DeepEye: A Workflow-Centric Agentic Data System for Steerable Data Analytics

Boyan Li
HKUST(GZ)
Guangzhou, China
bli303@connect.hkust-gz.edu.cn

Yiran Peng
HKUST(GZ)
Guangzhou, China
yiranpeng@hkust-gz.edu.cn

Yupeng Xie
HKUST(GZ)
Guangzhou, China
yxie740@connect.hkust-gz.edu.cn

Sirong Lu
HKUST(GZ)
Guangzhou, China
slu075@connect.hkust-gz.edu.cn

Yizhang Zhu
HKUST(GZ)
Guangzhou, China
yzhu305@connect.hkust-gz.edu.cn

Xing Mu
HKUST(GZ)
Guangzhou, China
xmu159@connect.hkust-gz.edu.cn

Xinyu Liu
HKUST(GZ)
Guangzhou, China
xliu371@connect.hkust-gz.edu.cn

ABSTRACT

Large language models have enabled natural language interaction with data, yet current data agents remain difficult to deploy as reliable analytical systems. They often couple all tools, intermediate observations, and execution traces into a single conversational context, making heterogeneous analysis fragile, opaque, and hard to optimize. We present DeepEye, a workflow-centric agentic data system that turns user intents into transparent and steerable analytical workflows. DeepEye abstracts heterogeneous capabilities as typed nodes, separates probabilistic AgentNodes from deterministic ToolNodes, and orchestrates them through a database-inspired engine with compilation, validation, optimization, and execution stages. This design allows the planner to reason at the level of typed interfaces while local agents solve specialized tasks under isolated contexts. It also exposes workflow states for human inspection and intervention. To support reliable database question answering, we develop a dedicated natural-language query engine inside DeepEye, DeepEye-SQL, which achieves 73.5% execution accuracy on BIRD-Dev and 89.8% on Spider-Test. For data video generation, we develop DataMagic as the system's Video Generator, a declarative multi-agent method that improves data-video quality from 2.13 to 3.89 on 109 samples. A case study further illustrates how DeepEye coordinates database querying, document analysis, and multimodal report generation in a single auditable workflow.

VLDB Workshop Reference Format:
Boyan Li, Yiran Peng, Yupeng Xie, Sirong Lu, Yizhang Zhu, Xing Mu, and Xinyu Liu. DeepEye: A Workflow-Centric Agentic Data System for Steerable Data Analytics. VLDB 2026 Workshop: ADS 2026: The Joint Workshop on Agentic Data Systems and Data-Centric AI (The 1st ADS & 3rd DATAI).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://deepeye.tech/.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

1 INTRODUCTION

Large Language Models (LLMs) are changing the way users interact with data. Instead of writing SQL, configuring dashboards, or manually switching among data tools, users increasingly expect *data agents* that can interpret analytical intent, access enterprise data sources, and synthesize actionable insights through natural language [17, 28]. This vision is especially appealing for business users, who often need to combine structured databases, spreadsheets, documents, and domain knowledge into a coherent analysis.

However, the current generation of data agents remains closer to "ChatBI" than to dependable analytical systems. Most systems assemble a set of tools around an LLM and let the model decide a linear sequence of calls. This design is flexible, but it becomes brittle as tasks become longer, more heterogeneous, and more interactive. A user request such as "analyze global sales performance and explain the main risks" may require database querying, document retrieval, metric definition lookup, visualization, and report generation. When all tool descriptions, intermediate outputs, and reasoning traces are placed in a single context window, the agent easily loses focus, hallucinates tool parameters, or fails to reuse partial results.

We identify three recurring obstacles: (C1) *heterogeneous data grounding*, where real analytical tasks rarely live in one modality and existing Text-to-SQL [12, 13] or retrieval-augmented systems [11] provide only partial coverage; (C2) *context explosion*, where complex workflows force a monolithic agent to keep many tools, intermediate artifacts, and reasoning traces in one global memory, creating interference between high-level planning and local task execution; and (C3) *unverifiable execution*, where LLM-generated chains are opaque and sequential, making it difficult for users to inspect whether SQL queries, chart specifications, or downstream report steps are well-typed, and for the system to exploit parallelism among independent steps.

This paper presents DeepEye, a workflow-centric agentic data system designed to address these challenges. DeepEye departs from

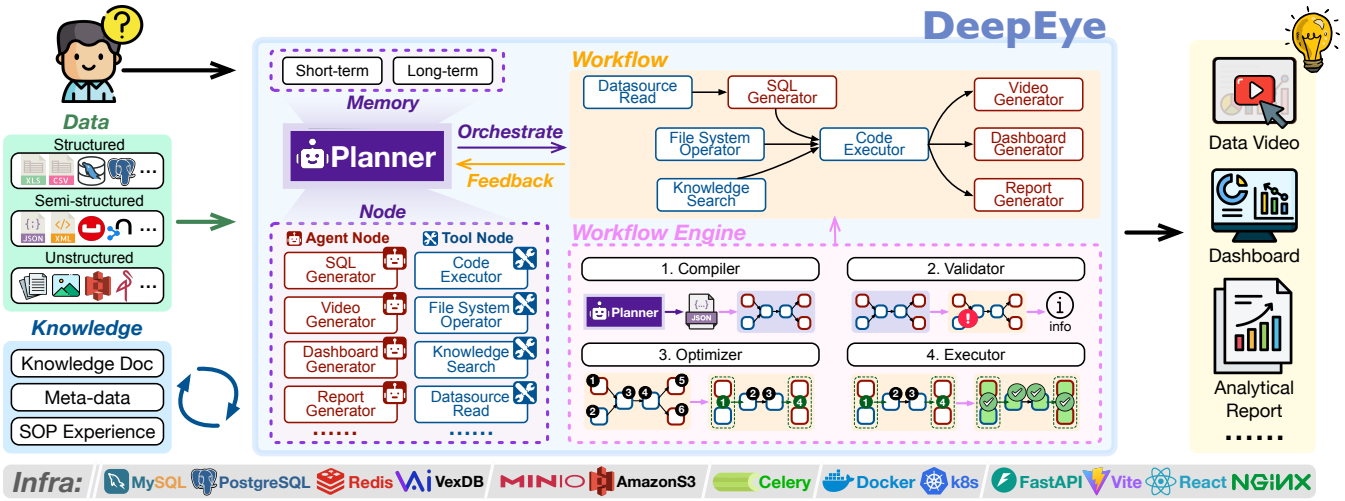

Figure 1: The workflow-centric architecture of DeepEye. A memory-augmented planner produces typed workflow DAGs; the workflow engine compiles, validates, optimizes, and executes them over heterogeneous AgentNodes and ToolNodes.

the linear chat paradigm and represents analysis as a transparent Directed Acyclic Graph (DAG). Each node exposes a typed interface and encapsulates either deterministic execution logic or probabilistic local reasoning. A memory-augmented planner maps user intent into a logical workflow, while a database-inspired workflow engine compiles the plan into executable nodes, validates type and dependency constraints, optimizes the topology, and executes independent branches in parallel.

The key idea is to make agentic analytics *steerable*. A workflow is not only an internal trace but also a user-facing object that can be inspected, edited, validated, and archived. Users may bind explicit context through references such as *@Sales Database* or *@Financial Metrics*, inspect generated SQL inside an AgentNode, modify node parameters, and rely on the validator to detect schema mismatches before execution. At the same time, the planner can learn from successful workflows by saving them as standard operating procedure (SOP) templates.

This paper makes three contributions. (1) We present DeepEye, an end-to-end workflow-centric data intelligence agent system that models heterogeneous analytics as inspectable and reusable workflow DAGs. (2) We design the core abstractions and runtime of DeepEye, including typed AgentNodes/ToolNodes, memory-augmented planning, context-isolated execution, validation, optimization, parallel scheduling, and human steering. (3) We evaluate key system capabilities through the DeepEye-SQL [12] and DataMagic [3] subsystems, together with a global-sales case study, demonstrating reliable database grounding and multimodal artifact generation.

2 PROBLEM FORMULATION

Inputs and Artifacts. An analytical session starts with a natural language request $\mathcal{R}$, an optional context binding $\mathcal{B}$, and system memory $\mathcal{M}$. The binding $\mathcal{B}$ contains user-selected resources such as databases, uploaded files, spreadsheets, or domain knowledge collections; $\mathcal{M}$ contains schema metadata, historical workflows, and reusable standard operating procedures (SOPs). We represent every intermediate result as a typed artifact $a = \langle x, \tau \rangle$, where x is the artifact content and τ is its type, e.g., relation, document chunk, chart specification, dashboard, data video, or report.

Agentic Data Workflow. We model an analytical task as the synthesis and execution of a workflow DAG:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}). \tag{1}$$

Each vertex $v \in \mathcal{V}$ is a typed node:

$$v = \langle d_v, I_v, O_v, c_v, \phi_v \rangle, \tag{2}$$

where d_v is a semantic description, I_v and O_v are input and output ports, c_v is the configuration, and ϕ_v is the node's internal execution logic. A port is a schema tuple $p = \langle k, \tau, \delta \rangle$, consisting of a key, type, and natural language description. Each edge $e = (u, o_i, v, p_j) \in \mathcal{E}$ connects an output port $o_i \in O_u$ to an input port $p_j \in I_v$, denoting an artifact dependency.

Workflow Synthesis. The planner solves a constrained synthesis problem:

$$\Pi_\theta(\mathcal{R}, \mathcal{B}, \mathcal{M}) \to \mathcal{G}, \tag{3}$$

where Π_θ denotes the LLM-based planner. Rather than optimizing an opaque scalar objective, DeepEye first requires the generated workflow to be executable and valid:

$$\textsc{Valid}(\mathcal{G}) \Leftrightarrow \textsc{Acyclic}(\mathcal{G}) \wedge \forall e = (u, o_i, v, p_j),\ \tau(o_i) \preceq \tau(p_j). \tag{4}$$

Here, $\preceq$ denotes type compatibility, allowing exact matches or approved coercions. The validator also checks that required ports, credentials, and bound resources are available before execution. Given a valid workflow, the executor produces user-facing analytical artifacts such as SQL results, charts, dashboards, data videos, and reports.

G1: Unified Heterogeneous Control. The workflow should expose a common interface for heterogeneous operators. A database query node, a document retrieval node, a Python executor, and a report writer should all be schedulable and composable without forcing the global planner to understand their internal implementations.

G2: Context-Isolated Reasoning. The global planner should reason about workflow topology and dependencies, not all local reasoning details. Specialized agents should receive only the context needed for their local task, thereby reducing prompt interference and limiting the blast radius of hallucinated intermediate thoughts.

G3: Execution-Grounded Reliability. The generated workflow should be checked before and during execution. Structural errors such as cycles, missing credentials, or incompatible data schemas should be caught before runtime; recoverable execution failures should be returned to the planner as concrete feedback for repair.

G4: Human Steerability. Enterprise analytics often requires user intervention. Users should be able to inspect intermediate artifacts, edit node parameters, validate modifications, and archive a verified workflow as reusable organizational knowledge.

3 DEEPEYE ARCHITECTURE

Figure 1 shows the architecture of DeepEye. The system instantiates the formulation in Section 2 through four tightly coupled components: a node registry, a memory-augmented workflow planner, context-isolated AgentNodes, and a steerable workflow control plane. The workflow engine in Section 4 then validates, optimizes, and executes the generated DAG.

3.1 Node Registry and Unified Protocol

The node registry is the system catalog of DeepEye. As summarized in Table 1, it stores a *node card* for each available capability, including its registry key, semantic description d_v, node class, input/output port schemas, default configurations, resource requirements, validation hooks, and runtime backend. This registry plays the same role for agentic workflows that a system catalog plays for a DBMS: it defines what operators exist and how they can be composed.

During planning, the planner does not prompt the LLM with arbitrary implementation details. Instead, it retrieves a compact set of candidate node cards according to the user request, context binding, and memory. Candidate retrieval uses the semantic description d_v for capability matching, while port schemas prune invalid compositions. For example, a node that expects a relation cannot be connected directly to a document chunk unless an approved conversion node exists. This design reduces the search space of workflow synthesis and limits hallucinated tool calls.

ToolNodes. ToolNodes encapsulate deterministic operations such as database access, file parsing, code execution, and artifact storage. Their execution logic is implemented as code and can be retried, sandboxed, and audited. ToolNodes are useful whenever correctness depends on deterministic system behavior rather than language-model judgment.

AgentNodes. AgentNodes encapsulate probabilistic reasoning. The SQL Generator is backed by the DeepEye-SQL query subsystem and may perform schema linking, SQL generation, verification, and repair. We develop DataMagic [3] as the Video Generator subsystem: it produces a DVSpec scene sequence that binds charts, narration, and animation triggers to data references before rendering video assets. A report AgentNode may synthesize retrieved evidence and computed results into a narrative artifact. Each AgentNode may own private prompts, tools, and short-term memory, but exposes only typed ports and structured logs to the outer workflow.

3.2 Memory-Augmented Workflow Planning

The planner maps $(\mathcal{R}, \mathcal{B}, \mathcal{M})$ to a logical workflow in four stages. First, it grounds the request using explicit bindings such as *@Sales Database* and *@Financial Metrics*. Second, it retrieves relevant schema memory, domain memory, and SOP memory. Schema memory contains table names, column types, constraints, and sample values; domain memory contains metric definitions and business documents; SOP memory stores verified historical DAGs. Third, it selects candidate nodes from the registry and constructs a topology. Fourth, it fills node parameters and artifact bindings.

The planner output is a structured logical plan rather than free-form text. Each node record contains a stable identifier, registry key, input bindings, output aliases, dependencies, configuration values, and an editability flag. This plan is the contract between the LLM planner and the workflow engine. It prevents execution from depending on hidden chain-of-thought text and gives the compiler a concrete intermediate representation to parse.

SOP reuse is important for enterprise settings. If the retrieved SOP matches the current request, the planner instantiates the stored template and only adapts bindings and parameters. Otherwise, it synthesizes a new topology. After a successful run, the workflow can be archived back into memory, either automatically or after expert confirmation from the user interface.

3.3 Context-Isolated AgentNodes

DeepEye separates global orchestration context from local reasoning context. The global planner sees the user request, selected bindings, retrieved node cards, high-level artifacts, and structured feedback. It does not see every token produced inside each AgentNode. Conversely, an AgentNode receives only the local inputs required by its port schema and may use internal tools specialized for its task.

This isolation is designed to mitigate context explosion. For example, the SQL Generator can maintain detailed schema-linking evidence and candidate SQL traces internally, and the Video Generator can maintain DVSpec scene plans, narration segments, and rendering traces internally, while the report AgentNode only receives final relations, SQL provenance summaries, video assets, and relevant business evidence. The outer DAG therefore remains compact and auditable even when individual nodes perform complex reasoning.

3.4 Steerable Workflow Control Plane

DeepEye exposes the workflow intermediate representation through a graphical control plane. The chat panel captures intent and context binding; the canvas visualizes nodes, dependencies, and execution status; the inspector exposes node parameters, generated SQL, prompt summaries, schemas, logs, and output artifacts. User edits are applied to the workflow IR rather than to an opaque chat transcript. After an edit, the modified DAG is sent back to the validator before execution resumes.

This control plane turns human intervention into a first-class system operation. A user can pause execution, inspect a suspicious

Table 1: Node contracts in the DeepEye registry. The table reports external interfaces exposed to the planner and validator.

Node	Kind	Main Inputs	Main Outputs	Validation / Runtime Checks
Agent Nodes				
SQL Generator	Agent	Question, Schema, Database Context	SQL Query, Relation, Query Log	Schema Linking; Syntax and Execution Feedback
Video Generator	Agent	Insights, Charts, Narration Goal	DVSpec, Data Video Assets	Data-Binding Coverage; Media Rendering Status
Dashboard Generator	Agent	Relation, Chart Intent, Layout Goal	Dashboard Specification and Preview	Data-Column Compatibility; Layout Completeness
Report Generator	Agent	Relations, Charts, Evidence	Analytical Report	Evidence Coverage; Citation and Provenance Links
Tool Nodes				
Datasource Read	Tool	Bound Database, Query or Table Reference	Relation, Schema Snapshot	Credential Availability; Schema and Access Checks
File System Operator	Tool	File Path, Operation, Artifact Handle	File Artifact, Parsed Metadata	Path Validity; File Size and Format Checks
Knowledge Search	Tool	Question, Knowledge Collection	Evidence Chunks, Provenance Summary	Collection Access; Chunk Provenance; Relevance Score
Code Executor	Tool	Code, Input Artifacts	Derived Table, Chart Spec, File Artifact	Sandboxed Execution; Timeout; Output Type Check

SQL query, adjust a filter, reconnect a downstream node, and rerun only affected parts of the workflow. Once accepted, the edited workflow can be saved as an SOP, allowing future planning to reuse human expertise instead of rediscovering the same workflow from scratch.

4 WORKFLOW ENGINE

DeepEye uses a database-inspired workflow engine to turn LLM-generated logical plans into reliable execution. The engine separates plan generation from plan execution, allowing the system to apply static checks and topology optimization before invoking expensive or stateful tools.

Compilation. The compiler parses a logical JSON plan into a DAG object. It resolves variable references, binds output ports to downstream input ports, instantiates concrete node classes, and materializes default configurations. The result is an executable plan whose structure can be inspected independently of the LLM prompt that produced it.

Validation. The validator performs static analysis before execution. It checks acyclicity, required inputs, port compatibility, credential availability, and artifact accessibility. For example, if a SQL node produces a relation but a downstream report node expects a chart specification, the validator reports a type mismatch before the workflow runs. Validation also supports human-in-the-loop editing: a user can manually revise SQL or reconnect nodes, and the same checks prevent invalid modifications from entering execution.

Optimization. The optimizer analyzes the DAG topology using dependency information. It groups nodes into execution layers, where nodes in the same layer have no unresolved dependencies and can be dispatched concurrently. This optimization is important for multimodal analytics: retrieving financial metric definitions from a document collection and querying sales records from a database can proceed in parallel, while a downstream report node waits for both artifacts.

Execution and Feedback. The executor schedules optimized layers over an asynchronous backend implemented with Celery [5] and Redis [22]. Artifacts are exchanged through a shared object store backed by PostgreSQL metadata [20] and MinIO/S3 object storage [1, 19]. ToolNode execution runs in sandboxed Docker containers [18]. Transient errors trigger retries; non-transient errors are returned to the planner with structured traces. The planner can then repair a node parameter, replace a node, or adjust the topology while preserving unaffected parts of the workflow.

This architecture yields two practical advantages. First, reliability is no longer delegated entirely to LLM self-reflection; it is grounded in typed interfaces, deterministic checks, and execution traces. Second, efficiency emerges from the workflow graph itself: independent tasks become explicit, and the scheduler can exploit parallelism without relying on the LLM to manually interleave operations.

5 EVALUATION

5.1 DeepEye-SQL Query Subsystem

Query Subsystem. Database access is a core capability of DeepEye. To support reliable question answering over structured data, we developed DeepEye-SQL [12] as the system's dedicated natural-language query subsystem. It follows the same workflow-centric philosophy as DeepEye: SQL generation is decomposed into schema grounding, N-version implementation, deterministic tool-chain testing, and confidence-aware release. Its evaluation therefore provides direct evidence for a critical subsystem of DeepEye and for the broader design principle of embedding LLM reasoning inside verifiable workflows rather than executing it as a single free-form

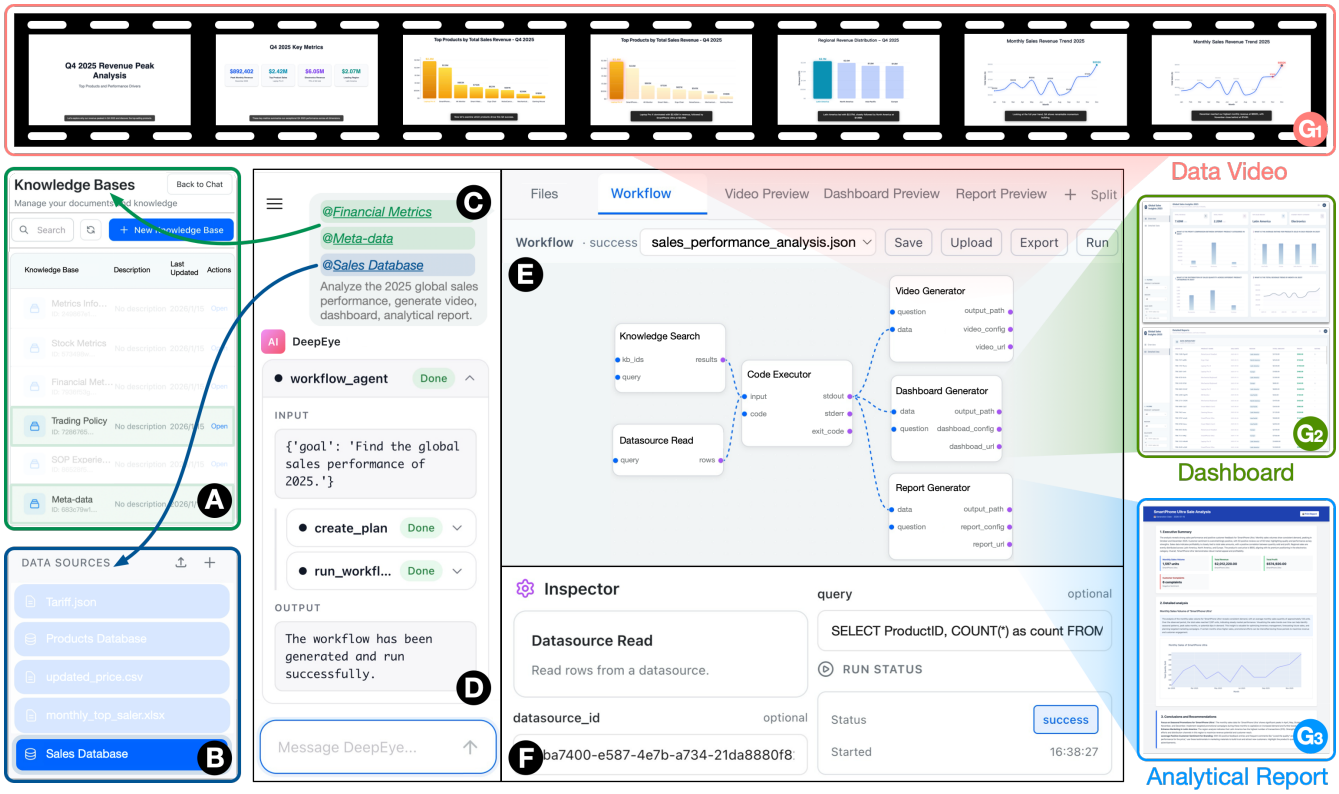

Figure 2: Running example of DeepEye on a global sales analysis task. The interface combines knowledge bases (A), data sources (B), chat-based context binding (C), workflow-agent execution logs (D), a workflow canvas (E), node inspection (F), and multimodal outputs including a data video (G1), dashboard (G2), and analytical report (G3).

Table 2: Representative performance comparison on BIRD-Dev and Spider-Test. BIRD-EX and Spider-EX denote execution accuracy (%).

Method	Model	BIRD	Spider
Fine-tuning-based Baselines			
OmniSQL [14]	Qwen2.5-Coder-32B	67.0	89.8
CHASE-SQL [21]	Gemini-1.5-Pro/Flash	73.0	87.6
XiYan-SQL [16]	GPT-4o + Qwen2.5-Coder	73.3	89.7
Prompting-based Baselines			
RSL-SQL [4]	GPT-4o	67.2	87.9
CHESS [23]	Gemini-1.5-Pro	68.3	–
Alpha-SQL [13]	Qwen2.5-Coder-32B	69.7	–
DeepEye-SQL Query Subsystem			
DeepEye-SQL	**Gemma3-27B**	**71.1**	**88.9**
DeepEye-SQL	**Qwen2.5-Coder-32B**	**70.9**	**88.7**
DeepEye-SQL	**Qwen3-Coder-30B**	**73.5**	**89.8**

response. The reported configurations cover representative proprietary and open-weight model families, including GPT-4o [8], Gemini [2], Gemma 3 [9], Qwen2.5-Coder [7], and Qwen3-Coder [25].

Benchmark Effectiveness. Table 2 evaluates whether the query subsystem can answer database-grounded questions accurately on standard Text-to-SQL benchmarks [15, 26]. For compact single-column presentation, we omit model-size columns and retain representative strong baselines from the dedicated DeepEye-SQL evaluation. With Qwen3-Coder-30B, DeepEye-SQL achieves 73.5% EX on BIRD-Dev, slightly above strong multi-generator and multi-path baselines, and reaches 89.8% EX on Spider-Test. On the official BIRD-Test split, it further reports 75.07% EX. More importantly for DeepEye, the same subsystem remains effective in the enterprise-style Spider 2.0 setting [10]: it reaches 50.5% on Snow and 38.2% on Lite, compared with 38.0% and 29.6% for ReFoRCE [6] and 33.1% on Lite for LinkAlign [24].

Workflow Mechanisms. The gains are consistent with the internal workflow design of the subsystem. Robust schema linking reaches 98.1% table recall and 95.4% column recall while reducing the schema context from 5486.2 full-schema tokens to 627.4 linked-schema tokens. N-version generation reaches 71.7% selected EX and 81.1% upper-bound EX, showing that independently generated candidates solve complementary subsets of questions. Removing ICL generation, semantic value retrieval, or tool-chain testing reduces accuracy by 2.5, 2.1, and 2.1 points, respectively, validating the value of both semantic grounding and executable verification.

Runtime Efficiency. Efficiency is equally important because the SQL Generator is invoked inside larger DeepEye workflows rather than as an isolated benchmark program. On BIRD-Dev, DeepEye-SQL uses 23.21K input tokens and 23.16K output tokens per query,

Table 3: Evidence for the DataMagic Video Generator. Quality scores use a 1–5 rubric on 109 real-world samples; the user study involved 12 participants.

Metric	Direct / Baseline	DataMagic
Avg. Video Quality	1.91–2.22	3.38–3.89
Execution Success	48.62–86.24%	95.41–98.17%
Narrative Quality	1.88–2.07	3.16–3.84
Animation Quality	1.65–1.94	3.79–4.39
Creation Time	39.2 min	8.4 min
Usability Score	3.22	5.97

with an average latency of 17.8 seconds and 73.5% EX. In comparison, CHESS [23] uses 327.02K input tokens and 284.4 seconds per query, while Alpha-SQL [13] uses 138.03K input tokens and 377.1 seconds per query. DeepEye-SQL further reports a 3.49× speedup on large schemas through parallel execution. These results validate the database-querying subsystem of DeepEye, while the case study below illustrates how the subsystem is composed with document retrieval and multimodal artifact generation.

5.2 DataMagic Video Generation Subsystem

Video Subsystem. To support data video generation in DeepEye, we develop DataMagic [3] as a declarative multi-agent subsystem for authoring data videos from raw tabular data. Given insights, charts, and a narration goal, the Video Generator produces a DVSpec scene sequence that binds visual elements, narration segments, and animation triggers to data references. This representation fits the DeepEye node abstraction: the outer workflow sees a typed data-video artifact, while local generation keeps scene planning, audio-visual synchronization, and rendering traces inside the node.

Video Generation Evidence. Table 3 summarizes the evidence most relevant to the Video Generator subsystem of DeepEye. On 109 real-world samples, direct LLM generation achieves average quality scores between 1.91 and 2.22, whereas our DataMagic variants reach 3.38–3.89 with execution success rates above 95%. The largest gains occur in narrative quality (1.88–2.07 to 3.16–3.84) and animation quality (1.65–1.94 to 3.79–4.39), matching the responsibilities of the Video Generator: selecting coherent scenes and synchronizing visual emphasis with narration. A within-subjects user study with 12 participants further shows that DataMagic reduces task completion time from 39.2 to 8.4 minutes, a 78.6% reduction, while increasing usability from 3.22 to 5.97.

5.3 Global Sales Analysis Case Study

Context Binding. Figure 2 illustrates DeepEye on a "Global Sales Performance Analysis" task. The user first uploads business documents to the knowledge base panel (Fig. 2-A) and structured sales files to the data source panel (Fig. 2-B). In the chat panel (Fig. 2-C), the user explicitly binds *@Financial Metrics*, *@Meta-data*, and *@Sales Database* before asking: *Help me analyze the 2025 global sales performance, generate video, dashboard, and analytical report.* This explicit binding grounds planning in user-selected resources rather than relying on implicit data selection.

Workflow Execution. The workflow agent records planning and execution progress in the process panel (Fig. 2-D), while the compiled DAG is exposed on the workflow canvas (Fig. 2-E). The generated workflow contains two independent evidence-gathering branches: a Knowledge Search node retrieves metric definitions, and a Datasource Read node retrieves database rows. The optimizer places these branches in the same execution layer, after which a Code Executor derives intermediate analytical artifacts. The downstream Video Generator, Dashboard Generator, and Report Generator consume these artifacts to produce user-facing outputs.

Human Steerability. The inspector (Fig. 2-F) exposes node-level parameters, query text, and runtime status, allowing the user to inspect or edit the workflow before downstream artifacts are finalized. If an edited node is connected to an incompatible downstream node, the validator raises a schema mismatch before execution; after correction, DeepEye re-executes only the affected downstream subgraph. The accepted workflow produces a data video (Fig. 2-G1), dashboard (Fig. 2-G2), and analytical report (Fig. 2-G3), making the end-to-end analysis both auditable and steerable.

6 RELATED WORK AND DISCUSSION

Text-to-SQL and Data Interaction. Text-to-SQL research translates natural language into executable relational queries and has made rapid progress with LLMs [12, 13]. These systems are essential for database-grounded analytics, but they typically focus on one modality and output type. In DeepEye, DeepEye-SQL provides this capability as a query subsystem whose outputs become typed workflow artifacts for downstream visualization, video generation, dashboard construction, and report writing.

Agentic Workflows. Recent agent frameworks explore planning, tool use, and workflow generation [27]. General-purpose agents often expose limited structure and rely on LLM-generated traces for correctness. DeepEye instead centers typed workflow graphs, enabling static validation, runtime optimization, provenance-aware execution, and human editing while keeping specialized reasoning inside AgentNodes. Successful workflows can be saved and adapted rather than reconstructed from chat histories.

Limitations. DeepEye is a system prototype rather than a finished benchmark suite. Current evidence covers its query and video-generation subsystems, while full heterogeneous-workflow evaluation remains ongoing. Typed ports catch structural mismatches but not semantic errors, motivating stronger validators, provenance tracking, task-specific judges, user studies, logging, recovery, and governance.

7 CONCLUSION

We presented DeepEye, a workflow-centric agentic data system that represents heterogeneous analytics as typed, inspectable workflow DAGs. By separating AgentNodes and ToolNodes, DeepEye makes planning, validation, execution, and user steering explicit system functions. Evaluations of its DeepEye-SQL and DataMagic subsystems, together with the global-sales case study, show that reliable query grounding and multimodal artifact generation can be composed in one auditable workflow. Overall, DeepEye advances data agents toward transparent, steerable, and reusable analytical systems.

REFERENCES

[1] Amazon Web Services. 2026. Amazon Simple Storage Service User Guide. https://docs.aws.amazon.com/AmazonS3/latest/userguide/ Accessed: 2026-06-06.

[2] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *CoRR* abs/2312.11805 (2023). https://doi.org/10.48550/ARXIV.2312.11805 arXiv:2312.11805

[3] Anonymous Authors. 2026. DataMagic: Authoring Data Videos through Declarative Multi-Agent Orchestration. Manuscript.

[4] Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. *CoRR* abs/2411.00073 (2024). https://doi.org/10.48550/ARXIV.2411.00073

[5] Celery Project. 2026. Celery: Distributed Task Queue. https://docs.celeryq.dev/ Accessed: 2026-06-06.

[6] Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Format Restriction, and Column Exploration. *CoRR* abs/2502.00675 (2025). https://doi.org/10.48550/ARXIV.2502.00675

[7] Binyuan Hui, Jian Yang, Zeyu Cui, et al. 2024. Qwen2.5-Coder Technical Report. *CoRR* abs/2409.12186 (2024). https://doi.org/10.48550/ARXIV.2409.12186 arXiv:2409.12186

[8] Aaron Hurst, Adam Lerer, Adam P. Goucher, et al. 2024. GPT-4o System Card. *CoRR* abs/2410.21276 (2024). https://doi.org/10.48550/ARXIV.2410.21276 arXiv:2410.21276

[9] Aishwarya Kamath, Johan Ferret, Shreya Pathak, et al. 2025. Gemma 3 Technical Report. *CoRR* abs/2503.19786 (2025). https://doi.org/10.48550/ARXIV.2503.19786 arXiv:2503.19786

[10] 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. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *ICLR*. OpenReview.net. https://openreview.net/forum?id=XmProj9cPs

[11] Patrick S. H. 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 *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[12] Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. 2026. DeepEye-SQL: A Software-Engineering-Inspired Text-to-SQL Framework. *Proc. ACM Manag. Data* 4, 3 (2026). https://doi.org/10.1145/3802035

[13] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-SQL: Zero-Shot Text-to-SQL using Monte Carlo Tree Search. In *ICML (Proceedings of Machine Learning Research)*. PMLR / OpenReview.net.

[14] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale. *Proc. VLDB Endow.* 18, 11 (2025), 4695–4709. https://www.vldb.org/pvldb/vol18/p4695-li.pdf

[15] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can LLM Already Serve as a Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. *Advances in Neural Information Processing Systems* 36 (2024).

[16] Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. 2025. XiYan-SQL: A Novel Multi-Generator Framework For Text-to-SQL. *CoRR* abs/2507.04701 (2025). https://doi.org/10.48550/ARXIV.2507.04701

[17] Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. 2025. nvBench 2.0: Resolving Ambiguity in Text-to-Visualization through Stepwise Reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[18] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal* 2014, 239 (2014), 2. https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

[19] MinIO, Inc. 2026. MinIO Object Storage Documentation. https://docs.min.io/ Accessed: 2026-06-06.

[20] PostgreSQL Global Development Group. 2026. PostgreSQL Documentation. https://www.postgresql.org/docs/ Accessed: 2026-06-06.

[21] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In *ICLR*. OpenReview.net. https://openreview.net/forum?id=CvGqMD5OtX

[22] Redis Ltd. 2026. Redis Documentation. https://redis.io/docs/latest/ Accessed: 2026-06-06.

[23] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *CoRR* abs/2405.16755 (2024). https://doi.org/10.48550/ARXIV.2405.16755

[24] Yihan Wang and Peiyu Liu. 2025. LinkAlign: Scalable Schema Linking for Real-World Large-Scale Multi-Database Text-to-SQL. *CoRR* abs/2503.18596 (2025). https://doi.org/10.48550/ARXIV.2503.18596

[25] An Yang, Anfeng Li, Baosong Yang, et al. 2025. Qwen3 Technical Report. *CoRR* abs/2505.09388 (2025). https://doi.org/10.48550/ARXIV.2505.09388 arXiv:2505.09388

[26] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*. Association for Computational Linguistics, 3911–3921.

[27] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. AFlow: Automating Agentic Workflow Generation. In *ICLR*. OpenReview.net.

[28] Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, et al. 2025. A Survey of Data Agents: Emerging Paradigm or Overstated Hype? *CoRR* abs/2510.23587 (2025).