FROM DATA LAKES TO TRAINING TASKS: GROUNDED AND CONTROLLABLE SYNTHESIS FOR DATA AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training data agents at scale rests on a steady supply of grounded, verifiable data-analysis tasks. Real-world data lakes make these hard to produce: they hold far more files than can be inspected individually, and the few files that share a meaningful join path cannot be identified from local context alone. We propose DATA2TASK, which represents a data pool as a compact *physical–semantic dual-view graph*: its physical view replaces exhaustive inspection with selective access, while its semantic view aligns columns denoting the same concept across files and so surfaces latent relations. DATA2TASK then samples a provenance-preserving subgraph and instantiates a compatible analytical operator over it, forming a *task motif* whose answer is computable from the sampled files and therefore grounded by construction; a staged pipeline turns each motif into a natural-language request with executable ground truth, kept only when independent solvers reproduce that answer, and scored separately for structural and semantic difficulty. With only 2,000 verified tasks and selected solution trajectories, fine-tuning Qwen3-8B achieves an average score of 55.17% across four held-out data-analysis benchmarks, outperforming the 29× larger Qwen3-235B and matching the 60× larger Qwen3-Coder-480B on TableBench.

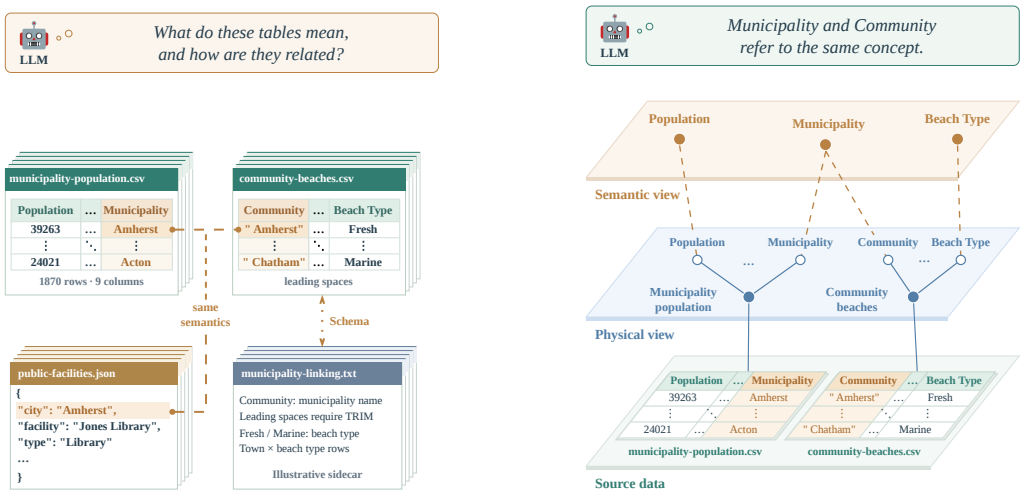
1 INTRODUCTION

Background and motivation. Data agents—LLM-based systems that turn natural-language intent into tool-supported analytical workflows and refine them through execution feedback—lower the expertise that data analysis demands (Sun et al., 2025; Weng et al., 2025; Zhu et al., 2025) and have emerged across text-to-SQL (Gao et al., 2024; Wang et al., 2025), tabular and semi-structured analysis (Zhang et al., 2025; Yi et al., 2025), business-intelligence reporting (Cecchi & Babkin, 2024; Weng et al., 2025), and end-to-end data-science pipelines (Guo et al., 2024; Hong et al., 2025; Luo et al., 2026). Despite differences in implementation, a data agent pairs a *foundation LLM* with a *data-analysis harness* that grounds it in the data. Trajectories and error signals refine both sides, and capabilities first implemented in the harness are progressively internalized by the model (Li et al., 2026; Ning et al., 2026; Chen et al., 2026). Both directions consume grounded, verifiable tasks, so scalable task generation is a central bottleneck.

Challenges and existing approaches. Unlike task synthesis from text corpora, data-analysis tasks must be grounded in the data itself: a task has to read as a realistic analytical query anchored in actual file content, not as an abstract specification to be coded (Huang et al., 2024; Ouyang et al., 2025; Lai et al., 2025; Gu et al., 2024). Real data lakes make this hard twice over. They are large—thousands of files and millions of records—so exhaustive inspection is impractical (Deng et al., 2024; Lei et al., 2024); and valid cross-file relations are rare, with the cues that reveal them buried among irrelevant columns, noisy values, and undocumented conventions (Cafarella et al., 2008) (Figure 1(a)).

Existing task construction follows three strategies. **(1) Direct generation from real data** exposes an LLM to file contents or metadata: DataMind works from headers, column names, and sampled rows (Qiao et al., 2026), which cannot establish whether columns in different tables are semantically equivalent and value-compatible, while DSGym has an agent explore the pool (Nie et al., 2026), where a finite budget and sparse relations may leave related files never observed together. **(2) Manual and hybrid curation** has experts select sources, write questions, and produce reference solutions; the tasks are grounded, but annotation cost caps the scale that repeated training needs (Lai

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) real-world data lake **(b)** physical–semantic dual-view graph

Figure 1: From a real-world data lake (a) to a physical–semantic dual-view graph (b). In (a), `Municipality`, `Community`, and `city` name the same towns under different headers, `Community` values carry leading spaces, and the conventions for reading them sit in a separate file. In (b), the physical view (blue) links tables to their columns, and the semantic view (orange) aligns `Municipality` and `Community` to one concept, exposing a join between town populations and beach records.

et al., 2025; Hu et al., 2024; Sun et al., 2026; Zhang et al., 2026a;b). **(3) Joint synthesis of data and tasks**, as in OmniSQL and Datarus-R1 (Li et al., 2025; Ben Chaliah & Dellagi, 2025), makes every task grounded by construction, but in synthetic artifacts that need not reproduce the irregular values, latent relations, and domain conventions of real data. Each strategy thus gives up one of sparse-relation discovery, cost, and real data; none secures all three.

Our method. To this end, we formalize grounded task synthesis over a data pool (Section 2) and propose DATA2TASK, which replaces raw file contents with a compact intermediate representation of the data pool (Section 3.2): a dual-view graph (Figure 1(b)) whose nodes are file clusters, tables, columns, and global concepts. Its *physical* view connects files with compatible schemas and exposes their tables and columns, so a model can reach the relevant subset of a large, heterogeneous lake without the exhaustive file inspection that its scale rules out; its *semantic* view links columns that share an underlying meaning across tables, surfacing the latent relations that local context alone cannot reveal, so the rarity of valid cross-file relations no longer bounds what can be discovered.

On top of this representation, DATA2TASK builds a task in two stages. It first samples a provenance-preserving subgraph—a single table, a file series, or a multi-table join tree—then instantiates a compatible analytical operator over it, yielding a task motif whose answer is computable from the sampled files and therefore grounded in real data by construction (Section 3.3). Separating the subgraph from the operator lets source structure and analytical complexity be varied independently. A comprehend–design–solve–verify–rewrite pipeline then turns the motif into a natural-language request with executable ground truth (Section 3.4), and a task is retained only when independent solvers, told which columns the task was built on, reproduce that answer—correctness without manual annotation, which is what lets generation reach volumes expert curation cannot.

Difficulty is scored on two axes (Section 3.5): structural, read off the motif, and semantic, the interpretation the query demands. Unlike a single score that grows with the number of composed operations (Qiao et al., 2026), this lets a training set be filtered and stratified by either kind of demand.

Using 2,000 verified tasks paired with selected solution trajectories, we fine-tune Qwen3-8B and evaluate it on four held-out data-analysis benchmarks (Section 4). The resulting model attains a mean score of 55.17%, outperforming both DataMind-7B and DataMind-14B, which are likewise trained on generated tasks. It also surpasses Qwen3-235B in mean score and reaches parity with Qwen3-Coder-480B on TableBench, although these models are 29× and 60× larger, respectively.

2 PROBLEM FORMULATION

Overview. We are given a *data pool* $\mathcal{D}$ of raw, heterogeneous files, partitioned by subject area into m *domains* $\mathcal{D} = \bigcup_{r=1}^{m} \mathcal{D}_r$, and declaring neither keys nor relations among its files. Our goal is to synthesize from $\mathcal{D}$ alone a set of data-analysis tasks $\mathcal{T} = \bigcup_{r=1}^{m} \mathcal{T}_r$, where $\mathcal{T}_r$ collects the tasks scoped to domain $\mathcal{D}_r$. Each task $t = (q, \mathcal{D}_r, g) \in \mathcal{T}_r$ pairs a natural-language analytical query q with an executable ground truth g over the files of domain r; a solver receives only q and $\mathcal{D}_r$, so locating the relevant files and recovering their relations is itself part of the task.

2.1 DATA-ANALYSIS TASKS

Query, context, and ground truth. The query q is a natural-language analytical request that states what to compute without listing the sources a solution must read; the context is the task's entire domain $\mathcal{D}_r$, supplied read-only. The ground truth $g = (p, a)$ pairs a *reference program* p with its answer $a = \mathrm{Exec}(p; \mathcal{D}_r)$, where Exec runs p in a fixed sandbox without network access so that a is reproducible. Executing p also exposes the *source set* $\mathcal{D}_r^\star(t) \subseteq \mathcal{D}_r$ of files it reads—typically $|\mathcal{D}_r^\star(t)| \ll |\mathcal{D}_r|$—which we keep as metadata and never include in the task.

Correctness. A *solver* S is a possibly randomized procedure that observes $(q, \mathcal{D}_r)$, may run its own code, and returns one answer $\hat{a}$ in the format that q itself prescribes—its expected type, unit, and precision—so correctness is exact equality $\hat{a} = a$ and grading ignores how the answer was obtained.

2.2 TASK DESIDERATA

To keep only tasks worth training on, we call $t \in \mathcal{T}_r$ *admissible* when it has three properties. **1. Grounded.** Since $a = \mathrm{Exec}(p; \mathcal{D}_r)$, the answer is fixed by the data rather than asserted by a model; it must also be non-empty and not obtainable without reading $\mathcal{D}_r$. **2. Verified.** At least $m \geq 2$ of k verification solvers, each told which columns t was built on, answer t correctly—that is, q determines a once its sources are known. **3. Nontrivial.** A *difficulty profile* $d(t) = (d_{\mathrm{str}}(t), d_{\mathrm{sem}}(t))$ scores structure (sources combined and chain length, read off its motif) and semantics (how far q is from transcribing that computation); their sum must clear a floor, which excludes single-value lookups and queries that spell out their own computation. Either can also stratify tasks (Section 3.5).

We formulate the problem of controllable grounded task synthesis as follows.

Problem 1 (Controllable grounded task synthesis)**.** *Let $\mathcal{K}$ be a set of analytical categories, each fixing the kind of computation a task performs. Given a data pool $\mathcal{D}$, k verification solvers, a requested number n_κ of tasks per category $\kappa \in \mathcal{K}$, and a synthesis budget B, measured in generator and verifier model calls, construct a task set $\mathcal{T}$ of maximal size such that* (C1) *every $t \in \mathcal{T}$ is admissible (Section 2.2),* (C2) *at most n_κ tasks fall in category κ, and* (C3) $\mathrm{cost}(\mathcal{T}) \leq B$

3 GROUNDED AND CONTROLLABLE TASK SYNTHESIS

DATA2TASK solves Problem 1 in five stages (Figure 2): parsing files into tables (Section 3.1), building the dual-view graph $\mathcal{G}$ (Section 3.2), sampling motifs from $\mathcal{G}$ under per-category quotas (Section 3.3), generating and verifying tasks (Section 3.4), and filtering them by difficulty (Section 3.5). LLMs only interpret the data and phrase the requests; the graph decides which sources and operator a task uses, and execution decides its answer. Tasks are thus grounded in the files they use, verified by independent solvers, and nontrivial after filtering, as (C1) requires; the quotas enforce (C2), and a fixed candidate budget with bounded model calls per candidate enforces (C3).

3.1 SOURCE PREPARATION

This stage turns heterogeneous files into tables with named columns that record their source file and block. For each group of similar files, an LLM emits a declarative plan configuring one of a fixed set of parser templates, which deterministic code then executes. The persisted plan is re-executed in the sandbox at task time, so a solution reads the same tables the graph was built from, making $a = \mathrm{Exec}(p; \mathcal{D}_r)$ reproducible (Appendix D.1).

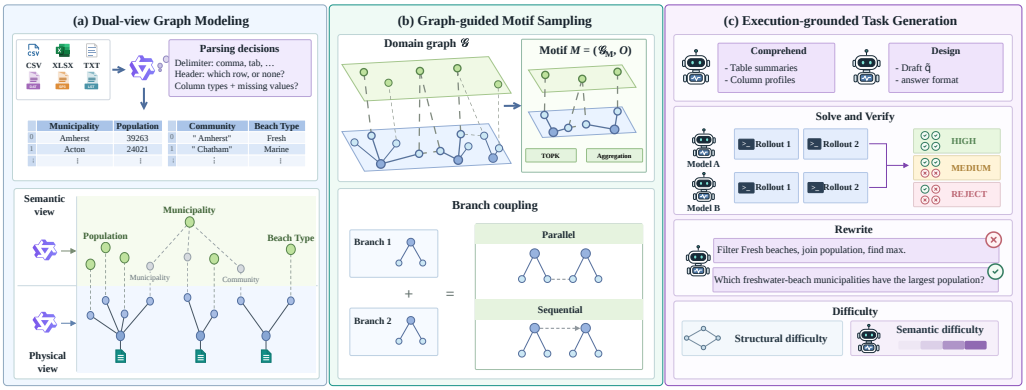


Figure 2: Overview of DATA2TASK. (a) Files are parsed into tables and organized into a dual-view graph. (b) Motifs $M = (\mathcal{G}_M, O)$ are sampled from the graph alone, without LLM calls or file access. (c) Each motif becomes a query with execution-verified ground truth, which is then scored for difficulty and filtered.

3.2 DUAL-VIEW GRAPH CONSTRUCTION

Over each domain we build a typed, attributed graph $\mathcal{G} = (V, E)$ with four kinds of node: CLUSTER nodes group source files, TABLE nodes are parsed tables, FIELD nodes are columns or column groups within a table, and CONCEPT nodes are the real-world entities and quantities that fields refer to, such as *Municipality* or *Population*; fields in different tables that refer to the same one share a concept node. Its relation types fall into two groups, which give the graph its two views. The *physical view* $\mathcal{R}_{\text{phy}} = \{\text{CONTAINS}, \text{HASFIELD}, \text{MEASURE}\}$ organizes sources and their local analytical structure; the *semantic view* $\mathcal{R}_{\text{sem}} = \{\text{INSTANCEOF}, \text{PARTITIONEDBY}, \text{FILESERIES}, \text{LINK}\}$ aligns concepts and establishes cross-source relations. The two views share their source nodes, so together they form a single representation of the data pool.

3.2.1 PHYSICAL VIEW CONSTRUCTION

Hierarchy construction. Parsed tables are organized into a cluster–table–field hierarchy connected by CONTAINS and HASFIELD edges. Clusters record file-group membership and are marked as partitioned series or heterogeneous groups; tables record their source file and block, and fields their source columns.

Field annotation. Given column profiles and table context, an LLM assigns each field an analytical role—KEY, DIMENSION, MEASURE, TIME, GEO, or META—a semantic type family, and a *concept hint*, a short description of its meaning that seeds concept alignment. Columns that jointly express one variable, such as latitude and longitude, are merged into a single grouped field.

Virtual fields. Some structure lives in filenames rather than columns: `weather_2023.csv` and `weather_2024.csv` are annual slices, yet no field records the year. For a partitioned cluster, we extract the tokens that vary across its filenames and add them to each member table as a constant *virtual field*, with a time role for years and dates and a dimension role otherwise, unless an existing field already records them. This field is the cluster's *partition axis*.

Within-table relations. MEASURE edges connect every DIMENSION, TIME, or GEO field to every MEASURE field in the same table and record the axis kind (categorical, ordinal-time, or geographic), marking where grouping and aggregation apply; in Figure 2a, for example, `Municipality`→`Population` marks municipality as the axis along which population can be selected or aggregated.

3.2.2 SEMANTIC VIEW CONSTRUCTION

Local annotations cannot tell whether fields in different tables mean the same thing, yet the rare cross-file relations a task needs hinge on exactly that. The semantic view resolves fields into shared concepts and derives cross-source relations from them (Figure 2a).

Global concept alignment. We resolve fields into CONCEPT nodes from coarse to fine. Within buckets of compatible roles, fields with the same concept hint are first merged directly. Candidate matches among the remaining groups are then screened by name overlap, by value overlap for keys and dimensions, and by type compatibility for measures, times, and geographic fields; these tests only propose candidates. An LLM partitions each set of connected candidates into distinct concepts, and a second pass reconciles same-named groups that fell into different role buckets. Partition axes are matched to existing concepts by a separate LLM step, or given new ones. In Figure 2a, for example, `Population.Municipality` and `Beaches.Community` resolve to one *Municipality* concept despite their different names. Appendix D gives the screening criteria, thresholds, and prompts.

Semantic edge construction. From the resolved concepts, four edge types follow by fixed rules, without further LLM calls. INSTANCEOF links each non-meta field to its concept. PARTITIONEDBY links a partitioned cluster to the concept of its partition axis, and FILESERIES links it to each measure concept present in a majority of its member tables. LINK joins fields in different tables that share a concept and have compatible roles, typed as equi or fuzzy for keys and dimensions, spatial for geographic fields, and temporal for shared time axes.

The graph thus decouples relation discovery from task synthesis. Every cross-file relation is materialized in $\mathcal{G}$ as an explicit edge before any task is written, and is recorded together with its source provenance and the evidence under which it was admitted (Appendix D). Subsequent stages obtain these relations by querying the graph rather than re-inferring them from raw file contents, so every relation a task relies on can be traced back and inspected.

3.3 GRAPH SAMPLING

Given a domain graph $\mathcal{G}$, we sample grounding subgraphs $\mathcal{G}_M \subseteq \mathcal{G}$ and bind compatible analytical operations to form motifs $M = (\mathcal{G}_M, O)$ (Figure 2b). Each motif is a basic unit for downstream task generation: $\mathcal{G}_M$ brings together the selected clusters, tables, fields, and their supporting concepts and relations, while O specifies the analytical structure over these sources. Since $\mathcal{G}_M$ keeps the provenance of every node and every operand of O is bound to a field of $\mathcal{G}_M$, any task instantiated from M is answerable from the sampled files alone, which secures the grounded property of (C1). Each subgraph is paired with every compatible operator family, so source structure and analytical complexity vary independently (Appendix E). Construction uses graph attributes without LLM calls or raw-file access; concrete requests and execution-grounded answers are produced in the next stage.

Subgraph enumeration. We enumerate subgraphs in three forms: a single table with its within-table relations, a cluster with its member tables, and table- or cluster-centered subgraphs joined by eligible LINK edges. Each subgraph is summarized by an *input signature* $\sigma(\mathcal{G}_M)$ recording what it offers an operator—keys, measures, grouping and time axes, a partition axis, and cross-source links—read directly off its edges and annotations. For the population and beaches tables of Figure 2b, linked through the shared *Municipality* concept, σ records `Population` as a measure, `Municipality` and `Community` as key fields, and an equi relation. Optional transformations π, such as filtering, binning, or deriving a column, can enrich the signature to $\sigma' = T_\pi(\sigma(\mathcal{G}_M))$ and so enable analyses the raw inputs do not support.

Operator matching. An operator family ω specifies input requirements, an output type, and an expansion into core operations, and is eligible when $\mathrm{requires}_\omega(\sigma') = 1$: AGG, for instance, needs a measure, and TOPK additionally a grouping dimension or key. The catalog organizes its families into nine analytical categories: aggregation, counting, ranking, time-based calculation, correlation analysis, statistical analysis, distribution analysis, anomaly detection, and cross-source linking. Binding a matched family to fields of $\mathcal{G}_M$ yields a single-branch plan O; task-specific values such as filter thresholds are left to task generation.

Branch coupling. To require more than one analysis, a coupler joins two branches through a relation in $\mathcal{G}$: CORRELATE correlates two time series aligned by a temporal link, DERIVE takes the difference of two scalar results on distinct measures with compatible units, and PIPE uses an entity found by one branch to select, via a partition axis, the file the other branch analyzes.

Finally, each motif is assigned to an analytical category by its operator family, or by its coupler if coupled, and the motif budget is split as evenly as possible across categories, which sets the

target quotas n_κ of (C2). This discourages common graph structures from dominating the candidate set, although sparse categories and later verification may change the final counts. Signature fields, transformations, coupling constraints, and the quota rule are detailed in Appendix E.

3.4 Task Generation

Each motif $M = (\mathcal{G}_M, O)$ is turned into a query q over the files $\mathcal{D}_r$ of its domain, with ground truth $g = (p, a)$, in four steps (Figure 2c). LLMs read the data and write the query; the answer comes only from executing solver code on the files.

Comprehend. We load the tables of M, compute column statistics, and detect data-quality issues. From these, the schemas, and the source metadata, an LLM writes table summaries and column profiles that describe field meanings, value characteristics, and required cleaning.

Design. Given M and these summaries, an LLM follows O to choose entities, filter conditions, and parameters, and writes a query $\tilde{q}$ with its expected answer type and comparison tolerance. Values named in $\tilde{q}$ are checked against the source tables, and mismatches are returned for revision.

Solve and verify. Given $\tilde{q}$, $\mathcal{D}_r$, and the columns the task was built on, two agents driven by different models each run two independent rollouts, writing and executing code on $\mathcal{D}_r$. Supplying the columns lets verification check the answer rather than the search for data, which is left to the agent trained on the task. After discarding failed runs and invalid outputs, we compare answers under the specified tolerance: cross-model agreement gives *high* confidence, agreement between the two rollouts of only one model gives *medium* confidence, and anything else is rejected. Unresolved candidates are retried within a fixed budget. The agreed answer becomes a, and the code of one agreeing rollout becomes p.

Rewrite. So that it reads as a natural request, an LLM rewrites $\tilde{q}$ into q, removing unnecessary file and table names and recasting step-by-step instructions as an analytical goal, while keeping the conditions that identify the data and fix the answer. We then solve q again in the same way and keep the task only if its answer still matches a; the two rounds together secure the verified property of (C1). Synthesis cost is bounded in advance: the motif budget (Section 3.3) fixes the number of candidates, and fixed retry limits bound the generator and verifier calls per candidate, so total cost is at most their product, which enforces (C3).

3.5 Difficulty Assessment and Filtering

We score each verified task on two axes (Figure 2c): structural difficulty, computed by fixed rules from the motif, and semantic difficulty, judged by an LLM. Scoring happens after verification and does not affect which motifs are generated; it decides which tasks are kept and how the retained set is stratified.

Structural difficulty. Structural difficulty measures the computation a task requires:

$$d_{\text{str}} = w_{\text{chain}} x_{\text{chain}} + w_{\text{link}} x_{\text{link}} + w_{\text{op}} x_{\text{op}} + w_{\text{dirty}} x_{\text{dirty}} + w_{\text{couple}} x_{\text{couple}}, \tag{1}$$

where x_{chain} is the operation-chain length plus transformation depth, x_{link} the cross-table linking complexity, $x_{\text{op}} \in \{0, \ldots, 4\}$ the tier of the operator family, x_{couple} the number of couplings, and x_{dirty} indicates whether the task requires data cleaning; the w are nonnegative weights. All features except x_{dirty} are read off the motif, so structural difficulty directly reflects the subgraph, operators, and couplings chosen in Section 3.3. Appendix F gives the feature definitions and operator tiers.

Semantic difficulty. Semantic difficulty $d_{\text{sem}} \in \{0, \ldots, 4\}$ measures how much interpretation q requires beyond the computation itself: disambiguating fields or entities, resolving business definitions, and applying domain knowledge. After verification, a separate LLM judge rates the final query without access to the data or solution.

Difficulty profile and filtering. A task's difficulty profile is the pair $(d_{\text{str}}, d_{\text{sem}})$, aggregated into $s = d_{\text{str}} + d_{\text{sem}}$. Tasks whose score falls below a floor are discarded as trivial, which secures the nontrivial property of (C1); thresholds on s sort the rest into bands. Both components are kept alongside s, so a training set can also be stratified by structural or semantic demand alone.

Table 1: **Main data-analysis evaluation.** We report pass@1 and pass@3 (%) on four held-out data-analysis benchmarks, together with the average pass@1 across the four benchmarks. **Bold** highlights DATA2TASK-8B and the Qwen3-Coder-480B TableBench reference. Scores are rounded to two decimals.

Model / Agent	DAEval		DABStep		QRData-V		TableBench		Avg.
	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3	
Large Open-weight Reference Models									
DeepSeek-V4-Pro	89.14	92.82	44.82	59.28	62.42	72.11	73.10	81.30	67.37
Kimi-K2.5	89.87	92.82	36.24	45.86	61.62	72.51	77.91	83.95	66.41
GLM-5.2	90.79	93.92	56.68	61.79	61.22	71.71	76.77	83.13	71.36
Qwen3-Coder-480B	87.48	92.82	27.22	40.04	53.65	68.53	**68.02**	**77.85**	59.09
Qwen3-235B	76.43	91.16	14.91	23.27	53.92	72.91	61.65	76.63	51.73
Open-weight Data-analysis Agents									
DataMind-14B	83.61	92.82	16.81	23.56	56.44	70.12	14.74	23.99	42.90
DataMind-7B	77.16	88.40	13.65	18.97	45.67	59.84	17.31	26.82	38.45
Our Models									
Qwen3-8B	67.96	84.53	12.01	15.88	46.48	62.15	55.89	72.36	45.59
DATA2TASK-8B	**84.16**	**92.82**	**18.57**	**23.27**	**49.94**	**70.52**	**68.02**	**80.90**	**55.17**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Setup. Our source pool contains 5,666 raw real-world files: 3,872 Kaggle-derived files from DataMind (Qiao et al., 2026) and 1,794 files from KramaBench (Lai et al., 2025); questions, answers, and trajectories are excluded. The student is Qwen3-8B. For each verified task, GLM-5.2 generates three candidate trajectories, from which we select one correct trajectory using execution-reliability and efficiency criteria (Appendix C). We evaluate on four held-out benchmarks—DAEval, DABStep (Egg et al., 2025), QRData (Liu et al., 2024), and TableBench (Wu et al., 2025)—using the DSGym-curated versions of the first three (Nie et al., 2026), and report pass@1 and pass@3, with mean pass@1 as the aggregate metric. Baselines comprise the untuned backbone, released task-specific data-analysis agents, and large open-weight models. Appendix Sections B.1–B.4 provide the full data-pool, benchmark, baseline, and training configurations.

4.2 MAIN RESULTS

Performance. As shown in Table 1, fine-tuning Qwen3-8B on DATA2TASK consistently improves performance across all four benchmarks, with pass@1 gains of 16.20 and 12.13 points on DAEval and TableBench, respectively. The resulting 8B model outperforms the substantially larger Qwen3-235B on three benchmarks and approaches the overall performance of Qwen3-Coder-480B, which has 60$\times$ more parameters. Compared with DataMind, another data-analysis agent trained on synthesized tasks, DATA2TASK-8B outperforms DataMind-7B across all four benchmarks and exceeds DataMind-14B on three. These results indicate that synthesizing training tasks from cross-table relations discovered in real heterogeneous data provides effective supervision and substantially strengthens data-analysis capabilities.

Data Efficiency. DATA2TASK uses only 2K selected trajectories with standard supervised fine-tuning. In comparison, DataMind uses 11.7K training trajectories together with SFT and reinforcement learning. Despite using substantially less data and a simpler training recipe, DATA2TASK-8B remains competitive with these data-analysis agents and improves its backbone across all four benchmarks, covering general data handling, multi-step analysis, statistical and causal reasoning, and table reasoning. These consistent gains across heterogeneous benchmarks show that DATA2TASK can improve general data-analysis capabilities with standard supervised fine-tuning alone.

Table 3: **Effect of training-task difficulty.** Each variant fine-tunes Qwen3-8B on 300 trajectories. The mixed set contains 100 easy, 100 medium, and 100 hard trajectories. We report DAEval, DABStep, and QRData pass@1 and pass@3 (%), together with the average pass@1 across the three benchmarks.

Training set	DAEval		DABStep		QRData		Avg.
	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3	
Easy	71.46	88.40	12.98	18.79	44.75	64.54	43.06
Medium	73.66	89.50	13.42	19.91	46.61	67.33	44.56
Hard	**80.11**	**91.16**	13.35	19.02	**48.07**	**69.72**	**47.18**
Mixed	77.53	90.06	**14.47**	**21.25**	44.09	63.75	45.36

4.3 Analysis of Synthesized Task Quality

We further analyze the properties of the synthesized training tasks. Our goal is to understand whether DATA2TASK can transform heterogeneous data lakes into training supervision that is not only automatically generated, but also diverse, grounded, and controllable. Across the 2,000 synthesized tasks, DATA2TASK covers all 18 domains and all nine analytical categories; Appendix G.2 provides the detailed distribution and analysis. We focus here on cross-table grounding and difficulty-based controllability.

4.3.1 Cross-table Relation Discovery

Table 2: **Cross-table task prevalence.** We compare the proportion of cross-table tasks across DataMind and DATA2TASK.

Method	Input	Tasks	Cross-table
DataMind	SQLite	4,383	3,333 (76.0%)
	CSV	6,676	0 (0.0%)
	XLSX	618	0 (0.0%)
	All	11,677	3,333 (28.5%)
DATA2TASK	File lake	2,000	**1,518 (75.9%)**

We analyze whether DATA2TASK can recover latent cross-table relations from heterogeneous file lakes. Table 2 compares its cross-table task coverage with DataMind across different input types.

DataMind produces cross-table tasks only from SQLite databases, where relational structure is already provided, while none are generated from standalone CSV or XLSX files. In contrast, DATA2TASK achieves a 75.9% cross-table rate directly from independently organized files, comparable to DataMind's SQLite setting. This result indicates that the physical–semantic dual-view graph can uncover latent relations across files and convert them into multi-source training tasks without relying on a pre-existing database schema.

4.3.2 Difficulty Characterization and Training Utility

The synthesized tasks span a broad range of difficulty levels, with medium tasks forming the largest group, followed by easy and hard tasks. Hard tasks are less frequent because they typically require richer cross-table structure and more demanding semantic reasoning.

To assess whether these difficulty levels correspond to different training utility, we construct matched training sets containing only easy, medium, or hard tasks, together with a mixed set, while keeping all other training and evaluation settings fixed.

Table 3 shows that harder tasks generally provide stronger supervision: hard-only training achieves the highest average pass@1 and performs best on DAEval and QRData, while the mixed set performs best on DABStep. These results indicate that task difficulty captures meaningful variation in training utility, while mixed-difficulty supervision can remain beneficial for some multi-step analysis settings. We therefore retain a mixed difficulty distribution in the main experiments.

4.4 Ablation Studies

Ablation of the Dual-View Graph. We isolate the role of the physical–semantic dual-view graph by removing it from the generation pipeline. In this variant, the generator receives the raw files of each domain $\mathcal{D}_r$ directly, rather than a sampled graph motif. This removes the graph-derived

structural and semantic guidance while preserving access to the same underlying source data. The generator, source domains, and verification procedure are otherwise unchanged. We then apply the same difficulty scorer to the retained tasks and use their executed reference programs to determine whether each task reads more than one table.

To measure downstream utility under a matched training budget, we additionally generate 2,000 tasks with this graph-free variant and convert them into 2,000 high-quality trajectories using the same rollout generation, trajectory selection, and SFT recipe as in the main experiment.

Table 4: **Effect of the dual-view representation.** Downstream results use 2,000 selected trajectories for both variants.

Distribution	w/o graph	DATA2TASK
Source-table composition		
1 table	35.1%	24.1%
2 tables	64.9%	56.6%
$\geq$ 3 tables	0.0%	**19.4%**
Cross-table ($\geq$ 2)	64.9%	**75.9%**
Difficulty distribution		
Easy	37.7%	31.8%
Medium	62.0%	50.6%
Hard	0.3%	**17.7%**
Downstream utility (2K trajectories)		
DAEval (%)	81.77	**84.16**

Table 4 shows that removing the dual-view graph substantially reduces task complexity. Without the graph, cross-table tasks are largely limited to two-table combinations, while tasks involving three or more tables disappear and hard tasks become almost absent. Under the matched 2,000-trajectory setting, removing the graph lowers DAEval pass@1 from 84.16% to 81.77%, a drop of 2.39 percentage points. This suggests that the greater coverage of cross-table and difficult tasks enabled by the graph provides more effective supervision for improving data-analysis capabilities. Nevertheless, the graph-free variant still substantially outperforms the 67.96% untuned backbone, confirming that the overall task-generation and trajectory-selection pipeline remains effective even without the graph.

Appendix G.1 provides an additional ablation of the trajectory-selection strategy.

5 Related Work

Recent work synthesizes training tasks for data-analysis agents through LLM-based generation coupled with execution- or consistency-based verification (Hu et al., 2024; Tang et al., 2025). DataMind (Qiao et al., 2026) composes progressively more difficult tasks according to a predefined taxonomy, whereas DSGym (Nie et al., 2026) allows an agent to explore a data pool and generate its own questions. Datarus-R1 (Ben Chaliah & Dellagi, 2025) instead constructs synthetic analysis challenges programmatically. Together, these approaches demonstrate the value of synthetic supervision, but they generally do not begin by constructing an explicit global representation of the relations within an underlying real-world data pool.

Related work in text-to-SQL offers finer control over operator coverage and query complexity (Li et al., 2025; Caferoğlu et al., 2025; Duan et al., 2025), but typically assumes structured database schemas and SQL outputs. In contrast, DATA2TASK first discovers cross-file relations in heterogeneous real-world data pools through an explicit physical–semantic representation, then synthesizes grounded, executable tasks with controllable structural and semantic properties. Appendix A provides a more detailed discussion of related work.

6 Conclusion and Limitations

We presented DATA2TASK, a framework for synthesizing grounded and verifiable training tasks from heterogeneous real-world data lakes. DATA2TASK uses a physical–semantic dual-view graph to enable selective access to large data collections and expose latent cross-file relations, then converts graph-grounded structures into natural-language tasks with executable ground truth. Experiments across held-out data-analysis benchmarks show that supervision synthesized in this way consistently improves data-analysis performance, highlighting real-world data lakes as a promising source of scalable training supervision for data agents.

A remaining limitation is the scalability of graph construction. As data lakes grow, cross-source relation discovery becomes increasingly expensive. Future work could reduce this cost through more efficient candidate retrieval, hierarchical indexing, and incremental graph construction.

AI USE STATEMENT

In this work, we used generative AI tools for research ideation and execution, including refining the conceptual framework and methodology, assisting with implementation and debugging, designing experiments, and interpreting experimental results. We also used generative AI to generate the synthetic task dataset studied in this paper. In particular, generative models are components of DATA2TASK and are used for source preparation, field annotation and semantic alignment, task instantiation and rewriting, semantic difficulty assessment, execution-based solution generation, and training trajectory generation.

We did not use generative AI to formulate or prove mathematical claims or to conduct qualitative or thematic analysis; the remaining required-disclosure activities are not applicable to this work.

Additionally, we used generative AI tools for retrieval and discovery of potentially relevant literature, summarizing materials during literature exploration, and editing the manuscript for structure, clarity, and language.

We reviewed all AI-assisted work. Suggested references were checked against the original publications, AI-assisted code was reviewed and tested, and reported experimental results were verified against the corresponding outputs. Synthetic tasks and answers were retained only after the validation, independent execution, consensus, and re-verification procedures described in the paper. AI-generated suggestions were reviewed by the authors, who made all final methodological, experimental, and writing decisions. We take responsibility for the final content of this work, including text, claims, code, data, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the synthesized task set used in our experiments in the Supplementary Material. We organize the paper and appendix to support reproduction of both the task synthesis procedure and the downstream training results. Sections 2 and 3 define the task formulation and describe source preparation, dual-view graph construction, motif sampling, task generation and verification, and difficulty assessment. Section 4 specifies the data sources, held-out benchmarks, evaluation metrics, baselines, and ablations. Appendix B documents the data-pool and leakage audit, benchmark and evaluation protocol, baseline selection, complete SFT configuration, and measured compute. Appendix C documents trajectory generation, correctness filtering, and best-of-three selection. Appendices D–F provide implementation details for parsing and replay, graph construction, operator compatibility, quotas and deduplication, and difficulty calibration.

REFERENCES

Ayoub Ben Chaliah and Hela Dellagi. Datarus-R1: An adaptive multi-step reasoning LLM for automated data analysis. *arXiv preprint arXiv:2508.13382*, 2025. URL https://arxiv.org/abs/2508.13382.

Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. WebTables: Exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008. doi: 10.14778/1453856.1453916. URL https://dl.acm.org/doi/10.14778/1453856.1453916.

Hasan Alp Caferoğlu, Mehmet Serhat Çelik, and Özgür Ulusoy. SING-SQL: A synthetic data generation framework for in-domain text-to-SQL translation. *arXiv preprint arXiv:2509.25672*, 2025. URL https://arxiv.org/abs/2509.25672.

Lucas Cecchi and Petr Babkin. ReportGPT: Human-in-the-loop verifiable table-to-text generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 529–537. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-industry.39. URL https://aclanthology.org/2024.emnlp-industry.39/.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-Harness: Co-evolving harnesses and model weights for LLM agents. *arXiv preprint*

arXiv:2607.22688, 2026. doi: 10.48550/arXiv.2607.22688. URL https://arxiv.org/abs/2607.22688.

Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-sql: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 2166–2187. International Committee on Computational Linguistics, October 2022. URL https://aclanthology.org/2022.coling-1.190/.

Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, Kaisen Jin, Chi Zhang, Yuqing Jiang, Yuanfang Zhang, Yuping Wang, Ye Yuan, Guoren Wang, and Nan Tang. LakeBench: A benchmark for discovering joinable and unionable tables in data lakes. *Proceedings of the VLDB Endowment*, 17(8):1925–1938, 2024. doi: 10.14778/3659437.3659448. URL https://dl.acm.org/doi/10.14778/3659437.3659448.

Shaoming Duan, Youxuan Wu, Chuanyi Liu, Yuhao Zhang, Zirui Wang, Peiyi Han, Shengyuan Yu, Liang Yan, and Yingwei Liang. DSQG-Syn: Synthesizing high-quality data for text-to-SQL parsing by domain specific question generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025. doi: 10.18653/v1/2025.findings-naacl.162. URL https://aclanthology.org/2025.findings-naacl.162/.

Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. DABstep: Data agent benchmark for multi-step reasoning. *arXiv preprint arXiv:2506.23719*, 2025. doi: 10.48550/arXiv.2506.23719. URL https://arxiv.org/abs/2506.23719.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145, 2024. doi: 10.14778/3641204.3641221. URL https://doi.org/10.14778/3641204.3641221.

Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M. Drucker. How do analysts understand and verify AI-assisted data analyses? In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 748:1–748:22. Association for Computing Machinery, 2024. doi: 10.1145/3613904.3642497. URL https://doi.org/10.1145/3613904.3642497.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16813–16848. PMLR, 2024. URL https://proceedings.mlr.press/v235/guo24b.html.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin Wu. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821. Association for Computational Linguistics, July 2025. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. InfiAgent-DABench: Evaluating agents on data analysis tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 19544–19572. PMLR, 2024. URL https://proceedings.mlr.press/v235/hu24s.html.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. DA-code: Agent data science code generation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 13487–13521. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.748. URL https://aclanthology.org/2024.emnlp-main.748/.

Myunghoon Kang, Dahyun Jung, Suhyune Son, Seonmin Koo, Changwoo Chun, Daniel Rim, Haeyoung Kwon, Yuna Hur, and Heuiseok Lim. EASE: Entity-aware sub-table generation for real-world multi-table QA. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. doi: 10.18653/v1/2026.acl-long.10. URL https://aclanthology.org/2026.acl-long.10/.

George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32:905–936, 2023. doi: 10.1007/s00778-022-00776-8. URL https://doi.org/10.1007/s00778-022-00776-8.

Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Michael J. Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. KramaBench: A benchmark for AI systems on data-to-insight pipelines over data lakes. *arXiv preprint arXiv:2506.06541*, 2025. doi: 10.48550/arXiv.2506.06541. URL https://arxiv.org/abs/2506.06541.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. *arXiv preprint arXiv:2411.07763*, 2024. doi: 10.48550/arXiv.2411.07763. URL https://arxiv.org/abs/2411.07763.

Fangyu Lei, Jinxiang Meng, Yiming Huang, Tinghong Chen, Yun Zhang, Shizhu He, Jun Zhao, and Kang Liu. Reasoning-table: Exploring reinforcement learning for table reasoning, 2025. URL https://arxiv.org/abs/2506.01710.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. OmniSQL: Synthesizing high-quality text-to-SQL data at scale. *Proceedings of the VLDB Endowment*, 18, 2025. doi: 10.14778/3749646.3749723. URL https://dl.acm.org/doi/10.14778/3749646.3749723.

Jinzhe Li, Yuan Wu, and Yi Chang. Harness engineering for LLM agents: A survey of harness component taxonomy, evaluation, and model–harness coevolution. *Preprints*, 2026. URL https://www.preprints.org/manuscript/202606.2203. Version 1, posted June 30, 2026.

Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. Are LLMs capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9215–9235, 2024. doi: 10.18653/v1/2024.findings-acl.548. URL https://aclanthology.org/2024.findings-acl.548/.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 37(10):5735–5754, 2025. doi: 10.1109/TKDE.2025.3592032. URL https://doi.org/10.1109/TKDE.2025.3592032.

Yuyu Luo, Guoliang Li, Ju Fan, and Nan Tang. Data agents: Levels, state of the art, and open problems. In *Companion of the 2026 International Conference on Management of Data*, pp. 571–579. Association for Computing Machinery, 2026. doi: 10.1145/3788853.3801878. URL https://doi.org/10.1145/3788853.3801878.

Fan Nie, Junlin Wang, Harper Hua, Federico Bianchi, Yongchan Kwon, Zhenting Qi, Owen Queen, Shang Zhu, and James Zou. DSGym: A holistic framework for evaluating and training data science agents. *arXiv preprint arXiv:2601.16344*, 2026. doi: 10.48550/arXiv.2601.16344. URL https://arxiv.org/abs/2601.16344.

Xuying Ning, Dongqi Fu, Tianxin Wei, Hanqing Zeng, Yuanchen Bei, Bingxuan Li, Zihao Li, Qifan Wang, Xiang Shen, Yifan Wu, Jiayi Liu, Hong Li, Yinglong Xia, Xiangjun Fan, Hanghang Tong, and Jingrui He. EvoHarness-RL: Learning self-evolving runtime harness for long-horizon LLM agents. *arXiv preprint arXiv:2608.05446*, 2026. doi: 10.48550/arXiv.2608.05446. URL https://arxiv.org/abs/2608.05446.

Shuyin Ouyang, Dong Huang, Jingwen Guo, Zeyu Sun, Qihao Zhu, and Jie M. Zhang. DSCodeBench: A realistic benchmark for data science code generation. *arXiv preprint arXiv:2505.15621*, 2025. doi: 10.48550/arXiv.2505.15621. URL https://arxiv.org/abs/2505.15621.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. MultiTabQA: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6322–6334. Association for Computational Linguistics, July 2023. doi: 10.18653/v1/2023.acl-long.348. URL https://aclanthology.org/2023.acl-long.348/.

Shuofei Qiao, Yanqiu Zhao, Zhisong Qiu, Xiaobin Wang, Jintian Zhang, Zhao Bin, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Scaling generalist data-analytic agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5PxFqpIYWC.

Ankit Pratap Singh, Xin Su, and Phillip Howard. Synthetic contrastive reasoning for multi-table Q&A. *arXiv preprint arXiv:2606.05382*, 2026. URL https://arxiv.org/abs/2606.05382.

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. A survey on large language model-based agents for statistics and data science. *The American Statistician*, pp. 1–14, 2025. doi: 10.1080/00031305.2025.2561140. URL https://doi.org/10.1080/00031305.2025.2561140.

Zhaoyan Sun, Shan Zhong, Daizhou Wen, Jiaxing Han, Guoliang Li, Ying Yan, Peng Zhang, Yu Su, Xiang Qi, Baolin Sun, Chengyuan Yang, Tao Fang, and Huaiyu Ruan. AgenticDataBench: A comprehensive benchmark for data agents. *arXiv preprint arXiv:2607.01647*, 2026. doi: 10.48550/arXiv.2607.01647. URL https://arxiv.org/abs/2607.01647.

Zirui Tang, Weizheng Wang, Zihang Zhou, Yang Jiao, Bangrui Xu, Boyu Niu, Xuanhe Zhou, Guoliang Li, Yeye He, Wei Zhou, Yitong Song, Cheng Tan, Bin Wang, Conghui He, and Fan Wu. LLM/Agent-as-Data-Analyst: A survey. *arXiv preprint arXiv:2509.23988*, 2025. doi: 10.48550/arXiv.2509.23988. URL https://arxiv.org/abs/2509.23988.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 540–557. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.coling-main.36. URL https://aclanthology.org/2025.coling-main.36/.

Luoxuan Weng, Yinghao Tang, Yingchaojie Feng, Zhuo Chang, Peng Chen, Ruiqin Chen, Haozhe Feng, Chen Hou, Danqing Huang, Yang Li, Huaming Rao, Haonan Wang, Canshi Wei, Xiaofeng Yang, Yuhui Zhang, Yifeng Zheng, Xiuqi Huang, Minfeng Zhu, Yuxin Ma, Bin Cui, and Wei Chen. DataLab: A unified platform for LLM-powered business intelligence. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 4346–4359. IEEE, 2025. doi: 10.1109/ICDE65448.2025.00326. URL https://doi.org/10.1109/ICDE65448.2025.00326.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Tongliang Li, Zhoujun Li, and Guanglin Niu. TableBench: A comprehensive and complex benchmark for table question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25497–25506, 2025. doi: 10.1609/aaai.v39i24.34739. URL https://ojs.aaai.org/index.php/AAAI/article/view/34739.

Sishi Xiong, Ziyang He, Zhongjiang He, Yu Zhao, Changzai Pan, Jie Zhang, Zhenhe Wu, Shuangyong Song, and Yongxiang Li. Tablezoomer: a collaborative agent framework for large-scale table question answering. *Vicinagearth*, 2(1):11, 2025. doi: 10.1007/s44336-025-00016-x. URL https://doi.org/10.1007/s44336-025-00016-x.

Deyin Yi, Yihao Liu, Lang Cao, Mengyu Zhou, Haoyu Dong, Shi Han, and Dongmei Zhang. TablePilot: Recommending human-preferred tabular data analysis with large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 355–410. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-industry.28. URL https://aclanthology.org/2025.acl-industry.28/.

Qiaohong Zhang, Weihao Ye, Jialong Chen, Yi Luo, BoYuan Li, Bowen Deng, Zibin Zheng, Jianhao Lin, Wei-Shi Zheng, and Chuan Chen. DataClawBench: An agent benchmark for exploratory real-world financial data analysis. *arXiv preprint arXiv:2605.02503*, 2026a. doi: 10.48550/arXiv.2605.02503. URL https://arxiv.org/abs/2605.02503.

Xiaokang Zhang, Sijia Luo, Bohan Zhang, Zeyao Ma, Jing Zhang, Yang Li, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, Jifan Yu, Shu Zhao, Juanzi Li, and Jie Tang. TableLLM: Enabling tabular data manipulation by LLMs in real office usage scenarios. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10315–10344. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.538. URL https://aclanthology.org/2025.findings-acl.538/.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. A survey of table reasoning with large language models. *arXiv preprint arXiv:2402.08259*, 2024. doi: 10.48550/arXiv.2402.08259. URL https://arxiv.org/abs/2402.08259.

Yuxin Zhang, Ju Fan, Meihao Fan, Shaolei Zhang, and Xiaoyong Du. CoDA-Bench: Can code agents handle data-intensive tasks? *arXiv preprint arXiv:2606.15300*, 2026b. doi: 10.48550/arXiv.2606.15300. URL https://arxiv.org/abs/2606.15300.

Wei Zhou, Bolei Ma, Annemarie Friedrich, and Mohsen Mesgar. Table question answering in the era of large language models: A comprehensive survey of tasks, methods, and evaluation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12161–12189. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.557. URL https://aclanthology.org/2026.acl-long.557/.

Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, Chengliang Chai, Chong Chen, Shimin Di, Ju Fan, Ji Sun, Nan Tang, Fugee Tsung, Jiannan Wang, Chenglin Wu, Yanwei Xu, Shaolei Zhang, Yong Zhang, Xuanhe Zhou, Guoliang Li, and Yuyu Luo. A survey of data agents: Emerging paradigm or overstated hype? *arXiv preprint arXiv:2510.23587*, 2025. doi: 10.48550/arXiv.2510.23587. URL https://arxiv.org/abs/2510.23587.

A Extended Related Work

A.1 Training-Task Synthesis for Data-Analysis Agents

A growing body of work synthesizes training tasks for data-analysis agents by pairing automated question generation with execution- or LLM-based verification (Hu et al., 2024; Tang et al., 2025; Sun et al., 2025). For instance, DataMind builds DataMind-12K via an 18-category taxonomy,

recursive easy-to-hard composition, and knowledge-augmented trajectory sampling filtered by self-consistency and rule-based checks (Qiao et al., 2026). Because such composition still asks an LLM to judge column relations from local metadata, DSGym instead lets an agent explore a data pool, posing and self-verifying questions under a six-dimensional LLM judge, yielding roughly 2K examples (Nie et al., 2026)—yet sparse cross-table relations may go unobserved under a finite exploration budget. Rather than exploring a fixed pool, Datarus-R1 programmatically generates the challenges themselves, controlling the mixture of successful, corrected, self-corrected, and failed trajectories (Ben Chaliah & Dellagi, 2025), trading real-world grounding for full control over generation. Across all three, task synthesis and relation discovery remain entangled within a single LLM-driven step—from a small local context or synthetic data—rather than factored through an explicit, verifiable representation of the real-data pool.

A.2 Synthetic Data for Text-to-SQL

Text-to-SQL synthesis is a well-studied setting with mature mechanisms for operator coverage and complexity control (Deng et al., 2022; Katsogiannis-Meimarakis & Koutrika, 2023; Liu et al., 2025). OmniSQL builds relational databases from web tables and generates complexity-aware SQL, natural-language back-translations, and chains of thought at scale (Li et al., 2025). SING-SQL hierarchically samples joinable table subsets and column windows before producing SQL at multiple complexity levels (Caferoğlu et al., 2025). DSQG-Syn uses predefined question types to ensure coverage of SQL operations (Duan et al., 2025). Our setting differs in both input and output: the source is a heterogeneous file pool rather than a database schema, and the target is an open-ended executable analysis program rather than SQL alone.

A.3 Table and Multi-Table Reasoning

Multi-table reasoning methods target the downstream consumption of already-formed questions over relational evidence, rather than the upstream problem of generating those questions in the first place (Zhou et al., 2026; Zhang et al., 2024; Pal et al., 2023). EASE extracts question entities to construct compact sub-table schemas for noisy multi-table question answering (Kang et al., 2026). Synthetic Contrastive Reasoning for Multi-Table Q&A trains with paired positive and negative reasoning trajectories and contrastive preference optimization (Singh et al., 2026). These methods assume that the question is already given. In contrast, our problem is to discover grounded cross-file relations, synthesize the question itself, and control the structural and semantic properties of the resulting task.

B Experimental and Reproducibility Details

This section reports the data pool, benchmark suite and evaluation protocol, baseline selection, and student-training configuration used in the downstream experiments. Trajectory selection is documented separately in Appendix C.

B.1 Data-Pool Statistics and Leakage Audit

We construct the initial source pool from two complementary collections of real-world data. First, we use the Kaggle-derived collection assembled by DataMind (Qiao et al., 2026), contributing 3,872 publicly available raw files. Before graph construction, we group source datasets into domains based on their topics and descriptions, and perform graph construction and task generation independently within each domain.

To broaden the diversity of file formats and data organization, we additionally incorporate the raw sources collected for KramaBench (Lai et al., 2025): 1,794 files spanning six domains and 21 filename-type buckets. Together, the two collections provide 5,666 files, which enter our parsing and quality-control pipeline. We reuse only raw source files, excluding all questions, reference answers, and solution trajectories from both collections; all downstream benchmark data are also held out from the source pool.

The DataMind inventory contains 3,563 CSV, 221 SQLite, 57 XLSX, 21 Markdown, three DB, one TXT, five extensionless, and one quoted-name SQLite file. The KramaBench inventory contains

Table 5: **Raw source-pool inventory before parsing.** Counts include 3,872 DataMind files in 15 domains and 93 second-level categories, and 1,794 KramaBench files in six workload domains. Format labels follow filename suffixes; "no ext." denotes an extensionless filename.

Domain	Files	Format counts
DataMind		
Science	786	csv 741; sqlite 32; xlsx 12; md 1
Healthcare	619	csv 578; sqlite 30; xlsx 6; md 3; txt 1; no ext. 1
Finance	459	csv 424; sqlite 31; xlsx 3; no ext. 1
Sports	425	csv 409; sqlite 11; xlsx 4; md 1
Media	349	csv 298; sqlite 36; md 6; xlsx 6; db 1; quoted-name sqlite 1; no ext. 1
Transportation	275	csv 255; sqlite 17; xlsx 3
Retail	223	csv 187; sqlite 18; xlsx 17; md 1
Sustainability	214	csv 211; md 2; xlsx 1
Other	135	csv 109; sqlite 21; xlsx 3; md 1; no ext. 1
Engineering	133	csv 113; sqlite 13; md 3; db 2; xlsx 1; no ext. 1
Agriculture	63	csv 63
Commerce	53	csv 45; sqlite 6; md 1; xlsx 1
Legal	52	csv 47; sqlite 5
RealEstate	50	csv 47; md 2; sqlite 1
Tourism	36	csv 36
KramaBench		
Astronomy	1,563	csv 1,444; hdr 37; sp3 37; txt 18; py 6; db 4; npy 3; dat 2; lst 2; md 2; pkl 2; tle 2; cdf 1; npz 1; text 1; txt.csv 1
Legal	134	csv 132; html 1; md 1
Environment	53	csv 37; py 13; json 1; md 1; txt 1
Wildfire	25	csv 17; gpkg 2; json 1; md 1; parquet 1; py 1; txt 1; xlsx 1
Archeology	10	csv 7; xlsx 2; md 1
Biomedical	9	xlsx 7; csv 1; md 1

1,638 CSV files and 156 files in 20 other filename-type buckets, including scientific formats such as HDR, SP3, CDF, NPY, NPZ, and TLE. The inventory includes the `caliber_definitions.md` file from all DataMind categories, but excludes SQLite sidecar files (`*.sqlite-shm` and `*.sqlite-wal`) and the KramaBench `__pycache__` bytecode file. Dataset subdirectories below a KramaBench domain's `input/` directory remain part of that domain.

Leakage control is enforced at the ingestion boundary. The source-file allow-list is restricted to `scripts/datamind/{domain}/DataMind/{subcategory}/` and `scripts/kramabench/{domain}/input/`; benchmark questions, answers, reference programs, and solution trajectories are never scanned or added to the graph. The four evaluation benchmark distributions are maintained outside these roots and are excluded from parsing, graph construction, task generation, and trajectory selection. Table 5 reports raw files before parser replay: because a workbook or database can produce multiple normalized tables, raw-file counts are not treated as parsed-table or parser-failure counts.

B.2 Benchmarks and Evaluation Protocol

Evaluation uses four held-out benchmarks whose data do not appear in our source pool. For the first three, we follow the curated versions and evaluation setup released by DSGym (Nie et al., 2026): DAEval, its curated variant of InfiAgent-DABench (Hu et al., 2024), tests general-purpose data handling and statistical competence; DABStep (Egg et al., 2025) tests multi-step financial analysis across multiple files; and QRData (Liu et al., 2024) tests statistical and causal reasoning over tabular data. The main and trajectory-selection tables denote this curated version as QRData-V, while the difficulty ablation shortens it to QRData. For TableBench (Wu et al., 2025), we follow prior table-reasoning evaluations (Lei et al., 2025; Xiong et al., 2025) and use its Fact Checking and Numerical Reasoning subsets. Accordingly, every TableBench score reported in the main text and tables refers to these two subsets.

Except for DataMind, all models run in the same smolagents CodeAgent harness with thinking disabled, at most 100 interaction steps, and temperature 0.7. DataMind-7B/14B are instead evaluated with their released native harness to preserve consistency between their training and inference configurations. We use three evaluation seeds (42, 123, and 456), report pass@1 averaged over these runs, and compute pass@3 under each benchmark's official evaluator. Each task has a one-hour wall-clock timeout.

For inference, vLLM serves each checkpoint using the tokenizer's native chat template. Eight GPUs host four independent inference engines with tensor parallelism two; each engine serves eight evaluation workers, for 32 concurrent workers in total. For models evaluated with the shared smolagents harness, we retain its default prompt variant and observation-history masking settings.

B.3 Baselines

We compare against three groups: the untuned Qwen3-8B backbone, released task-specific data-analysis agents, and large open-weight models. The closest task-specific comparisons are DataMind-7B/14B (Qiao et al., 2026). The large open-weight references reported in Table 1 are DeepSeek-V4-Pro, Kimi-K2.5, GLM-5.2, Qwen3-Coder-480B, and Qwen3-235B; GLM-5.2 and DeepSeek-V4-Pro are also the two teacher models used in task synthesis. We exclude DSGym-SFT-2K (Nie et al., 2026) because its training tasks are synthesized from seed subsets of QRData and DABStep, exposing benchmark-specific schemas, statistics, and cross-file relations despite using synthetic queries.

B.4 Training Configuration

Training corpus. The final training corpus contains 2,000 selected multi-turn code-agent trajectories in ShareGPT conversation format; Appendix C describes their generation and selection. Each trajectory follows the same agent protocol, in which interleaved reasoning, code execution, and tool observations culminate in a `final_answer`. Only assistant turns contribute to the SFT loss (`train_on_prompt=false`); per-turn supervision is specified by `turn_mask`, while earlier assistant turns are not additionally masked (`mask_history=false`).

Optimization and measured compute. We train all 8.19B parameters with LlamaFactory using eight-way distributed training and DeepSpeed ZeRO-3 without CPU or NVMe offload. The effective global batch size is 64 (one example per GPU, eight gradient-accumulation steps, and eight GPUs). We use BF16, FlashAttention-2, gradient checkpointing, and the Liger loss kernel; sequence packing is disabled. The Qwen3 chat template is applied with thinking mode disabled. The trainer reported 9.68×10^{17} total floating-point operations.

C Trajectory Selection

Trajectory generation. We run GLM-5.2 over the same set of verified tasks under three different random seeds, producing three candidate solution trajectories for each task. The objective is to construct a high-quality trajectory corpus for supervised fine-tuning while retaining at most one trajectory per task.

Pass filter. We first apply a task-level correctness filter. A task is retained if at least one of its three rollouts produces a correct final answer under the task's evaluator. If all three rollouts fail, the task is discarded because it has no valid trajectory from which the student can learn. For a retained task, only the correct rollouts proceed to within-task selection.

Best-of-three selection. When a retained task has multiple correct trajectories, we rank them using the following efficiency and reliability indicators:

1. the number of steps containing a code-execution exception or error;
2. the number of steps producing no effective action or an empty output;
3. the total number of action steps required to complete the task;
4. the total number of tokens consumed by the trajectory; and
5. the trajectory's wall-clock execution time.

Lower values are preferred for every indicator. We compare candidates by the five indicators jointly and retain the highest-ranked correct trajectory. This procedure yields at most one selected trajectory per task for supervised fine-tuning of Qwen3-8B.

Table 6: **Configuration of the final Qwen3-8B SFT run.** The global batch size is per-device batch size × gradient accumulation × number of GPUs.

Setting	Value
Backbone	Qwen3-8B (8.19B)
Update strategy	Full-parameter SFT
Training examples	2,000 selected multi-turn trajectories
Context length	32,768 tokens
Sequence packing	Disabled
Epochs	3
Optimizer	AdamW
Learning rate	1×10^{-5}
Learning-rate schedule	Cosine
Warmup ratio	0.03
Weight decay	0
Maximum gradient norm	1.0
Per-device batch size	1
Gradient accumulation steps	8
Number of GPUs	8
Global batch size	64
Precision	BF16
Attention implementation	FlashAttention-2
Distributed training	DeepSpeed ZeRO-3, no offload
Gradient checkpointing	Enabled
Loss kernel	Liger
Chat template	Qwen3
Thinking mode	Disabled

D Source Preparation and Graph Construction

D.1 Parsing Plans and Replay

The preparation stage separates deciding *how to read a file* from executing that decision. Candidate files are grouped by directory, extension, and a structural key: tabular files preferentially use a normalized header signature, while other formats use a filename pattern with numeric components normalized. An LLM inspects one representative per group and classifies it as `data`, `schema`, `meta`, or `ignore`. For a data group, it emits a declarative JSON parsing plan rather than executable code. The plan selects a fixed parser template and specifies its read parameters, optional schema reference, block or sheet selection, and layout transformations. Implemented templates cover delimited and fixed-width tables, Excel, JSON records, SP3, TLE, CDF, NPZ, HTML, GeoPackage, and text corpora; HDR and XML files are treated as background metadata rather than analytical tables.

Schema documents are processed separately into a registry of column names, formats, missing-value sentinels, units, descriptions, and declared types, together with source and filename-matching information. Cluster-level plans are then specialized per file. Multi-sheet workbooks may assign separate plans and stable block identifiers to data, metadata, and ignored sheets; multi-block text files use start and end markers and a supported sub-parser for each block. If sheet expansion fails, all sheets are treated as candidate data blocks. Only files with a valid data plan enter graph construction, while all classifications and failures remain recorded.

Deterministic format-specific parsers execute the plans. They normalize and disambiguate column labels, apply explicit or filename-matched schemas, replace declared missing-value sentinels, and retain units, descriptions, formats, and declared types as metadata. Partial schemas name only the columns they cover. Optional `melt` and `transpose_melt` instructions convert wide or matrix layouts into long form, specifying identifier and value columns, output names, header-value mappings, and missing-value handling. Numeric conversion is applied only when all nonmissing measure values are convertible. A failed reshape preserves the original table and records the failure; invalid plans, unsupported parser types, and file- or block-level failures likewise produce explicit error records rather than silently disappearing.

For every materialized block, the graph stores the relative source path, stable block identifier, parsing plan, schema reference, and document metadata. At task time, the sandbox retrieves this record, re-executes the same parser template, and selects the same block. It therefore reproduces the table used for graph construction when the source file, plan, schema registry, and parser implementation are unchanged, including the same column normalization and reshape rules. A missing or failed plan may trigger a best-effort tabular fallback for diagnosis, but a fallback result is not eligible for graph construction, motif sampling, or reference-program execution because it has no replay-equivalence guarantee; an unresolved read returns no table. Thus, every table used by a retained task is loaded through the same persisted plan used during graph construction, as assumed in Section 3.1.

D.2 PHYSICAL-VIEW CONSTRUCTION

Each usable parsed block becomes a TABLE node with source-file and block provenance, and its columns or valid atomic column groups become FIELD nodes. A CONTAINS edge connects each cluster to its member tables, and a HASFIELD edge connects each table to its fields. A cluster with at least two members is marked as a candidate partitioned series when its members share a header signature or its filename pattern contains a varying numeric token; otherwise it remains a heterogeneous group. This is a structural designation, not evidence that the members represent the same business quantity.

For each table, profiles contain missingness, cardinality, frequent values, applicable numeric ranges, and sample values, together with schema descriptions and units. The field-annotation model receives these profiles, authoritative column names, a table preview, and the filename pattern. Its structured output assigns a role from KEY, DIMENSION, MEASURE, TIME, GEO, and META; a semantic type family; a local concept hint; and, when applicable, geometry type or measure unit. Complementary columns that jointly express one atomic variable, such as a latitude–longitude pair, are represented as one grouped field while retaining their source-column mappings. Invalid or missing annotations fall back to deterministic role and type heuristics; they do not create global concepts. For a partitioned cluster, varying filename tokens create constant virtual year, date, index, or category fields, unless a matching constant field already exists in most member tables. The resulting time or dimension field is the candidate partition axis; after concept resolution, its concept is the target of the cluster's PARTITIONEDBY edge. Finally, within each table, every DIMENSION, TIME, or GEO field is connected to every MEASURE field by a MEASURE edge; key and meta fields are not used as analytical axes.

D.3 CONCEPT ALIGNMENT

Concept alignment maps non-meta fields from different tables to shared CONCEPT nodes in four stages. First, fields are placed into compatible role buckets: keys and dimensions share a reference bucket, while measures, times, and geographic fields use separate buckets, with geographic fields further separated by geometry type. Each field descriptor contains its role, semantic type, normalized concept hint, name tokens, and, for keys and dimensions, a bottom-K_{sketch} value-set sketch. Within a bucket, fields with the same nonempty normalized concept hint are merged directly; fields without a hint remain separate even when they have the same generic column name.

Second, candidate group pairs are proposed by shared tokens, available value signatures, or a structural gate. For a group B_i, let N_i be the tokens from its column names and concept hints and V_i its distinct nonmissing values. We use

$$s_{\text{name}}(B_i, B_j) = \frac{|N_i \cap N_j|}{|N_i \cup N_j|}, \qquad s_{\text{value}}(B_i, B_j) = \frac{|V_i \cap V_j|}{|V_i \cup V_j|}, \tag{2}$$

where value overlap is estimated from the sketches. A pair passes if $s_{\text{name}} \geq \tau_{\text{name}}$, $s_{\text{value}} \geq \tau_{\text{value}}$, or the structural gate holds. The structural gate applies to measure, time, and geographic fields and requires matching roles and compatible semantic types, plus matching geometry types for geographic fields. Passing pairs form connected candidate components; connectivity proposes groups for joint assessment but does not itself merge them. We set $\tau_{\text{name}} = 0.34$ and $\tau_{\text{value}} = 0.05$, and use bottom-$k$ sketches with $k = 256$.

Third, for each component with multiple groups, the concept model receives a group identifier, candidate label, roles, semantic types, column aliases, and sample values. It must assign every group a canonical label, using identical labels only for groups judged to denote the same concept.

Groups with matching returned labels are merged. If the model call fails or is unusable, the groups remain separate; screening evidence alone never creates a merge. After buckets are resolved, groups with the same normalized label but different roles are reassessed once, allowing, for example, age used as a measure in one table and a dimension in another to share a concept. Groups with different labels are not searched exhaustively across buckets.

Finally, filename-derived axis fields are resolved separately because their per-file singleton values need not overlap. The linker receives the union of partition values, filename-token samples, and compatible existing concepts; it either selects one existing concept or proposes a new filename-sourced concept. Failure or uncertainty creates a new concept rather than assuming a match. Each non-meta field belongs to at most one emitted concept through an INSTANCEOF edge, and a selected file-level axis is attached to its cluster through PARTITIONEDBY.

The concept-partitioning prompt presents the candidate groups, their proposed labels, roles, semantic types, aliases, and sample values, and instructs the model to assign the same canonical label only when the groups denote the same underlying entity or quantity. The partition-axis prompt presents filename tokens, the union of partition values, and compatible existing concepts, and asks the model either to select one concept or to create a new one; uncertainty must result in a new concept. Both prompts require structured assignments that cover every input group and forbid merges based on a shared name alone.

D.4 CROSS-SOURCE RELATIONS AND PROVENANCE

Cross-source edges are produced by fixed rules after concept alignment, with no additional model calls. Concept sharing licenses a candidate correspondence, not an arbitrary join. A row LINK is emitted for a shared concept of kind KEY or DIMENSION when at least one endpoint is not a filename-derived virtual field, the fields belong to different tables, and both endpoint roles are KEY or DIMENSION. Two key endpoints yield an equi link; otherwise the link is fuzzy. We exclude pairs in the same `shared_schema` cluster, numeric–string type mismatches, and pairs in which both endpoints are dense sequential integer keys, defined by `n_unique=count` and $\texttt{count}/(\texttt{max} - \texttt{min} + 1) \geq 0.9$. There is no minimum value-overlap requirement: `value_overlap` stores the Jaccard overlap when it is positive and computable, and is null otherwise, without suppressing the edge. The threshold $\texttt{UNIQUE_KEY_MIN_RATIO} = 0.98$ affects only the estimated cardinality. After adding `source` and `target`, each serialized row link stores `type`, `relation`, `cardinality`, `concept_id`, `confidence=1.0`, `value_overlap`, `transferable=true`, and `description`.

Spatial links connect geographic fields under a shared concept and record the geometry-dependent predicate: point–point pairs use `nearest`, while other supported geometry combinations use `intersects`. FILESERIES connects a partitioned cluster to each measure concept present in a majority of its materialized member tables and records the member coverage and partition concept when available. Temporal links join time fields in different tables that share a time-axis concept and retain the evidence used to align the resulting series. After adding their endpoints, serialized temporal links store `type`, `relation=temporal`, `align_concept_id`, `left`, `right`, `transferable=false`, and `description`.

Every emitted relation therefore retains source provenance and the evidence used by its admission rule. Later motif sampling reads these graph edges rather than re-inferring relations from raw files. The evidence supports inspection and operator selection, but does not by itself guarantee that every admitted relation is meaningful for every downstream analysis.

E MOTIF SAMPLING AND OPERATOR CATALOG

E.1 CANDIDATE ASSEMBLY AND INPUT SIGNATURES

As described in Section 3.3, candidates are centered on a table, a file cluster, or sources connected by graph-supported links. Their eligible keys, measures, grouping dimensions, time axes, and partition metadata, when the cluster is partitioned, are read from graph nodes and edges. Cross-source candidates use only recorded equi, fuzzy, or spatial links whose endpoints resolve to different tables. Equi and fuzzy links require recorded value overlap, spatial links use their recorded `nearest`

or `intersects` predicate, and an expansion cannot reintroduce a table already in the candidate. Temporal links are reserved for branch coupling. Many-to-many links are treated as dimension alignment rather than row-level joins, avoiding duplicated contributions to downstream aggregates.

Each candidate is summarized by the input signature

$$\begin{aligned}\sigma(\mathcal{G}_M) = \big(&\mathsf{keys}, \mathsf{measures}, \mathsf{group_dims}, \mathsf{time_axes},\\ &\mathsf{filename_axis}, \mathsf{relations}, \mathsf{side_count}, \mathsf{member_files},\\ &\mathsf{routing}, \mathsf{text_parse_cols}, \mathsf{grouped_context}\big).\end{aligned} \tag{3}$$

Here keys, measures, group_dims, time_axes, and text_parse_cols record eligible fields. filename_axis is an optional virtual partition field; relations records the links used for assembly; side_count distinguishes an unlinked candidate from a linked composition; and member_files records cluster size. routing indicates that a partition value can select a cluster member, while grouped_context indicates within-group analysis.

E.2 Operator Compatibility and Catalog

An optional transformation plan π updates the signature to $\sigma' = T_\pi(\sigma)$. For each family ω, the catalog specifies

$$\begin{aligned}&\mathrm{requires}_\omega(\sigma') \in \{0,1\}, \qquad \mathrm{produces}_\omega(\sigma') \in \mathcal{Y},\\ &\mathrm{expand}_\omega(\sigma') = (o_1, \ldots, o_\ell),\end{aligned} \tag{4}$$

where $\mathcal{Y} = \{\mathrm{scalar}, \mathrm{series}, \mathrm{set}, \mathrm{entity}, \mathrm{list}, \mathrm{label}\}$ is the output vocabulary, and the expansion supplies executable analysis operations. A family is compatible only when its input contract accepts σ'. The catalog contains thirteen families: Agg, TopK, Distinct, Set, Read, Trend, ReduceGrp, Correlate, Stats, Contingency, Outlier, Model, and Text. Compatibility establishes input availability, not the validity of a task-specific statistical interpretation, which is checked during generation and verification.

Table 7 gives the final quota taxonomy and the concrete operator chains represented by each family or coupler. For `STATS`, chains containing `skewness_kurtosis` or `normality_test` belong to Distribution Analysis, while the remaining chains belong to Statistical Analysis. A coupled motif is categorized by its coupler rather than by either child family.

E.3 Transformations and Coupling

The binding β assigns graph fields or transformation outputs to the roles required by a matched family, such as `measure`, `group_by`, `axis`, `target`, and `features`. It fixes the data operands but defers task-specific values—including filter values, bin boundaries, ranking cutoffs, window sizes, and significance levels—to task generation. A candidate is discarded if its required operands cannot be bound.

A transformation prefix contains zero to three equality or range filters, derived-column, binning, string-parsing, rolling, cumulative, year-over-year, or within-group operations. Each step updates the signature before matching and may introduce a symbolic derived field without adding a graph node. For a single branch b, the resulting chain is

$$\begin{aligned}L_b = \ &\mathrm{assemble}(\mathcal{G}_M) \oplus \mathrm{transform}(\pi, \beta)\\ &\oplus \mathrm{analyze}(\mathcal{G}_M, \omega, \sigma', \beta) \oplus \mathrm{format}(y_b),\end{aligned} \tag{5}$$

where $\oplus$ denotes ordered concatenation and $y_b = \mathrm{produces}_\omega(\sigma') \in \mathcal{Y}$ is the branch output type. The chain and binding form a branch in O.

A coupled motif combines two complete branches through a graph-supported anchor. The Correlate coupler requires two series outputs aligned by a temporal relation and sufficient overlapping axis points. Derive requires two scalar outputs on distinct measure concepts whose normalized units agree; two unknown units are admitted, but a known–unknown pair is not. It returns the difference between the two scalar outputs. Pipe requires an upstream entity output that matches a partition concept and routes the downstream scalar analysis to the selected cluster member. Each child must independently satisfy its operator and binding contract. Source expansion uses a separate hop budget, whose default maximum is one.

Quota category	Family / coupler	Concrete operator chain(s)
Aggregation	AGG	accumulate_across_files; groupby_aggregate; mean
	REDUCE_GRP	latest_per_group + tie_breaking + groupby_aggregate
	DERIVE coupler	gap_compute
Counting	DISTINCT	count_distinct
	TEXT	string_parsing + token_frequency
Ranking	TOPK	topk_select
Time-based Calculation	TREND	trend_direction_compare
Correlation Analysis	CORRELATE family	correlation
	CORRELATE coupler	correlation + trend_direction_compare
Statistical Analysis	STATS	confidence_interval; hypothesis_test
	CONTINGENCY	chi_square_test; expected_frequency; conditional_probability
	MODEL	linear_regression_fit, or train_test_split + linear_regression_fit + model_error_metric
Distribution Analysis	STATS	skewness_kurtosis; normality_test
Anomaly Detection	OUTLIER	zscore_anomaly; iqr_outlier
Cross-source Linking	READ	value_lookup
	SET	set_difference; set_intersection; set_union
	PIPE coupler	dynamic_file_lookup

Table 7: Final nine-category quota taxonomy. A semicolon separates alternative operator chains, whereas + denotes ordered composition within one chain.

The name CORRELATE is used in two explicitly distinguished roles. The single-branch family correlates compatible variables available within one candidate, whereas the CORRELATE coupler combines the series outputs of two complete branches through a temporal link. References to a family or a coupler always specify the role; the coupler contributes to coupling complexity rather than replacing either child's family tier.

E.4 QUOTAS AND DEDUPLICATION

Motifs are grouped into nine categories in the fixed order Aggregation, Counting, Ranking, Time-based Calculation, Correlation Analysis, Statistical Analysis, Distribution Analysis, Anomaly Detection, and Cross-source Linking. Let $\mathcal{C}$ be the categories with at least one eligible candidate, retaining this order, and let N_{motif} be the requested budget. For $\mathcal{C} \neq \emptyset$, the initial quota of category $\kappa \in \mathcal{C}$ is

$$n_\kappa = \left\lfloor \frac{N_{\text{motif}}}{|\mathcal{C}|} \right\rfloor + \mathbb{I}[\text{rank}_{\mathcal{C}}(\kappa) \leq N_{\text{motif}} \bmod |\mathcal{C}|] \,, \tag{6}$$

where ranks start at one, so $\sum_{\kappa \in \mathcal{C}} n_\kappa = N_{\text{motif}}$. Selection alternates among motif kinds within each category. Unused quota spills to later nonempty categories until the budget is filled or all candidates are exhausted. A canonical key over source structure, family or coupler, bindings, transformation plan, and coupled children provides global deduplication. Fixed ordering and seeded choices make sampling deterministic for fixed inputs. Sparse categories may miss their quota, and verification and difficulty filtering may further change the final counts.

Table 8: **Ablation of SFT trajectory selection.** Both variants fine-tune Qwen3-8B on 2,000 trajectories from the same candidate pool using an identical SFT recipe.

Trajectory selection	DAEval		DABStep		QRData-V		TableBench	
	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3	pass@1	pass@3
Random	78.64	90.61	15.58	22.82	48.34	68.92	67.34	77.23
DATA2TASK	**84.16**	**92.82**	**18.57**	**23.27**	**49.94**	**70.52**	**68.02**	**80.90**
Δ over Random	+5.52	+2.21	+2.99	+0.45	+1.60	+1.60	+0.68	+3.67

F DIFFICULTY SCORING

F.1 FEATURE EXTRACTION

Structural features are extracted from the serialized motif sample without modifying it. Chain complexity is `len(operation_chain)` plus `transform_depth`. Linking complexity sums `hop_count`, the indicator that any branch has `signature.member_files`> 1, and the number of relations labeled `fuzzy`. The operator feature is the maximum tier over branch families. Coupling complexity counts nodes labeled `coupled` in `structure_tree`; if the tree is absent but a `coupler` is present, the count is one. Missing structural fields use neutral defaults for compatibility with older artifacts. The dirty-data indicator is one if the task generator supplies `dirty_used=true` or the operation chain contains `fuzzy_match`, `unit_conversion`, or `dirty_numeric_cast`; otherwise it is zero.

F.2 OPERATOR TIERS

The family-to-tier mapping is: tier 0, `READ` and `DISTINCT`; tier 1, `AGG`, `TOPK`, and `TEXT`; tier 2, `TREND`, `REDUCE_GRP`, and `SET`; tier 3, `STATS`, `CONTINGENCY`, `OUTLIER`, and `CORRELATE`; and tier 4, `MODEL`. Here `CORRELATE` denotes the single-branch family; a CORRELATE coupler is accounted for by x_{couple}. Unknown families default to tier 1. This mapping must remain aligned with the operator catalog.

G ADDITIONAL RESULTS AND ABLATIONS

G.1 EFFECT OF SFT TRAJECTORY SELECTION

We evaluate the trajectory-selection stage using two matched SFT sets drawn from the same candidate pool. The *random selection* baseline uniformly samples 2,000 trajectories, while DATA2TASK selects 2,000 trajectories using its scoring and filtering procedure. All other training settings are kept identical.

The selected trajectories outperform random selection across all four benchmarks, with the largest pass@1 gain on DAEval. This confirms that the trajectory-selection stage identifies more useful supervision for SFT than uniform sampling.

G.2 DIVERSITY OF SYNTHESIZED TASKS

A key objective of DATA2TASK is to generate diverse analytical tasks from heterogeneous real-world data sources. We analyze the synthesized task distribution across broad domains and analytical categories. The heatmap in Figure 3 reports the task count for each domain–category combination; color indicates the corresponding category's within-domain share relative to an even allocation of 11.1% across the nine categories.

The 2,000 synthesized tasks cover all 18 broad domains and all nine analytical categories, demonstrating substantial diversity in both data context and analytical behavior. The distribution is intentionally heterogeneous rather than uniform: correlation analysis is especially prominent in finance and retail, while cross-source linking occurs frequently in sports and science. Other domain-specific concentrations, such as distribution analysis in astronomy and aggregation in agriculture, further

indicate that DATA2TASK instantiates operators where the underlying data support them instead of repeating a small set of domain-agnostic templates.

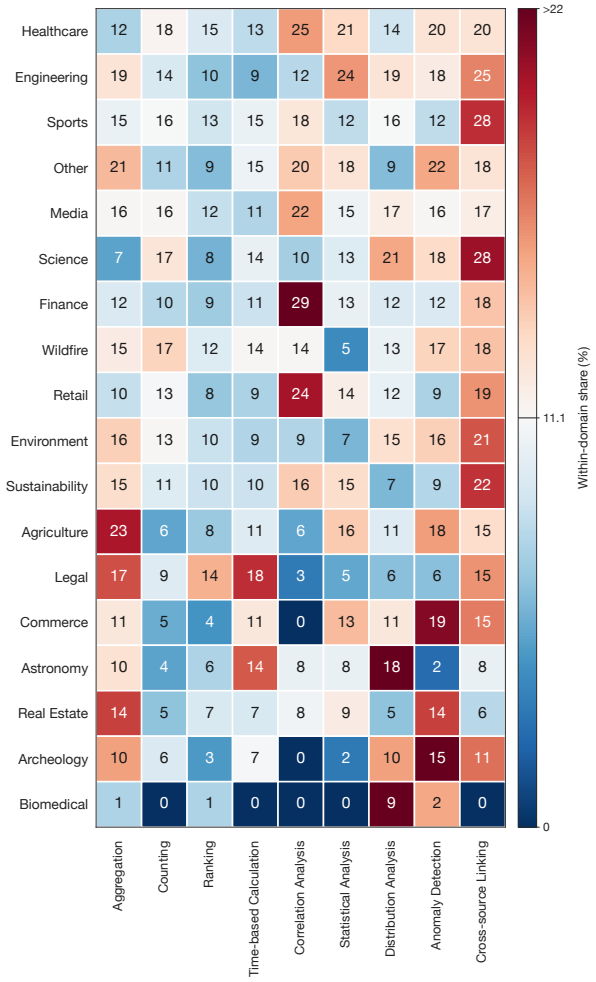


Figure 3: **Domain–category distribution of synthesized tasks.** Cell labels give task counts, while color encodes each category's within-domain share relative to an even 11.1% allocation across the nine analytical categories.