in streaming services.

Supplier selection for business expansion into a new country. Bundle retrieval by example is valuable in business settings where an owner seeks to expand into a new country and must identify a set of new *suppliers* to establish relationships with. The objective is to select a bundle of about 100 suppliers from a large candidate pool over 10,000 suppliers (similar to the TPC-H [69] dataset). Manually constructing an optimal supplier set here is tedious and error-prone, as the selected suppliers must collectively satisfy a range of constraints. Formulating these requirements as an explicit package query is equally challenging: business owners often lack precise knowledge of the exact parameter values for their desired constraints in an unfamiliar market (e.g., acceptable price ranges, inventory availability, or financial stability thresholds). However, they can readily provide example bundles of suppliers drawn from countries where their business already operates successfully. Ex2Bundle can leverage these examples to infer the underlying implicit constraints, adjust them to the new supplier database of the target country, and retrieve an optimal set of suppliers that collectively satisfies the constraints.

Focused text snippet extraction. Another use case of Ex2Bundle is focused snippet extraction from text documents, also termed as personalized extractive summarization [76]. Here, the goal is to select (extract) a subset of sentences (snippet) from a text document that, collectively, best aligns with user's information need (focus). For example, when a user wishes to extract sentences from a technical paper that are most relevant to their own research interests. While such intent can, in principle, be expressed in natural language, prior work [19, 76] shows that articulating subjective intents this way is difficult and often leads to user fatigue [78] due to iterative, back-and-forth intent refinement via conversations. Instead, users can provide a few example ⟨document, snippet⟩ pairs to implicitly communicate their focus [32, 62, 66, 77]. For instance, a journalist summarizing reports of all U.S. states with a particular focus—e.g., "basic" economic information, "moderate" coverage of education, and "strong" emphasis on technology—can highlight sentences from a few state's reports to form example snippets. From these exemplars, Ex2Bundle can infer the journalist's focus, transparently encode it as PaQL constraints, and apply them to new state reports to extract snippets with the same focus. Notably, for reports with substantially different structure or content, Ex2Bundle can minimally adjust these inferred constraints to ensure feasible snippet extraction.

Playlist recommendation for streaming services. In streaming services, users seek recommendations for *bundles* of items that collectively satisfy their preferences. Spotify, YouTube Music, and Netflix exemplify this through playlist/watchlist recommendations (e.g., Discover Weekly [63], Your Daily Discover [7], and Top Picks for You [3]), where the goal is to generate a set of songs/movies aligned with user interest. Existing approaches typically rely on item-level similarity—recommending songs similar to those in a user's listening history—or on similarity to a single reference playlist. For instance, a user who frequently listens to `AC/DC` may receive playlists dominated by rock songs, or playlists resembling those curated for users with similar histories. While effective in homogeneous preference settings, such approaches struggle to capture diverse tastes and is notoriously known to cause user frustration, as evident from complaints about lack of variety in Spotify's Discover Weekly playlist [1]. Consider a user whose taste spans multiple genres—predominantly energetic rock, with a mix of heavy metal, some soft country songs, alongside occasional calm classical music. A playlist based only on rock similarity or a single reference list cannot capture such diversity. Moreover, explicitly specifying such nuanced preferences (e.g., how to balance genre, artist, mood, and tempo) is difficult, even with natural language or LLM-based interfaces. Example-driven bundle recommendation addresses this: users provide a few example playlists that reflect their desired diversity across genres, artists, tempo, and mood, and Ex2Bundle infers implicit constraints to generate playlists that collectively satisfy the inferred constraints.

3 PROBLEM FORMULATION

In this section, we formalize the problem of example-driven bundle retrieval. We begin by introducing how we model the *source* and *target data*—from which bundles are retrieved—and *example bundles*—which communicate the user's intent. We then show how bundle retrieval can be naturally modeled within the package query framework and define the problem of *example-driven package query synthesis* for bundle retrieval, our focus in this paper. Table 4 provides a summary of notations used throughout the paper.

Source and target data. In example-driven bundle retrieval, each example bundle is taken from a corresponding source data. In Example 2, {Trinity, Cypher} is an example bundle from the source data of University X and {Link, Niobe, Seraph} from the source data of University Y. One key requirement here is that the source data for all the example bundles must have the same schema, to allow for effective intent discovery across the shared attributes.

We define a schema $\mathbf{f}$ of a single table[2] over K numerical[3] attributes/features $\{f_1, \ldots, f_K\}$. Each source data T_i^s must conform to the schema $\mathbf{f}$, denoted as $T_i^s \models \mathbf{f}$. In Table 2, the source data

[2]While our formalization focuses on a single-table schema, it is not an inherent limitation: we support relational databases by joining tables into a denormalized table, as shown in our experiments over the TPC-H dataset (Section 5).

[3]For unstructured data such as text, we derive numerical features through domain-specific methods such as topic modeling or learned vector embeddings (Section 4) to produce a structured representation under a fixed numerical schema.

Symbol	Description
$\mathbf{f} = \{f_1, \ldots, f_K\}$	Common schema over K attributes/features
$T_i^s \models \mathbf{f}$	A source data over the schema $\mathbf{f}$
$t \models \mathbf{f}$	A single tuple over the schema $\mathbf{f}$
$\mathbf{f}(t) = [f_1(t), \ldots, f_K(t)] \in \mathbb{R}^K$	Feature vector of tuple t
$E_i \subseteq T_i^s$	Example bundle for the source data T_i^s
$\mathbf{T}^s = \{T_1^s, \ldots, T_N^s\}$	Set of N source data
$\mathbf{E} = \{E_1, \ldots, E_N\}$	Set of N user-provided example bundles
$T^q \models \mathbf{f}$	A target data over $\mathbf{f}$ to retrieve a bundle from
$B \models \mathbf{f}$	A bundle over the schema $\mathbf{f}$
$\Theta = \{\langle \mathtt{lb}_j, \mathtt{ub}_j \rangle\}_{j=1}^K$	Constraint bounds for K attributes
$\mathbf{F}(B) = [F_1(B), \ldots, F_K(B)]$	Feature profile of bundle B
$\mathbf{F}(B) \vdash \Theta$	$\mathbf{F}(B)$ satisfies the constraints given by Θ
$\sigma(t) \mapsto \mathbb{R}$	Domain-specific tuple-level scoring function

Table 4: Table of notations. Bold letters denote sets or vectors.

for all the Universities conform to the schema {ai_score, db_score, teaching_score, reco_score}. A tuple t over $\mathbf{f}$, denoted as $t \models \mathbf{f}$, is characterized by a feature vector $\mathbf{f}(t)=[f_1(t), \ldots, f_K(t)] \in \mathbb{R}^K$. For instance, in Table 2, Trinity's feature vector is $[0.6, 0.5, 0.3, 0.7]$. We use $T^q \models \mathbf{f}$ to denote the *target data*, the data from which the user wants to extract their desired bundle from. In Example 2, Table 1 represents the target data.

Example bundles. In example-driven bundle retrieval, users convey their intent via a set of *example bundles* $\mathbf{E} = \{E_1, \ldots, E_N\}$, where each $E_i \subseteq T_i^s$ consists of a subset of tuples from the corresponding source data T_i^s. In Example 2, $\mathbf{E}$ = {{Trinity, Cypher}, {Link, Niobe, Seraph}} and the source data set $\mathbf{T}^s = \{T^s_{\text{UofX}}, T^s_{\text{UofY}}\}$.

3.1 Example-driven bundle retrieval

We are now ready to (informally) define our problem:

Problem 3.1 (Example-Driven Bundle Retrieval). *Given a set of source data $\mathbf{T}^s$ over the same schema $\mathbf{f}$, corresponding example bundles $\mathbf{E}$, where $E_i \in \mathbf{E}$ is an example bundle for the source data $T_i^s \in \mathbf{T}^s$, and a single target data $T^q \models \mathbf{f}$, identify a bundle $B^* \subseteq T^q$ that (i) satisfies the "user intent" implicit in $\mathbf{E}$ w.r.t $\mathbf{T}^s$ and (ii) is the "best" among such bundles.*

However, this problem is underspecified: it is unclear how to model the user intent and define "best" or the notion of optimality.

3.1.1 Modeling user intent. In its most general form, capturing user intent from example bundles may require arbitrarily complex models—non-linear constraints, distribution matching, or opaque learned representations. However, the use cases in Section 2 share a common structure: (i) the desired bundle is a *set* of items, (ii) quality of the bundle depends on some *aggregate* properties (e.g., total score) of the set, and (iii) the user's intent is expressible as *bounds* on those aggregates. These are precisely the defining characteristics of a *package query* [10]—which retrieves subsets of tuples from relational tables that collectively satisfy (linear) aggregate constraints while optimizing a (linear) global objective—as illustrated by Q2 in Example 2. We therefore model user intent as *linear constraints* on the bundle's *feature profile*, with an optional cardinality constraint on the bundle size. The benefit of this modeling is twofold: it provides *interpretability*—the resulting constraints are human-readable—and *tractability*—it maps to Integer Linear Programming (ILP) with well-established solvers and foundations [10].

Feature profile of a bundle. To capture the feature-wise properties of a bundle B, we define its *feature profile* $\mathbf{F}$ as a vector obtained by aggregating, for each feature, the values of that feature across all tuples in B. Formally,

$$\mathbf{F}(B) = [F_1(B), \ldots, F_K(B)], \quad \text{where } F_j(B) = \mathcal{A}_{t \in B} F_j(t)$$

Here, $\mathcal{A}$ is an aggregate such as SUM. In Table 2, using SUM as the aggregate, the feature profile of the bundle {Trinity, Cypher} is [0.8, 1.3, 0.7, 1.6]—the column-wise sums for University X hires.

Formulating constraints. Following prior work in QbE [21], given a set of user-provided example bundles, we posit that the user's intended result bundle would exhibit a feature profile similar to that of the example bundles. Accordingly, we model the user's intent as a *set of bounded constraints over the feature profile* of the desired bundle, where the derivation of the bounds should be guided by the feature profiles of the example bundles in $\mathbf{E}$, the source data set $\mathbf{T}^s$, and the target data T^q. Formally:

$$\{\texttt{lb}_1 \leq \mathbf{F}_1(B) \leq \texttt{ub}_1, \ldots, \texttt{lb}_K \leq \mathbf{F}_K(B) \leq \texttt{ub}_K\}$$
$$\text{where, } \texttt{lb}_j \text{ and } \texttt{ub}_j \text{ depend on } \mathbf{E}, \mathbf{T}^s, \text{ and } T^q \text{ for } 1 \leq j \leq K$$

We use $\Theta = \{\langle \texttt{lb}_1, \texttt{ub}_1 \rangle, \ldots, \langle \texttt{lb}_K, \texttt{ub}_K \rangle\}$ to denote constraint bounds for all K attributes over the schema $\mathbf{f}$. The notation $\mathbf{F}(B) \vdash \Theta$ denotes that the feature profile of the bundle B satisfies the constraint bounds specified by Θ, i.e., $\forall\, 1 \leq j \leq K,\ \texttt{lb}_j \leq F_j(B) \leq \texttt{ub}_j$.

3.1.2 Defining bundle optimality. The second issue in Problem 3.1 is that, when multiple candidate bundles satisfy Θ, a principled way is required to quantify their quality and ensure optimality. To this end, we assume knowledge of a tuple-level function $\sigma(t) \mapsto \mathbb{R}$, which is application-specific and is typically provided by a domain expert who configures the system for end users. We then define the quality of a bundle B by aggregating σ over all $t \in B$, which naturally fits the linear objective function optimized by the package query framework. For example, this could correspond to maximizing the total recommendation score in Example 1, or minimizing the word count in text summarization. Alternatively, σ can be learned from feedback signals reflecting users' perceived quality of returned bundles in a human-in-the-loop setting, which is especially practical for playlist recommendation in music streaming (Section 2).

3.2 Example-driven package query synthesis

With our models of user intent and bundle optimality, which naturally align with the package query framework, we reformulate Problem 3.1 of example-driven bundle retrieval as that of synthesizing a package query—specifically, its parameters—from example bundles. The synthesized package query serves as a *mechanism* for efficiently retrieving the optimal result bundle.

Given a target data T^q, constraint bounds $\Theta = \{\langle \texttt{lb}_j, \texttt{ub}_j \rangle\}_{j=1}^K$, and (optional) cardinality constraint bounds $C = \langle \texttt{lb}_c, \texttt{ub}_c \rangle$ we fix the following parameterized package query:

$$\begin{aligned}\text{PQ}(T^q, \Theta, C):\ & \textbf{SELECT}\ \texttt{PACKAGE(*)}\ \textbf{AS}\ \texttt{B}\ \textbf{FROM}\ T^q \\ & \textbf{SUCH THAT COUNT}\texttt{(B)}\ \textbf{BETWEEN}\ \texttt{lb}_c\ \textbf{AND}\ \texttt{ub}_c \\ & \textbf{AND}\ F_j(B)\ \textbf{BETWEEN}\ \texttt{lb}_j\ \textbf{AND}\ \texttt{ub}_j\ \ \forall 1 \leq j \leq K \\ & \textbf{MAXIMIZE}\ \mathop{\mathcal{A}}_{t \in B} \sigma(t)\end{aligned}$$

Here, $\mathcal{A}$ aggregates the tuples $t \in B$ using σ to compute the quality of B, which the package query aims to maximize as its objective. Without loss of generality, a minimization objective can be expressed as a maximization objective by simply negating the objective. The choice of aggregation functions (including those used to compute the feature profile $\mathbf{F}(B)$) and the scoring function σ is typically guided by the application domain. These are system parameters determined during setup and are not optimized over. We discuss how to choose the aggregation and scoring functions in Section 4.2. Also, without loss of generality, the cardinality constraint $\texttt{lb}_c \leq \texttt{COUNT}(B) \leq \texttt{ub}_c$ can be subsumed into the feature-profile constraints by augmenting the feature profile with an additional dimension representing bundle cardinality. Hence, we do not explicitly include the cardinality constraint in the remainder of the paper.

In Example 2, Q2 is a package query over the schema $\mathbf{f}$={ai_score, db_score, teaching_score}, so $K = 3$. The aggregation used to compute the feature profile is SUM, and the objective is to maximize SUM(reco_score) (i.e., σ = reco_score).

While Morepheus manually derived the constraint bounds in Θ, the resulting query was infeasible over the target data T^q representing his university. Thus, the main requirement is to determine the parameter Θ, and use it to synthesize a package query that will retrieve the optimal bundle $B \subseteq T^q$, as specified in Problem 3.1.

Problem 3.2 (Example-driven package query synthesis). *Given a schema* $\mathbf{f}$, *a set of source data* $\mathbf{T}^s = \{T_1^s, \dots, T_N^s\}$ *where each* $T_i^s \models \mathbf{f}$, *corresponding example bundles* $\mathbf{E}$, *a target data* $T^q \models \mathbf{f}$, *and a tuple-level scoring function* $\sigma : t \mapsto \mathbb{R}$, *together with the corresponding aggregator* $\mathcal{A}$, *determine a synthesis function* $G : (\mathbf{E}, \mathbf{T}^s, T^q) \mapsto \langle \mathbb{R}, \mathbb{R} \rangle^K$ *to derive the constraint bounds* Θ *such that:*

(1) ***Feasibility****: the package query* $PQ(T^q, \Theta)$ *returns a non-empty result over* T^q *ensuring* $\exists B \subseteq T^q$ *s.t.* $\mathbf{F}(B) \vdash \Theta$.
(2) ***Optimality****: the retrieved bundle* $B^* = PQ(T^q, \Theta)(T^q)$ *maximizes the objective, i.e.,* $B^* = \underset{B \subseteq T^q \text{ s.t. } \mathbf{F}(B) \vdash \Theta}{\arg\max} \underset{t \in B}{\mathcal{A}} \sigma(t)$.

The core challenge here is to determine the synthesis function $G : (\mathbf{E}, \mathbf{T}^s, T^q) \mapsto \langle \mathbb{R}, \mathbb{R} \rangle^K$ to determine the constraint bounds Θ by capturing the intent implicit in $\mathbf{E}$ while generalizing to the target data T^q. The challenge is that the distribution of T^q may differ from $\mathbf{T}^s$: overly tight constraints (Example 2) can yield infeasible queries on the target data (overfitting), while overly relaxed constraints return bundles that deviate from user intent.

Section 4 describes the techniques to ensure query feasibility. Package queries avoid exhaustive search over T^q for efficiency, so the returned bundle may be suboptimal; however, our empirical results (Section 5) show the suboptimality is minimal.

4 THE EX2BUNDLE FRAMEWORK

Ex2Bundle consists of three main components as shown in Figure 5: (§4.1) *constraint synthesizer*, which encodes user intent into *feasible* constraint bounds; (§4.2) *objective synthesizer*, which encodes an application-specific quality function as the objective of the PaQL query; and (§4.3) *PaQL execution engine*, which synthesizes and executes package queries to retrieve result bundles.

Running example for focused text snippet extraction. Throughout the rest of the paper, we use a running example centered on the task of *Focused Text Snippet Extraction* (FTSE), a primary application through which we evaluate Ex2Bundle (Section 5). Because Ex2Bundle requires numeric features and text is non-numeric, we apply *contextualized topic modeling* (CTM) [5] with sentence embeddings from SBERT [67] to map sentences into a numeric vector space of topic-based features. Under the resulting topic model defined over schema $\mathbf{f}$, $f_j(t) \in [0, 1]$ denotes the relevance of a sentence t to topic f_j, and $\mathbf{f}(t)$ denotes the vector representation of t in the topic space $\mathbf{f}$. For ease of understanding, we use the following, more specific and domain-relevant terms for FTSE, instead of generic terms:

General term	FTSE term	General term	FTSE term
source/target data	source/target document	tuple	sentence
bundle	snippet or summary	schema	topic model
attribute/feature	topic	feature profile	topic profile

Example 4. *Consider a topic model* $\mathbf{f}$ *over* `Demographics`, `Geography`, *and* `Economy` *applied to documents describing U.S. states. Let* T_{UT}^s *denote the Utah document and* $E_{UT} = \{t_{UT}^1, t_{UT}^2, t_{UT}^3\} \subseteq T_{UT}^s$ *be three sentences (Table 6). Under* $\mathbf{f}$, t_{UT}^1 *maps to* $\mathbf{f}(t_{UT}^1) = [0.90, 0.00, 0.00]$, *indicating a demographics focus;* t_{UT}^2 *to* $[0.90, 0.90, 0.00]$, *covering demographics and geography; and* t_{UT}^3 *to* $[0.90, 0.30, 0.10]$, *primarily demographics with weaker geography and economy associations.*

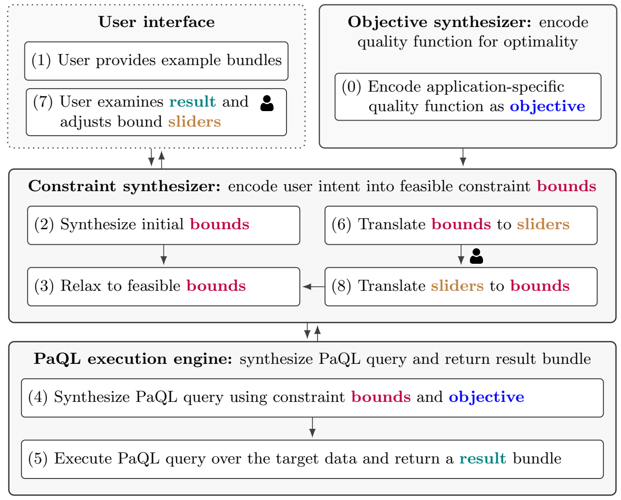

Figure 5: Ex2Bundle workflow: before end-user interaction, (0) a domain expert defines a quality function, which Ex2Bundle encodes as the objective. During use, (1) the user provides example bundles, from which (2) Ex2Bundle synthesizes initial constraint bounds and (3) relaxes them to ensure feasibility. A PaQL query is (4) formed using these bounds and the objective and (5) executed to retrieve a result bundle. For further intent refinement, (6) an interactive slider interface is generated, (7) the user inspects results and adjusts sliders, (8) Ex2Bundle maps adjustments back to bounds, and repeats (3)–(5).

Sentence/snippet	Topic score/profile (Demographics)	(Geography)	(Economy)
(t_{UT}^1) The largest ancestry groups in the state are: 26.0% English, 11.9% German, [...]	0.90	0.00	0.00
(t_{UT}^2) In comparison to all the U.S. states and territories, Utah, with a population of just over three million, is the 13th-largest by area, the 30th-most populous, and the 11th-least densely populated.	0.90	0.90	0.00
(t_{UT}^3) St. George was the fastest-growing metropolitan area in the United States from 2000 to 2005.	0.90	0.30	0.10
(E_{UT}) The largest ancestry groups in the state are: 26.0% English, 11.9% German, [...]. In comparison to all the U.S. states and territories, Utah, with a population of just over three million, is the 13th-largest by area, the 30th-most populous, and the 11th-least densely populated. St. George was the fastest-growing metropolitan area in the United States from 2000 to 2005.	2.70	1.20	0.10

Table 6: Topic scores for sentences t_{UT}^1, t_{UT}^2, and t_{UT}^3; and topic profile for the example snippet $E_{\mathrm{UT}} = \{t_{\mathrm{UT}}^1, t_{\mathrm{UT}}^2, t_{\mathrm{UT}}^3\}$.

4.1 Encoding intent into feasible constraints

The constraint synthesizer (i) synthesizes initial, potentially infeasible, bounds from user examples to form constraints, (ii) relaxes the initial bounds to ensure feasibility, (iii) exposes the feasible bounds via a slider interface for iterative refinement, and (iv) translates the adjusted sliders back to feasible bounds after refinement.

Algorithm 1: Ex2Bundle bound relaxation algorithm

Input : Initial bounds Θ_{init}, schema $\mathbf{f}$, target data T^q
Relaxation parameters μ and τ

Output : Feasible bounds Θ to construct the PaQL constraints

1 $\Theta \leftarrow \Theta_{\text{init}}$
2 #attempts $\leftarrow 0$ `// Initialize #attempts`
3 **while** $PQ(\Theta, T^q) = \emptyset$ `// While query is infeasible`
4 **do**
5 $\quad\Theta_{\text{violated}} = \text{IdentifyViolations}(\Theta, T^q)$ `// Violated constraints`
6 $\quad\Theta_{\text{relaxed}} = \emptyset$ `// Relaxed constraints`
7 $\quad\rho \leftarrow \exp\Big(\mu \cdot \max\Big(1, \Big\lfloor \frac{\#\text{attempts++}}{\tau} \Big\rfloor\Big)\Big)$ `// Relaxation step multiplier`
8 $\quad$**foreach** $\langle \texttt{lb}_j, \texttt{ub}_j \rangle \in \Theta_{violated}$ **do**
9 $\quad\quad\epsilon \leftarrow \frac{F_j(T^q)}{|T^q|}$ `// Relaxation step size`
10 $\quad\quad\texttt{lb}_j \leftarrow \max(\texttt{lb}_j - \rho\cdot\epsilon,\ \ 0)$ `// Decrease lower bound`
11 $\quad\quad\texttt{ub}_j \leftarrow \min(\texttt{ub}_j + \rho\cdot\epsilon,\ \ F_j(T^q))$ `// Increase upper bound`
12 $\quad\quad\Theta_{\text{relaxed}} = \Theta_{\text{relaxed}} \cup \{\langle \texttt{lb}_j, \texttt{ub}_j \rangle\}$
13 $\quad\Theta = (\Theta \setminus \Theta_{\text{violated}}) \cup \Theta_{\text{relaxed}}$ `// Update constraint bounds`
14 **return** Θ

4.1.1 Synthesizing the initial bounds. We extend SQuID [21], which learns user intent from example tuples, to a bundle-level granularity. Given example bundles $\mathbf{E} = \{E_i\}_{i=1}^{N}$, we use SUM as the aggregate[4] to compute their feature profiles $\{\mathbf{F}(E_i)\}_{i=1}^{N}$. We derive the initial constraint bounds $\Theta_{\text{init}} = \{\langle \texttt{lb}_j, \texttt{ub}_j \rangle\}_{j=1}^{K}$ by taking the topic-wise minimum and maximum of the feature profiles of bundles in $\mathbf{E}$:

$$\texttt{lb}_j = \min_{1\le i\le N} F_j(E_i), \quad \texttt{ub}_j = \max_{1\le i\le N} F_j(E_i)$$

Example 5. *Consider 3 example snippets E_{UT}, E_{AZ}, and E_{WA} over the source documents T^s_{UT}, T^s_{AZ}, and T^s_{WA} that represent Wikipedia pages for Utah, Arizona, and Washington, respectively. The topic profile of E_{UT}, $\mathbf{F}(E_{UT}) = \mathbf{f}(t^1_{UT}) + \mathbf{f}(t^2_{UT}) + \mathbf{f}(t^3_{UT}) = [2.70, 1.20, 0.10]$, as shown in the last row of Table 6. The topic profiles for the example snippets, along with the initial constraint bounds are given below, which results in $\Theta_{init} = \{\langle 1.50, 2.70\rangle, \langle 0.90, 1.20\rangle, \langle 0.10, 0.40\rangle\}$.*

	F_1 (*Demographics*)	F_2 (*Geography*)	F_3 (*Economy*)
$\mathbf{F}(E_{UT})$	2.70 ↑	1.20 ↑	0.10 ↓
$\mathbf{F}(E_{AZ})$	1.80	0.90 ↓	0.40 ↑
$\mathbf{F}(E_{WA})$	1.50 ↓	1.00	0.30
lb *(min)*	1.50	0.90	0.10
ub *(max)*	2.70	1.20	0.40

4.1.2 Bound relaxation to ensure feasibility. Overly tight bounds can lead to infeasible queries that return no valid bundle over the target data (Example 3), particularly when the constraint bounds are misaligned with the target data's feature distribution due to distributional shift. Infeasibility generally arises from two sources. The first is *insufficient feature coverage*, where the target data cannot satisfy a feature's lower-bound constraint even if all tuples are selected. For instance, if the source data (e.g., Utah, Arizona, & Washington) and example snippets heavily emphasize the topic `national parks` (e.g., Bryce, Grand Canyon, Olympics), but the target document is Kansas, which contains no national parks, then its maximum possible contribution to that topic is 0. Any positive lower bound (e.g., 0.10) becomes unattainable, leading to infeasibility via lower-bound violation. The second source of infeasibility is *atomicity*, which arises from the indivisibility of tuples. If selecting any single tuple already violates an upper bound, no feasible solution exists. For example, if every sentence in the target document has a topic score of at least 0.15 while the constraint's upper bound is 0.10, feasibility is impossible since tuples cannot be fractionally selected.

Bound relaxation algorithm. We address infeasibility via *bound relaxation*, which iteratively widens constraint bounds until the resulting package query becomes feasible. Algorithm 1 outlines this process, which repeats until feasibility is achieved (lines 5–13). If the initial constraint bounds Θ_{init} are infeasible (line 3), we first identify the violated constraints Θ_{violated} (line 5) using ILP solver signals (e.g., IBM CPLEX [34]) used by the package query engine (§4.3). To preserve the original intent, we selectively relax only the violated constraints, while keeping all others unchanged.

Since it is unknown whether the upper or lower bound (or both) caused the violation, we relax both bounds symmetrically. We decrease $\texttt{lb}_j$ toward 0 (line 10) and increase $\texttt{ub}_j$ toward $F_j(T^q)$ (line 11). We determine the relaxation amount at each iteration using $\rho\cdot\epsilon$ (lines 10 & 11), where the step multiplier ρ (line 7) controls the relaxation rate based on the number of prior attempts, and the step size ϵ (line 9) is a target-data-specific parameter.[5]

Relaxation step multiplier, ρ. It controls the relaxation rate and is defined as $e^{\mu\cdot\max(1,\lfloor \#\text{attempts}/\tau \rfloor)}$. Here, μ (default 0.5) sets the base relaxation rate, while τ (default 10) controls how frequently ρ is boosted. Consequently, ρ increases exponentially every τ attempts, enabling faster escape from infeasible regions.

Relaxation step size, ϵ. While ρ determines the rate of relaxation, different features exhibit varying score distributions within T^q. To account for this, we derive a feature-specific step size from the target data distribution and set $\epsilon = \frac{F_j(T^q)}{|T^q|}$, which corresponds to the average contribution of feature f_j per tuple in T^q.

<u>Remark:</u> Although our relaxation algorithm is iterative (rather than a binary search or other principled optimization method), it achieves rapid convergence in practice. We empirically found that relaxation terminates in approximately 40 iterations and takes about 4 seconds on average, even when examples contain around 100 tuples. We provide more details in §5.5 and §5.6.

Example 6. *Consider the California document T^q_{CA} over 5 sentences ($|T^q_{CA}| = 5$), where $F_2(T^q_{CA}) = 0.60$. For `Geography` (F_2), the initially synthesized constraint is $0.90 \le F_2(B) \le 1.20$ from Example 5. This is infeasible since the maximum achievable score, even when all sentences from T^q_{CA} are used to form the bundle, is 0.60, requiring relaxation. For the first 10 iterations, $\rho{=}e^{0.5\times 1}{\approx}1.65$ and $\epsilon{=}\frac{0.60}{5}{=}0.12$. Thus, the first iteration relaxes the bounds to $\max(0.90{-}1.65{\times}0.12, 0){=}0.70 \le F_2(B) \le min(1.20{+}1.65{\times}0.12, 0.60){=}0.60$. This is still infeasible, so the second iteration yields $\max(0.70{-}1.65{\times}0.12, 0){=}0.50 \le F_2(B) \le \min(0.60{+}1.65{\times}0.12, 0.60){=}0.60$, at which point the constraint becomes feasible and relaxation stops. Note that, the upper bound is not relaxed due to it already being at max.*

4.1.3 Refining intent via bound sliders. A key usability requirement (desideratum D4 in §1) for Ex2Bundle is an intuitive user interface for iterative intent refinement. A straightforward approach is to let

[4] SUM is preferred since AVG cannot distinguish single- from multi-tuple bundles, and MAX/MIN are outlier-sensitive. Our framework can support any linear aggregate.

[5] The step size ϵ is analogous to *gradient* and the step multiplier ρ to *learning rate* in stochastic gradient descent. However, constraint violation is not differentiable, since it yields zero gradients in the interior and undefined gradients at the boundary.

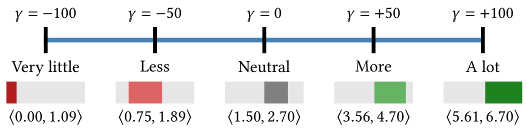

Figure 7: Slider for Demographics. Users can adjust topic emphasis from -100 (very little) to $+100$ (a lot). Each slider position maps to specific constraint bounds; the neutral position (0) corresponds to the feasible bounds synthesized (and relaxed) from user examples.

users directly edit the upper and lower bounds of the synthesized constraints. While this offers full transparency and suits experts, it reduces usability for non-expert users, who do not know how to map their intent to the corresponding constraint bounds.

Motivated by studies showing that humans are better at relative judgments (more vs. less) than absolute ones (assigning ratings) [68], we introduce a single-parameter slider [14, 22, 37, 59, 64, 73] that abstracts raw bounds (Figure 7). Each slider position corresponds to a pair of upper and lower bounds. For example, for Geography, the neutral point 0 corresponds to the feasible bounds $\langle 1.50, 2.70\rangle$ synthesized from user examples (Example 5). The slider further encodes other bounds into a scale from -100 to $+100$ around the neutral point to allow fine-grained adjustments. Upon reviewing the result bundle, users simply indicate whether they want more (less) relevance to the topic by moving the slider right (left). Beyond enabling relative adjustments, this design reduces the number of controllable parameters, consistent with human-computer interaction findings that simpler control spaces are cognitively less demanding [33].

Mapping between slider positions and bounds. We use γ_j to denote the position of the slider for topic f_j. We map the initial feasible bounds $\langle \mathtt{lb}_j, \mathtt{ub}_j\rangle$—synthesized from user examples—to the midpoint of the slider at γ_j=0. For a target document T^q, the minimum and maximum attainable bundle scores for topic f_j are 0 and $F_j(T^q)$, respectively. Let us use $\mathtt{mx}$ to denote $F_j(T^q)$ for simplicity. These two extreme values—0 and $\mathtt{mx}$—map to the lower bound of the left-most slider position and upper bound of the right-most slider position. However, a key challenge here is that both sides of the midpoint of the slider have equal number of positions (100 on each side), however, the distance between the initial feasible bounds and the two extreme bounds are not necessarily equal. Thus, for slider positions left of the midpoint, we determine the *lower* bound by interpolating based on the position's relative distance between the left-most endpoint (γ_j=−100) and the midpoint (γ_j=0). Symmetrically, for positions right of the midpoint, we determine the *upper* bound by interpolating based on the position's relative distance between the right-most endpoint (γ_j=+100) and the midpoint (γ_j=0). Assuming a constant width $\mathtt{w}$ for all bounds, the mappings are as follows:

Slider Position	Lower bound	Upper bound	γ_j
(left-most)	0	$\mathtt{w}$	-100
(left of the center)	$\mathtt{lb}_j \cdot \frac{(100-x)}{100}$	$\mathtt{lb}_j \cdot \frac{(100-x)}{100} + \mathtt{w}$	$-x$
(center)	$\mathtt{lb}_j$	$\mathtt{ub}_j$	0
(right of the center)	$\mathtt{ub}_j + \frac{(\mathtt{mx}-\mathtt{ub}_j)\cdot x}{100} - \mathtt{w}$	$\mathtt{ub}_j + \frac{(\mathtt{mx}-\mathtt{ub}_j)\cdot x}{100}$	$+x$
(right-most)	$\mathtt{mx} - \mathtt{w}$	$\mathtt{mx}$	$+100$

A question still remains: how to determine the width of the bounds $\mathtt{w}$ in the above mappings? Should it be constant for all slider positions? We use the width of the neutral position ($\mathtt{ub}_j - \mathtt{lb}_j$) as a guide to derive the bound width of slider position at γ_j=x with the formula: $\mathtt{w}(x) = e^{-\frac{\alpha\cdot|x|}{100}} \cdot (\mathtt{ub}_j - \mathtt{lb}_j)$.

Here, α is a tunable parameter between 0 and ∞, which determines how aggressively bound widths are narrowed as the slider moves farther away from the neutral position. When α=0, all slider positions use the same width as the neutral position, i.e., ($\mathtt{ub}_j - \mathtt{lb}_j$). As α increases, the width shrinks more rapidly as the slider moves away from the neutral position, resulting in increasingly tighter bounds for extreme values of γ_j. Regardless, Ex2Bundle automatically "snaps" the slider to a feasible position in case the user's adjustment leaves it at an infeasible state. Note that the bounds represented by different slider positions can be overlapping (Figure 7).

Example 7. *For feature f_1 (Demographics), the synthesized bounds from the user examples are $\langle 1.50, 2.70\rangle$ (Example 5). The maximum score along this feature is $F_1(T^q_{CA})$=6.70. Figure 7 illustrates the corresponding slider: the neutral position (γ=0) is mapped to $\langle 1.50, 2.70\rangle$. At γ=+100, the upper bound is fixed to the maximum value 6.70. With α=0.1, the width is $\mathtt{w}(+100) = e^{\frac{-0.1\cdot|100|}{100}} \cdot (2.70-1.50) = e^{-0.1} \cdot 1.20 \approx 1.09$. The lower bound is therefore $6.70-1.09 = 5.61$, yielding the bounds $\langle 5.61, 6.70\rangle$ for γ=+100. At γ=−50, the lower bound is interpolated as $1.50 \cdot \frac{(100-50)}{100} = 0.75$. The width is $\mathtt{w}(-50) = e^{\frac{-0.1\cdot|50|}{100}} \cdot 1.20 = e^{-0.05} \cdot 1.20 \approx 1.14$. Adding this width to the lower bound yields the upper bound $0.75+1.14 = 1.89$, so γ=−50 maps to $\langle 0.75, 1.89\rangle$.*

Bound relaxation after slider interaction. Since slider adjustments may re-introduce infeasibility, we re-relax the bounds afterwards using Algorithm 1. Once feasible bounds are obtained, we snap the slider to the position whose bounds are closest to the feasible bounds and subsume them, ensuring transparency to the user. Unlike the initial relaxation, we have an advantage here since we only need to relax bounds for a single constraint, with all other constraint bounds fixed. Therefore, if previously feasible bounds $\langle \mathtt{lb}^*_j, \mathtt{ub}^*_j\rangle$ are known, we use it as an additional reference during relaxation. Specifically, we reduce $\mathtt{lb}_j$ until $\max(0, \mathtt{lb}^*_j)$ (line 10 of Algorithm 1) and increase $\mathtt{ub}_j$ until $\min(\mathtt{ub}^*_j, F_j(T^q))$ (line 11). Additionally, if the user requests termination of relaxation before the process finishes, we move the slider to the closest feasible bounds.

Overall, our slider interface offers the following benefits: (1) *simplicity* by reducing the number of controls, (2) *semantic clarity* through an intuitive scale from less (negative) to more (positive), (3) *feasibility preservation* by automatically snapping user adjustments to a feasible state, and (4) *reversibility* by allowing users to reset to the original (or any intermediate) intent. Users can still adjust the bounds directly if they wish, but the slider provides a more intuitive alternative that bypasses the complex underlying numeric bounds. We evaluate the effectiveness of our design choices for the slider interface in our user study (Section 6).

4.2 Devising a quality function for optimality

As discussed in Section 3.1.2, the quality function defines the notion of optimality used to break ties among multiple valid bundles that satisfy the constraints. This becomes particularly important as bounds relaxation often yields multiple valid bundles. The quality function is application-specific and is typically specified by a domain expert prior to user interaction with an instance of Ex2Bundle, as shown in step (0) in Figure 5. Given (i) a tuple-level scoring function $\sigma : t \mapsto \mathbb{R}$, which assigns a numeric score to a tuple t, and (ii) an application-specific aggregate function $\mathcal{A}$, the quality of a candidate bundle B, $quality(B) = \mathcal{A}_{t \in B}\, \sigma(t)$. Below, we present example quality functions—along with suitable aggregators—for the three applications discussed in Section 2.

Supplier selection. *Cost* and *reliability* are common quality functions when selecting a bundle of suppliers for business expansion into a new country. For cost, a natural aggregate is SUM, with the objective of minimizing total cost, i.e., $\Sigma_{t \in B}\texttt{cost}(t)$. For reliability, suitable aggregates include AVG or MIN, depending on the preference: maximizing average reliability, or maximizing the minimum reliability among the selected suppliers.

Focused text snippet extraction. *Informativeness* and *conciseness* are common quality functions when extracting snippets or summaries from a text document. Sentence-level informativeness can be modeled using term frequency within a sentence t, weighted by inverse document frequency in the corpus (TF-IDF), which assigns higher scores to sentences containing rarer words with respect to the corpus. Alternatively, informativeness can be approximated using structural signals such as the presence of numbers, citations, or hyperlinks, which often indicate factual or reference-rich content. Conciseness can be modeled using the word count or character count of a sentence, with the objective of minimizing the total number of words or characters in a snippet. In both cases, the most appropriate aggregate is SUM. Alternatively, conciseness can be modeled by minimizing the number of sentences in a snippet, using COUNT as a bundle-level quality function that returns the snippet cardinality.

Playlist recommendation. *Popularity* and *diversity* are common quality functions in playlist recommendation. Popularity can be modeled using the number of weekly plays or chart rankings of a song, with the objective of maximizing overall popularity. A suitable aggregate in this case is AVG, i.e., $\texttt{AVG}_{t \in B}\texttt{popularity}(t)$. Diversity can be modeled using pairwise dissimilarity between songs based on features such as genre, mood, or artist characteristics. This can be captured via a distance function over song features, with the objective of maximizing diversity within the playlist. A suitable aggregate is MIN over all pairwise distances, i.e., $\texttt{MIN}_{t_i, t_j \in B, i \neq j}\texttt{distance}(t_i, t_j)$, and maximizing this ensures that even the most similar pair of songs in the playlist remains sufficiently dissimilar.[6]

4.3 Package query execution

For the package query execution engine, we use PaQL [10], a query engine for declarative package queries. PaQL translates queries into Integer Linear Programs (ILPs), which are then solved using off-the-shelf solvers such as IBM CPLEX [34]. For large target data with many tuples, solving the ILP becomes computationally expensive due to the combinatorial nature of the search space. To address this, PaQL includes the SketchRefine algorithm, a scalable method with a guaranteed $(1 + \epsilon)^6$ approximation. SketchRefine first clusters similar tuples to form a compact sketch of the data, solves the ILP over the cluster representatives, and then maps the solution back to the original tuples to construct the final bundle.

Ex2Bundle formulates a PaQL query that encodes the constraint bounds Θ (§4.1) as constraints over packages (analogous to snippets in FTSE or bundles in general) and defines the optimization objective using the scoring function σ together with the aggregate $\mathcal{A}$ (§4.2), as shown in Section 3.2. It then executes the package query, and, if feasible, returns the optimal bundle. The PaQL engine also informs the constraint synthesizer (§4.1) about any violated constraints when the query is infeasible in line 5 of Algorithm 1.

Example 8. *For the constraint bounds* $\Theta = \{\langle 1.50, 2.70\rangle, \langle 0.50, 0.60\rangle, \langle 0.10, 0.40\rangle\}$ *(derived in Example 5 and relaxed in Example 6) and the target document* T^q_{CA}*, we want the snippet with the minimum total word count. The function* $\texttt{word\_count} : t \mapsto \mathbb{N}^+$ *returns the number of words in a sentence* t*. The synthesized PaQL query is as follows:*

$\mathrm{PQ}(T^q_{CA}, \Theta)$:
```
SELECT PACKAGE(*) AS B FROM T^q_CA
SUCH THAT F_1(B) BETWEEN 1.50 AND 2.70
      AND F_2(B) BETWEEN 0.50 AND 0.60
      AND F_3(B) BETWEEN 0.10 AND 0.40
MINIMIZE SUM_{t∈B} word_count(t)
```

5 EXPERIMENTAL RESULTS

In this section, we present experimental results addressing the following research questions. **(RQ1)** Does Ex2Bundle satisfy user-specified bundle constraints, and how does this compare to constraint-agnostic baselines? (§5.2) **(RQ2)** Do Ex2Bundle's retrieved bundles align with user intent, and how does this compare to extractive and LLM-based baselines? (§5.3 & §5.4) **(RQ3)** How often is constraint relaxation triggered, and how does it scale with the number of sentences in the example snippets? (§5.5) **(RQ4)** How does Ex2Bundle's runtime scale with the number of sentences in the example snippets? (§5.6)

5.1 Experimental setup

Implementation. Ex2Bundle uses a Python backend—with Gensim for LDA (frequency-based topic modeling for FTSE), SBERT [67] for CTM (semantics-aware topic modeling), IBM CPLEX 20.1 via docplex for ILP solving, and Flask 2.0 for serving—and a JavaScript/Bootstrap frontend with an interactive slider interface.[7] We ran experiments on a 2019 MacBook Pro with a 2.4 GHz Quad-Core Intel Core i5 CPU and 16 GB of 2133 MHz LPDDR3 RAM.

Datasets. We evaluate Ex2Bundle in two applications: supplier selection over the TPC-H dataset, and focused text snippet extraction over SubSumE and CNN/DailyMail. For *TPC-H* [69], we synthesize a view from Supplier (100 rows), Partsupp (8,000 rows), and Nation (25 rows) at scale factor 0.01, with 5 attributes including price, availability, balance, region_europe, and region_america. *SubSumE* [76] contains 275 (user, intent) pairs, each with 8 manually curated summaries; we use 5 as user input and hold out the remaining 3 as a test set. *CNN/DailyMail* [30] contains news articles with human-written highlights, adapted using ChatGPT-4o to map each highlight to its closest sentences in the source article. We use a subset of 100 articles drawn from the ACL2020 split, with an average of 39 sentences per article (median 35, range 7–151). Within each category, we use 5 articles as user examples and 3 as targets, matching the train/test protocol used for SubSumE.

Baselines. Section 5.2 introduces task-specific baselines for RQ1. For RQ2–RQ4: *Top-k* selects the top-k sentences from the target document by SBERT [67] cosine similarity to the example sentences (k is the average example size). *SuDocu* [19] is our prior system for

[6]While in this paper we assume a linear, tuple-wise scoring function, Ex2Bundle can support quadratic objective via CPLEX's quadratic programming or MILP linearization.

[7]Ex2Bundle source code: https://github.com/kuangfei-long/ex2bundle

System	CSR (%) ↑	Average Objective Score ↑	Runtime (s) ↓
Random	<u>70.0</u>	10.48	**0.0005**
Greedy	33.3	**15.61**	<u>0.0024</u>
Ex2Bundle	**100.0**	<u>11.72</u>	0.0207

Table 8: Ex2Bundle achieves full constraint satisfaction with a competitive objective score. Bold = best; <u>underlined</u> = second-best.

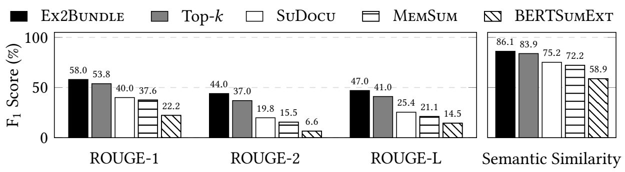

Figure 9: For focused text snippet extraction on the SubSumE dataset, Ex2Bundle outperforms all other baselines across all metrics.

focused text snippet extraction, using LDA [6] topic distributions. *BERTSumExt* [45] is a widely-adopted extractive summarizer; we adapt it for example-driven extraction by pre-filtering the document to sentences semantically similar to the examples. *MemSum* [25] is a state-of-the-art reinforcement-learning extractive summarizer for long documents; we adapt it via the same pre-filtering as BERTSumExt. *ChatGPT-4o* [54] is OpenAI's LLM; we provide the target document and example summaries via file attachment and prompt it to extract snippets.

Metrics. We use the following metrics:

- **Constraint Satisfaction Rate (CSR)**: percentage of inferred constraints satisfied by the retrieved bundle, averaged across runs.
- **Objective score**: composite of TPC-H attributes maximizing the sum of `price`, `availability`, `balance`.
- **ROUGE-1/2/L** [41]: F_1 scores measuring unigram, bigram, and longest-common-subsequence overlap between retrieved and ground-truth snippets.
- **Semantic Similarity (SS)**: cosine similarity between average SBERT embeddings of retrieved and ground-truth snippets.

More details on the setup, including additional description of the datasets and baselines, are in our technical report [13].

5.2 Constraint satisfaction performance

We evaluate constraint satisfaction in the BQbE setting for the task of supplier selection (RQ1) on the TPC-H dataset. We construct 3 example supplier bundles representing diverse procurement strategies (conservative, price-focused, balanced); Ex2Bundle infers constraints from these examples per §4.1, synthesizes a PaQL query, and executes it to retrieve a result bundle. We compare Ex2Bundle against *Greedy* (top-k tuples by objective score, ignoring constraints) and *Random* (uniform k tuples), where k is the average example bundle size, averaged over 5 runs.

As shown in Table 8, Ex2Bundle satisfies all constraints (CSR = 100%) with a competitive average objective score (11.72). Greedy achieves the highest objective score (15.61) but satisfies on average only 33.3% of the inferred constraints, indicating that objective-only optimization fails to enforce explicit constraint requirements. Random selection has a higher CSR (70.0%) than Greedy with a lower objective score (10.48), highlighting the importance of jointly optimizing both constraint satisfaction and objective.

Intent	System	Semantic Similarity ↑
Intent 1: What about this state's arts and culture attracts you the most?	ChatGPT-4o	0.64
	Ex2Bundle	**0.88**
Intent 2: What are some of the most interesting things about this state?	ChatGPT-4o	0.65
	Ex2Bundle	**0.76**

Table 10: Ex2Bundle outperforms ChatGPT-4o on both SubSumE intents; the advantage narrows on the more general one.

Key Takeaways

- Ex2Bundle achieves full constraint satisfaction with a competitive objective score.
- Greedy fails on constraint satisfaction, Random on objective score; Ex2Bundle jointly optimizes both.

5.3 Retrieval-based FTSE performance

For RQ2, we evaluate Ex2Bundle on SubSumE against Top-k, SuDocu, BERTSumExt, and MemSum. We report ROUGE-1/2/L F_1 scores and SBERT-based semantic similarity (SS) between retrieved and ground-truth snippets, together with per-query runtime.

Figure 9 shows that Ex2Bundle consistently outperforms all baselines on ROUGE and semantic similarity. SuDocu, which is example-driven, outperforms the pure extractive summarizers MemSum and BERTSumExt (which lack example-driven intent modeling), but its topic-modeling approach is inferior to Ex2Bundle and Top-k, both of which use semantics-aware topic modeling.

Key Takeaways

- Ex2Bundle outperforms all baselines on ROUGE and SS.
- Systems that use semantics-aware topic modeling (Ex2Bundle) and Top-k outperform SuDocu and pure extractive summarization (BERTSumExt, MemSum).

5.4 LLM-based summarization performance

Ex2Bundle is LLM-free: its latency scales with corpus rather than model size, it applies to private data where LLM pipelines cannot run, and it incurs no per-query inference cost. The tradeoff is potential accuracy loss; for RQ2, we test this against ChatGPT-4o on two settings: *focused intents* from SubSumE (specific needs) and *generic intents* from CNN/DailyMail (open-ended summaries).

Focused intent. We compare Ex2Bundle against ChatGPT-4o (RAG, few-shot) on two intents from SubSumE—one more specific, one less so—to see how each tool handles different levels of intent specificity (Table 10). Ex2Bundle outperforms ChatGPT-4o on both (SS=0.88 vs 0.64 for the more specific Intent 1; 0.76 vs 0.65 for the less specific Intent 2). The advantage narrows on the less specific intent—Ex2Bundle drops from 0.88 to 0.76 while ChatGPT-4o stays roughly flat (0.64 to 0.65)—but Ex2Bundle leads in both.

Generic intent. On CNN/DailyMail, we compare against ChatGPT-4o (few-shot) across four categories (Politics, Crime, Sports, & Lifestyle). ChatGPT-4o leads on all ROUGE and SS metrics in every category (Table 11); on Politics, for instance, ChatGPT-4o achieves ROUGE-L 0.4814 and SS 0.7965, vs. 0.2335 and 0.6272 for Ex2Bundle (similar gaps elsewhere). This is expected: generic news snippets have broad intent that example-driven retrieval cannot capture [80], marking the regime where Ex2Bundle is not the appropriate tool.

Category	System	ROUGE-1↑	ROUGE-2↑	ROUGE-L↑	Semantic Similarity↑
Politics	ChatGPT-4o	**0.5504**	**0.4403**	**0.4814**	**0.7965**
	Ex2Bundle	0.3235	0.1590	0.2335	0.6272
Crime	ChatGPT-4o	**0.4844**	**0.3513**	**0.4019**	**0.7601**
	Ex2Bundle	0.3246	0.1499	0.2204	0.5829
Sports	ChatGPT-4o	**0.5170**	**0.4219**	**0.4548**	**0.8022**
	Ex2Bundle	0.3148	0.1743	0.2330	0.6255
Lifestyle	ChatGPT-4o	**0.5060**	**0.3812**	**0.4294**	**0.7949**
	Ex2Bundle	0.3136	0.1304	0.1982	0.6016

Table 11: ChatGPT-4o outperforms Ex2Bundle in generic intent alignment across CNN/DailyMail news categories.

Key Takeaways

- Ex2Bundle outperforms ChatGPT-4o on focused intents.
- The advantage narrows on less specific focused intents.
- On generic extraction, ChatGPT-4o leads, marking the regime where example-driven retrieval is less effective.

5.5 Relaxation analysis

We analyze relaxation behavior (RQ3) on 50 target documents varying example sizes from 5 to 45 sentences. For each query, Ex2Bundle synthesizes constraints from the example bundles (§4.1) and tries to retrieve a bundle from the target document satisfying those constraints; if no feasible solution exists, Ex2Bundle iteratively relaxes the violated constraints until feasibility is reached (§4.1.2). About 17% of queries are initially infeasible, independent of example size, indicating that infeasibility is driven by the match between target content and example-derived bounds rather than by the size of examples. For infeasible queries, Figure 12 shows the distribution of relaxation iterations needed to reach feasibility, which grows with example size: more example sentences yield tighter bounds, requiring more relaxation iterations to reach feasibility.

Key Takeaways

- About 17% of queries require constraint relaxation, independent of example size.
- More example sentences yield tighter constraint bounds, requiring more relaxation iterations to reach feasibility.

5.6 Scalability analysis

We evaluate Ex2Bundle's scalability with respect to example-bundle complexity in terms of their size, i.e., the number of sentences k (RQ4). Since scalability in target document size is inherited from the PaQL framework [10], we focus on the example complexity. On CNN/DailyMail, we vary the example size k from 5 to 100 sentences in increments of 5, using 5 source documents and 45 target documents per setting. We synthesize examples via three sampling strategies: *single-topic* (top-k sentences for one randomly selected topic), *multi-topic* (iteratively sampling topics at random until k sentences are collected), and *random* (uniform random k sentences from the document). We report *learning time* (average time to compute bounds and synthesize the PaQL query from examples) and *retrieval time* (average time to execute the PaQL query and retrieve the snippet from the target document).

Figure 13 shows that learning time (left) dominates total time (right), exceeding retrieval time (center) across all strategies, since

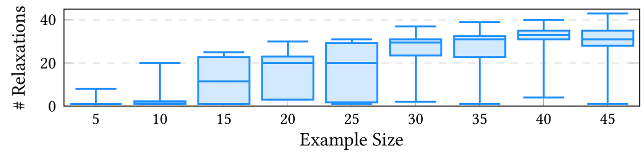

Figure 12: Relaxation Analysis for Ex2Bundle across varying size of examples. The number of relaxations triggered increases with the number of example sentences.

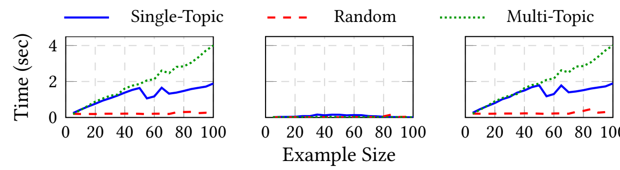

Figure 13: Average learning (left), retrieval (center), and total times (right) while varying the example sizes in the example snippets for single-topic, random, and multi-topic sampling strategies.

computing bounds is more expensive than PaQL execution. The completion time remains within 5 seconds even at 100-sentence examples, with total time scaling linearly. Among strategies, multi-topic has the highest learning time (it computes bounds across a more diverse snippet), while random has the lowest (treating selection like independent Top-k rather than a combinatorial bundle).

Key Takeaways

- Learning time dominates total time across all sampling strategies; bound computation dominates the runtime.
- Ex2Bundle runs in under 5 seconds even for 100-sentence example snippets, with total time scaling linearly.

Experiments evaluating the impact of quality function variants on retrieval performance, prompt details, and extensive comparisons against baselines are in our technical report [13].

6 INTENT SLIDERS: A USER STUDY

To evaluate the effectiveness of the intent-refinement slider interface of Ex2Bundle for FTSE, we conducted a within-subjects study with 20 participants recruited via Amazon Mechanical Turk, fluent in English and experienced with document search. Participants were tasked with identifying their preferred US state for relocation using Wikipedia articles, comparing Ex2Bundle against SBERT (Top-k baseline, §5.3), a widely-used embedding model in vector databases that outperforms other baselines in our evaluation. We exclude non-BQbE baselines (e.g., conversational LLMs), which underperform on FTSE (Table 10). Slider parameter α was set to 0.1. Following a mixed-methods approach [24], participants were randomly assigned to start with either tool (anonymized) to mitigate transfer effects. Each participant was assigned to 25 randomly selected states per tool and was instructed to provide snippets for at least 2 states with no upper limit. While interacting with Ex2Bundle, they could select sentences, search using keywords, and refine intents by adjusting sliders. In SBERT, they could select sentences and search using keywords as well, but intent adjustment had to be done via manual addition/removal of sentences from example snippets.

We quantitatively analyzed the data by comparing the participants' ratings of usefulness, ease of use, and daily usage preference for both SBERT and Ex2Bundle. We also compared the number of interactions with the interface (e.g., selecting examples, adjusting sliders, requesting snippet retrieval, etc.) between the two tools to understand how participants engaged with them.

Participants' satisfaction. On task completion, participants rated each tool on usefulness, ease of use, and daily usage preference. Ex2Bundle received higher ratings than SBERT on all three: (i) 12/20 rated Ex2Bundle more useful (8/20 for SBERT), (ii) 16/20 found Ex2Bundle easier to use (only 4/20 for SBERT), and (iii) 11/20 preferred Ex2Bundle for daily use (9/20 for SBERT).

Participants' interactions. To understand how participants interacted with the tools, we categorized interactions into six types: *Search*: keyword queries, *Selection*: example selection, *Update*: example modification, *Learn*: intent learning, *Retrieval*: backend retrieval, and *Slider*: interactive parameters. Averaging interaction counts per user, SBERT users performed about 3.1 times more *Search* interactions than Ex2Bundle users. In contrast, Ex2Bundle users had more *Slider* interactions and consequently about 2.1 times more *Retrieval* interactions, as Ex2Bundle automatically retrieves bundles upon slider adjustments. Ex2Bundle's interactive sliders encouraged more iterative refinement of results, while SBERT's lack of interactivity led to more manual searching.

Participants provided feedback on their experience with both tools. The majority used the interactive sliders to refine their intents: *"I raised the slider up so that the output summaries would prioritize information that pertained to the economy and climate of each state since those were the most important factors for me."* However, some preferred the simplicity of SBERT, while others noted difficulty giving the system examples, highlighting a limitation of the example-based approach: *"I found the original text hard to focus on due to being one large text block [...] trying to find stuff like food and cuisine didn't necessarily pick up on things without knowing exact keywords."* [8]

7 RELATED WORK

Programming by-example [27, 28] is a paradigm where users provide examples to express their intent and has seen significant success in databases, particularly in data discovery-by-example [38, 50, 57] and query-by-example (QbE) [8, 15, 20, 21, 56, 60, 82], which allows users to express their query intent through examples rather than formal query languages. QPlain [15] incorporates explanations with data provenance, some works prune the search space for efficient query synthesis [8, 60] or generalize the output [56], and recently SQuID [20, 21] semantically synthesizes SQL queries from user-provided examples rather than relying on structural similarity. However, all existing QbE systems are designed to retrieve individual items and do not account for the combinatorial nature of bundle retrieval. Ex2Bundle addresses the bundle retrieval problem through the lens of QbE, where users express their intent through example bundles, as opposed to example tuples.

Focused text snippet extraction, also known as query-focused extractive summarization, allows users to specify keywords or queries to guide snippet selection [36, 40, 42], but requires users to explicitly formulate their intents. RAG-based approaches [2, 29, 39, 43] can also support snippet extraction via few-shot prompting [9], but are not designed for extracting bundles that jointly satisfy multiple constraints. Recently, *faceted query-by-example* approaches leverage representative documents as an example query, additionally conditioned on explicit facets (constraints) to extract snippets across multiple constraints [16, 51, 71]. However, existing systems either require users to specify constraints as keywords (e.g., `Method` in scientific articles) [16] or as natural language queries [71], or rely on opaque vector matching that provides no interpretable constraint values [51]. Moreover, because these baselines score snippets independently, their objective functions fail to capture the joint constraint satisfaction required for bundle retrieval.

Bundle retrieval is a fundamental task in database that involves selecting an optimal set of items subject to a collection of constraints [10, 12, 17, 49, 72]. It has also been studied extensively in recommender systems, where the goal is to generate a bundle that collectively satisfies user preferences across various domains, such as e-commerce [44, 65, 74], entertainment [11, 55], and travel planning [18, 26, 79]. However, these systems rely on greedy heuristics or approximate generative models to generate bundles and struggle to enforce strict constraints. In relational databases, prior work on package queries addresses the efficiency of bundle query performance [10, 49], but assumes users can already express a well-formed query. Diversification-based retrieval approaches [12, 17, 72] also retrieve bundles that collectively satisfy user constraints. However, they typically operate over a limited set of constraints such as cardinality or diversity scores. In contrast, Ex2Bundle addresses the upstream challenge of automatically synthesizing multi-constraint package queries from user-provided examples.

8 SUMMARY AND FUTURE DIRECTIONS

We introduced Ex2Bundle, an example-driven framework for bundle retrieval that enables users to specify intents through example bundles. Ex2Bundle synthesizes package queries from examples and employs principled constraint relaxation to ensure feasibility. Our experiments and user study show that Ex2Bundle captures user intent and outperforms other baselines in intent alignment.

Several directions extend the framework. Manual example selection is burdensome, especially for complex intents or large data; an adaptive example recommendation that infers preference patterns from prior selections would lower this barrier. The current relaxation widens bounds symmetrically once violations are detected; more principled policies, such as minimal or preference-weighted relaxation, could better preserve user intent. Ex2Bundle currently optimizes linear objectives over per-tuple scores; native PaQL support for pairwise or quadratic objectives (e.g., bundle diversity for playlists) would broaden expressiveness. Specification of the quality function requires domain expertise; inferring both objective and constraints from example bundles—a problem related to inverse optimization [31]—would remove the domain-expertise requirement. Inference relies on example bundles alone; when examples are few or fail to convey aspects of intent that are better specified using natural-language modifiers such as "like these examples but with stronger emphasis on diversity", a hybrid example-driven inference with LLM-based reasoning under symbolic soundness guarantees would extend coverage while preserving bound verifiability.

[8]More details on the user study are in our technical report [13].

REFERENCES

[1] 2020. Discover Weekly keeps giving me the same genre that I'm completely sick of. https://community.spotify.com/t5/Your-Library/Discover-Weekly-keeps-giving-me-the-same-genre-that-I-m/td-p/5065068.

[2] 2026. LangChain. https://github.com/langchain-ai/langchain Accessed: 2026-01-07.

[3] 2026. Recommendations: Figuring out how to bring unique joy to each member. https://research.netflix.com/research-area/recommendations.

[4] Jinze Bai, Chang Zhou, Junshuai Song, Xiaoru Qu, Weiting An, Zhao Li, and Jun Gao. 2019. Personalized Bundle List Recommendation. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. ACM, 60–71.

[5] Federico Bianchi, Silvia Terragni, Dirk Hovy, Debora Nozza, and Elisabetta Fersini. 2021. Cross-lingual Contextualized Topic Models with Zero-shot Learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Association for Computational Linguistics, Online, 1676–1683.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022.

[7] Jay Bonggolto. 2025. Google is testing a new 'Daily Discover' feed on YouTube Music. https://tech.yahoo.com/streaming/articles/google-testing-daily-discover-feed-170300058.html.

[8] Angela Bonifati, Radu Ciucanu, and Slawek Staworko. 2016. Learning Join Queries from User Examples. *ACM Trans. Database Syst.* 40, 4 (2016), 24:1–24:38.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[10] Matteo Brucato, Juan Felipe Beltran, Azza Abouzied, and Alexandra Meliou. 2016. Scalable Package Queries in Relational Database Systems. *Proc. VLDB Endow.* 9, 7 (2016), 576–587.

[11] Jianxin Chang, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2023. Bundle Recommendation and Generation With Graph Neural Networks. *IEEE Trans. Knowl. Data Eng.* 35, 3 (2023), 2326–2340.

[12] Whanhee Cho and Anna Fariha. 2026. Data-Semantics-Aware Recommendation of Diverse Pivot Tables. *Proc. ACM Manag. Data* 4, 1, Article 23 (April 2026), 28 pages.

[13] Whanhee Cho, Kuangfei Long, Mahmood Jasim, Matteo Brucato, Alexandra Meliou, Peter J. Haas, and Anna Fariha. 2026. Example-Driven Intent Synthesis for Constrained Data Bundle Retrieval: Focused Text Snippet Extraction and Beyond (Technical Report). *Preprint* (2026). https://users.cs.utah.edu/~afariha/ex2bundle_tech_rep.pdf

[14] John Joon Young Chung and Eytan Adar. 2023. PromptPaint: Steering Text-to-Image Generation Through Paint Medium-like Interactions. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 6, 17 pages.

[15] Daniel Deutch and Amir Gilad. 2016. QPlain: Query by explanation. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society, 1358–1361.

[16] Heejin Do, Sangwon Ryu, Jonghwi Kim, and Gary Lee. 2025. Multi-Facet Blending for Faceted Query-by-Example Retrieval. In *ACL*. 28577–28590.

[17] Marina Drosou and Evaggelia Pitoura. 2012. DisC diversity: result diversification based on dissimilarity and coverage. *Proc. VLDB Endow.* 6, 1 (2012), 13–24.

[18] Shih Hsin Fang, Eric Hsueh-Chan Lu, and Vincent S. Tseng. 2014. Trip Recommendation with Multiple User Constraints by Integrating Point-of-Interests and Travel Packages. In *IEEE 15th International Conference on Mobile Data Management, MDM 2014, Brisbane, Australia, July 14-18, 2014 - Volume 1*. IEEE Computer Society, 33–42.

[19] Anna Fariha, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. 2020. SuDocu: Summarizing Documents by Example. *Proc. VLDB Endow.* 13, 12 (2020), 2861–2864.

[20] Anna Fariha, Lucy Cousins, Narges Mahyar, and Alexandra Meliou. 2026. Example-driven semantic-similarity-aware query intent discovery: Empowering users to cross the SQL barrier through query by example. *Inf. Syst.* 138 (2026), 102687.

[21] Anna Fariha and Alexandra Meliou. 2019. Example-Driven Query Intent Discovery: Abductive Reasoning using Semantic Similarity. *Proc. VLDB Endow.* 12, 11 (2019), 1262–1275.

[22] Rohit Gandikota, Joanna Materzynska, Tingrui Zhou, Antonio Torralba, and David Bau. 2024. Concept Sliders: LoRA Adaptors for Precise Control in Diffusion Models. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XL (Lecture Notes in Computer Science)*. Springer, 172–188.

[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 (2024), 1132–1145.

[24] Jennifer C Greene, Valerie J Caracelli, and Wendy F Graham. 1989. Toward a conceptual framework for mixed-method evaluation designs. *Educational evaluation and policy analysis* 11, 3 (1989), 255–274.

[25] Nianlong Gu, Elliott Ash, and Richard H. R. Hahnloser. 2022. MemSum: Extractive Summarization of Long Documents Using Multi-Step Episodic Markov Decision Processes. (2022), 6507–6522. https://doi.org/10.18653/V1/2022.ACL-LONG.450

[26] Qi Gu, Jian Cao, and Yancen Liu. 2022. CSBR: A Compositional Semantics-Based Service Bundle Recommendation Approach for Mashup Development. *IEEE Trans. Serv. Comput.* 15, 6 (2022), 3170–3183.

[27] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*. ACM, 317–330.

[28] Sumit Gulwani and Prateek Jain. 2017. Programming by Examples: PL Meets ML. In *Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings (Lecture Notes in Computer Science)*, Vol. 10695. Springer, 3–20.

[29] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Retrieval Augmented Language Model Pre-Training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 3929–3938.

[30] Karl Moritz Hermann, Tomás Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *NeurIPS*. 1693–1701.

[31] Clemens Heuberger. 2004. Inverse Combinatorial Optimization: A Survey on Problems, Methods, and Results. *Journal of Combinatorial Optimization* 8, 3 (2004), 329–361.

[32] Po Hu, Donghong Ji, Chong Teng, and Yujing Guo. 2012. Context-Enhanced Personalized Social Summarization. In *Proceedings of COLING 2012*. The COLING 2012 Organizing Committee, Mumbai, India, 1223–1238.

[33] Andy Hunt, Marcelo M. Wanderley, and Matthew Paradis. 2002. The Importance of Parameter Mapping in Electronic Instrument Design. In *New Interfaces for Musical Expression, NIME-02, Proceedings, Dublin, Ireland, May 24-26, 2002*. Media Lab Europe, 149–154.

[34] IBM ILOG CPLEX Optimization Studio [n.d.]. IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/docs/en/icos.

[35] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques. In *SIGMOD*. ACM, 277–281.

[36] Hal Daumé III and Daniel Marcu. 2006. Bayesian Query-Focused Summarization. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*, Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle (Eds.). The Association for Computer Linguistics. https://doi.org/10.3115/1220175.1220214

[37] Rahul Jain, Amit Goel, Koichiro Niinuma, and Aakar Gupta. 2025. AdaptiveSliders: User-aligned Semantic Slider-based Editing of Text-to-Image Model Output. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 541, 27 pages.

[38] Akash Khatri, Mahathir Mohammad, and El Kindi Rezig. 2025. Sort it Like You Mean It: Discovering Semantically Interesting Attribute Augmentations to Sort Tables. *Proc. VLDB Endow.* 18, 12 (2025), 5427–5430. https://doi.org/10.14778/3750601.3750688

[39] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[40] Yanran Li and Sujian Li. 2014. Query-focused Multi-Document Summarization: Combining a Topic Model with Graph-based Semi-supervised Learning. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin City University and Association for Computational Linguistics, Dublin, Ireland, 1197–1207.

[41] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[42] Marina Litvak and Natalia Vanetik. 2017. Query-based summarization using MDL principle. In *Proceedings of the multiling 2017 workshop on summarization and summary evaluation across source types and genres*. 22–31.

[43] Jerry Liu. 2022. *LlamaIndex*. https://github.com/jerryjliu/llama_index

[44] Xiaohao Liu, Jie Wu, Zhulin Tao, Yunshan Ma, Yinwei Wei, and Tat-Seng Chua. 2025. Fine-tuning Multimodal Large Language Models for Product Bundling. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.1, KDD 2025, Toronto, ON, Canada, August 3-7, 2025*. ACM, 848–858.

[45] Yang Liu and Mirella Lapata. 2019. Text Summarization with Pretrained Encoders. (2019), 3728–3738. https://doi.org/10.18653/V1/D19-1387

[46] Xuan Lu, Sifan Liu, Bochao Yin, Yongqi Li, Xinghao Chen, Hui Su, Yaohui Jin, Wenjun Zeng, and Xiaoyu Shen. 2025. MultiConIR: Towards Multi-Condition Information Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, Suzhou, China, 13471–13494.

[47] Gang Luo. 2006. Efficient Detection of Empty-Result Queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*. ACM, 1015–1025.

[48] Gang Luo. 2006. Efficient Detection of Empty-Result Queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*. ACM, 1015–1025.

[49] Anh L. Mai, Pengyu Wang, Azza Abouzied, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. 2024. Scaling Package Queries to a Billion Tuples via Hierarchical Partitioning and Customized Optimization. *Proc. VLDB Endow.* 17, 5 (2024), 1146–1158.

[50] Mir Mahathir Mohammad and El Kindi Rezig. 2026. Qualitative Join Discovery in Data Lakes using Examples. *Proc. ACM Manag. Data* 4, 1, Article 68 (April 2026), 28 pages. https://doi.org/10.1145/3786682

[51] Sheshera Mysore, Arman Cohan, and Tom Hope. 2022. Multi-Vector Models with Textual Guidance for Fine-Grained Scientific Document Similarity. 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, 4453–4470.

[52] Arnab Nandi and H. V. Jagadish. 2011. Guided Interaction: Rethinking the Query-Result Paradigm. *Proc. VLDB Endow.* 4, 12 (2011), 1466–1469.

[53] Ani Nenkova, Sameer Maskey, and Yang Liu. 2011. Automatic Summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA - Tutorial Abstracts*. The Association for Computer Linguistics, 3.

[54] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[55] Apurva Pathak, Kshitiz Gupta, and Julian J. McAuley. 2017. Generating and Personalizing Bundle Recommendations on *Steam*. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*. ACM, 1073–1076.

[56] Fotis Psallidas, Bolin Ding, Kaushik Chakrabarti, and Surajit Chaudhuri. 2015. S4: Top-k Spreadsheet-Style Search for Query Discovery. In *SIGMOD*. ACM, 2001–2016.

[57] El Kindi Rezig, Anshul Bhandari, Anna Fariha, Benjamin Price, Allan Vanterpool, Vijay Gadepally, and Michael Stonebraker. 2021. DICE: Data Discovery by Example. *Proc. VLDB Endow.* 14, 12 (2021), 2819–2822.

[58] Thibault Sellam and Martin L. Kersten. 2013. Meet Charles, big data query advisor. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org.

[59] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*. ACM, 365–377.

[60] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. 2014. Discovering queries based on example tuples. In *SIGMOD*. ACM, 493–504.

[61] Yunxiao Shi, Haoning Shang, Xing Zi, Wujiang Xu, Yue Feng, and Min Xu. 2025. Answering Narrative-Driven Recommendation Queries via a Retrieve–Rank Paradigm and the OCG-Agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Suzhou, China, 13181–13202.

[62] Aviv Slobodkin, Niv Nachum, Shmuel Amar, Ori Shapira, and Ido Dagan. 2023. SummHelper: Collaborative Human-Computer Summarization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023 - System Demonstrations, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 554–565.

[63] Spotify Advertising Team. 2020. Five years of discovery and engagement through Discover Weekly. https://ads.spotify.com/en-US/news-and-insights/five-years-of-discovery-and-engagement-through-discover-weekly/. Accessed: 2026-03-17.

[64] Hendrik Strobelt, Albert Webson, Victor Sanh, Benjamin Hoover, Johanna Beyer, Hanspeter Pfister, and Alexander M. Rush. 2023. Interactive and Visual Prompt Engineering for Ad-hoc Task Adaptation with Large Language Models. *IEEE Trans. Vis. Comput. Graph.* 29, 1 (2023), 1146–1156.

[65] Wenqi Sun, Ruobing Xie, Junjie Zhang, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2023. Generative Next-Basket Recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems, RecSys 2023, Singapore, Singapore, September 18-22, 2023*. ACM, 737–743.

[66] Hiroaki Takatsu, Takahiro Kashikawa, Koichi Kimura, Ryota Ando, and Yoichi Matsuyama. 2021. Personalized Extractive Summarization Using an Ising Machine Towards Real-time Generation of Efficient and Coherent Dialogue Scenarios. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*. Association for Computational Linguistics, Online, 16–29.

[67] Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. 2021. Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 296–310.

[68] Louis L. Thurstone. 1974. *A Law of Comparative Judgment* (1st ed.). Routledge. 12 pages.

[69] Transaction Processing Performance Council (TPC). [n.d.]. *TPC-H: Decision Support Benchmark*. Technical Report. TPC. http://www.tpc.org/tpch/

[70] Joseph Tso, Preston Schmittou, Quan Huynh, and Jibran Hutchins. 2026. ConstraintBench: Benchmarking LLM Constraint Reasoning on Direct Optimization. *arXiv preprint arXiv:2602.22465* (2026).

[71] Jianyou Wang, Kaicheng Wang, Xiaoyue Wang, Prudhviraj Naidu, Leon Bergen, and Ramamohan Paturi. 2023. DORIS-MAE: scientific document retrieval using multi-level aspect-based queries. , Article 1668 (2023), 16 pages.

[72] Yue Wang, Alexandra Meliou, and Gerome Miklau. 2018. RC-Index: Diversifying Answers to Range Queries. *Proc. VLDB Endow.* 11, 7 (2018), 773–786.

[73] Zhijie Wang, Yuheng Huang, Da Song, Lei Ma, and Tianyi Zhang. 2024. PromptCharm: Text-to-Image Generation through Multi-modal Prompting and Refinement. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*. ACM, 185:1–185:21.

[74] Penghui Wei, Shaoguo Liu, Xuanhua Yang, Liang Wang, and Bo Zheng. 2022. Towards Personalized Bundle Creative Generation with Contrastive Non-Autoregressive Decoding. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*. ACM, 2634–2638.

[75] Wen Xiao and Giuseppe Carenini. 2019. Extractive Summarization of Long Documents by Combining Global and Local Context. In *EMNLP-IJCNLP*. Association for Computational Linguistics, 3009–3019.

[76] Nishant Yadav, Matteo Brucato, Anna Fariha, Oscar Youngquist, Julian Killingback, Alexandra Meliou, and Peter Haas. 2021. SubSumE: A Dataset for Subjective Summary Extraction from Wikipedia Documents. In *New Frontiers in Summarization Workshop*. https://summarization2021.github.io

[77] Rui Yan, Jian-Yun Nie, and Xiaoming Li. 2011. Summarize What You Are Interested In: An Optimization Framework for Interactive Personalized Summarization. In *EMNLP*. ACL, 1342–1351.

[78] Hamed Zamani, Bhaskar Mitra, Everest Chen, Gord Lueck, Fernando Diaz, Paul N. Bennett, Nick Craswell, and Susan T. Dumais. 2020. Analyzing and Learning from User Interactions for Search Clarification. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. ACM, 1181–1190.

[79] Jiale Zhang, Mingqian Ma, Xiaofeng Gao, and Guihai Chen. 2024. Encoder-Decoder Based Route Generation Model for Flexible Travel Recommendation. *IEEE Trans. Serv. Comput.* 17, 3 (2024), 905–920.

[80] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen R. McKeown, and Tatsunori B. Hashimoto. 2024. Benchmarking Large Language Models for News Summarization. *Trans. Assoc. Comput. Linguistics* 12 (2024), 39–57.

[81] Siyan Zhao, Mingyi Hong, Yang Liu, Devamanyu Hazarika, and Kaixiang Lin. 2025. Do LLMs Recognize Your Preferences? Evaluating Personalized Preference Following in LLMs. In *ICLR*.

[82] Moshé M. Zloof. 1975. Query-by-Example: the Invocation and Definition of Tables and Forms. In *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*. ACM, 1–24.