HYVE: Hybrid Views for LLM Context Engineering over Machine Data

Jian Tan[1*], **Fan Bu**[2*], **Yuqing Gao**[2], **Devavrat Khanolkar**[2],
Jason Mackay[2], **Boris Sobolev**[2], **Lei Jin**[2], **Li Zhang**[2]
[1]Google LLC, Sunnyvale, California, USA
[2]Cisco Systems, Inc., San Jose, California, USA
jiantan@google.com {fabu,yuqingga,dkhanolk}@cisco.com
{jasmacka,bsobolev,leiji2,liz4}@cisco.com

Abstract

Machine data is central to observability and diagnosis in modern computing systems, appearing in logs, metrics, telemetry traces, and configuration snapshots. When provided to large language models (LLMs), this data typically arrives as a mixture of natural language and structured payloads such as JSON or Python/AST literals. Yet LLMs remain brittle on such inputs, particularly when they are long, deeply nested, and dominated by repetitive structure.

We present HYVE (HYbrid ViEw), a framework for LLM context engineering for inputs containing large machine-data payloads, inspired by database management principles. HYVE surrounds model invocation with coordinated preprocessing and postprocessing, centered on a session-scoped datastore augmented with schema information for reuse across related calls. During preprocessing, HYVE detects repetitive structure in raw inputs, materializes it in the datastore, transforms it into hybrid columnar and row-oriented views, and selectively exposes only the most relevant representation to the LLM. During postprocessing, HYVE either returns the model output directly, queries the datastore to recover omitted information, or performs a bounded additional LLM call for SQL-augmented semantic synthesis.

We evaluate HYVE on diverse real-world workloads spanning knowledge QA, chart generation, anomaly detection, and multi-step network troubleshooting. Across these benchmarks, HYVE maintains or improves output quality, and for data-intensive tasks it reduces model token consumption by up to 50–90%. On structured generation tasks, it improves chart-generation accuracy by up to 132% and reduces latency by up to 83%. Overall, HYVE offers a practical approximation to an effectively unbounded context window for prompts dominated by large machine-data payloads.

1 Introduction

Machine data, such as logs, metrics, telemetry traces, and configuration snapshots, is ubiquitous in modern computing systems and underpins observability. When fed into LLM prompts or carried across multi-turn interactions, it often appears as an interleaving of natural language and large structured payloads. Such inputs pose three recurring difficulties. *Token explosion:* nested keys and repeated schema consume context and crowd out useful evidence. *Context rot:* the model can miss task-relevant signals buried in a large payload or drift from the instruction. *Sequence reasoning:* long numeric and categorical sequences obscure anomalies, trends, and entity relationships needed for data analytics.

*Equal contribution. This work was conducted while the first author was with Cisco Systems, Inc.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

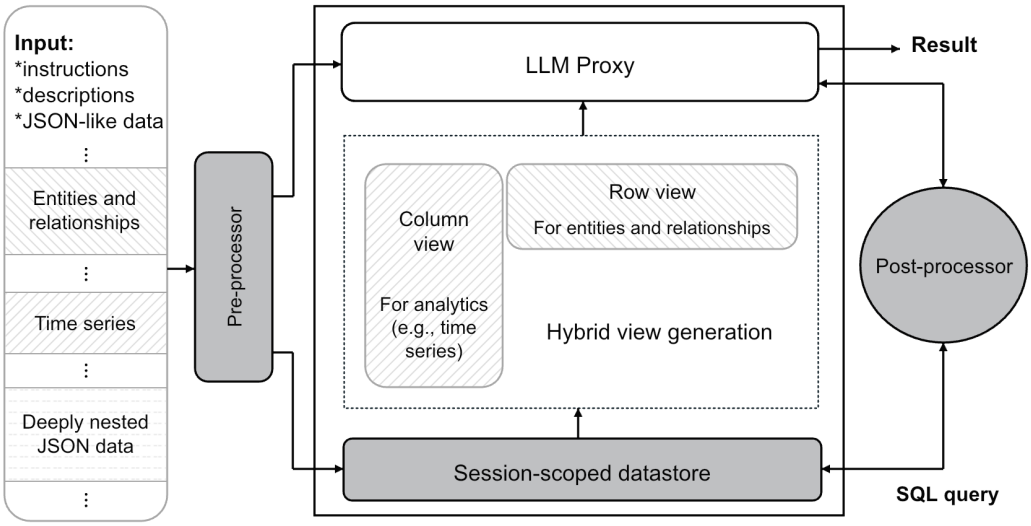


Figure 1: HYVE architecture. The preprocessor parses input strings with structured segments into hybrid tabular views. The session-scoped datastore retains the full data; a *session* is a series of related requests that share a datastore, allowing later ones to reuse previously materialized data. The postprocessor inspects LLM outputs and, when needed, queries the datastore to enrich the response.

HYVE performs context engineering for these machine-data-heavy inputs. It receives an opaque request, not a pre-existing database or schema, and discovers compatible repeated structures in embedded machine-data regions when present. Requests without useful structure to reduce can follow the direct model path. HYVE can therefore complement general-purpose agent harnesses; it is not a text-to-SQL system over an existing database.

Machine data often becomes long because system code naturally emits repeated structures: loops generate arrays, repeated keys, and nested records in the observable representation. The bottleneck is therefore not raw length alone. Effective use of such data requires *structural transformation* and *signal enhancement*, so that the same information is presented in forms better aligned with the strengths of LLMs. Empirically, we observe that, in long-context settings, LLMs still make errors when correlating multiple arrays through their indices, whereas a row-oriented view makes this easier by presenting related values side by side. Conversely, LLMs often struggle to extract values from the same field across a list of dictionaries, such as when generating a line chart, whereas a column-oriented view places those values consecutively in order.

1.1 Motivating examples

We illustrate the problem with two real examples. Figure 2 shows a slot-filling task, i.e., semantic querying over network telemetry data subject to specified requirements. Baseline GPT-4.1 uses a mean of about 5,000 tokens per request yet fails on 30.6% of samples (scores $\leq$2), showing that naively embedding raw machine data in prompts is both inefficient and unreliable. In contrast, HYVE reduces token usage by 53% while achieving near-perfect quality (mean score 4.98) on all 700 test cases. Figure 3 shows a second example: when asked to generate a line chart from 778 exchange-rate points, baseline GPT-4.1 truncates output arrays to 40–120 points, causing most of the data to disappear and collapsing detailed time series into sparse straight-line segments. With HYVE, all 778 points are recovered, producing a faithful chart. Together, these examples suggest that HYVE is effective across a broad range of machine-data analytics tasks; Appendix E.4 provides the full benchmark descriptions, and Appendix E.8 gives detailed per-task analysis.

1.2 Guiding principles

HYVE is guided by database-inspired principles for analytics-oriented context engineering. It treats prompt construction not as ad hoc compression, but as a disciplined process of structure discovery, information preservation, and queryable representation.

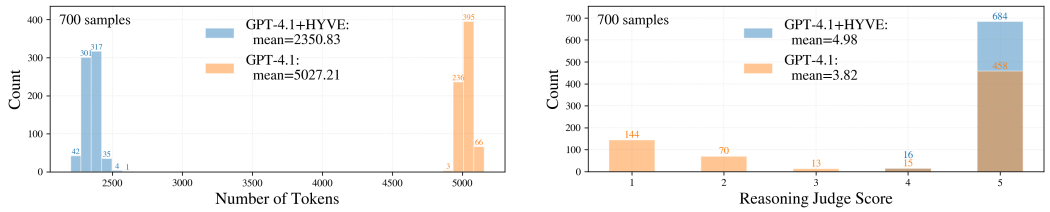


Figure 2: An illustrative 700-sample slot-filling run, separate from the 4,096-example Canvas aggregate in Table 1. Left: baseline GPT-4.1 consumes ∼5000 tokens per request; HYVE reduces usage by 53%. Right: baseline GPT-4.1 fails on 30.6% of samples (scores ≤2); HYVE achieves almost perfect scores on all samples.

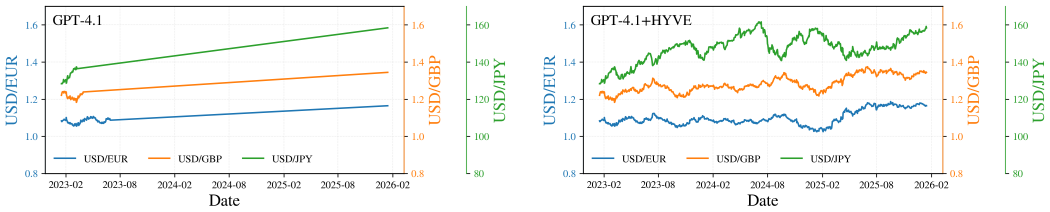


Figure 3: Line chart over three years of USD exchange-rate data (778 points per series). Baseline GPT-4.1 truncates output arrays to 40–120 points, collapsing detailed time series into sparse straight-line segments. HYVE recovers all 778 points through data backfilling.

HYVE applies complementary row/column organization to a new setting - context engineering for machine-data-heavy requests. It introduces an end-to-end harness architecture that coordinates structure discovery, hybrid views, session-scoped state, controlled querying, bounded recovery, and safe fallback under a single design invariant: reduce visible context without discarding recoverable evidence.

- **Guarantee data fidelity:** Context reduction must not lose information. Every part of the raw input is either preserved verbatim in the visible LLM context or retained in a session-scoped datastore with explicit schema information, so that omitted content remains fully recoverable.
- **Avoid semantic compression:** Context reduction does not rely on LLM-based summarization, which may be ineffective for machine data. Instead, HYVE detects repetitive structures, organizes them into tables with precise schema, and exposes only a subset of entries selected by statistical ranking, while explicitly indicating that additional data remain available in the datastore. When hidden data are needed, the LLM can generate SQL over the provided schema, after which the postprocessor either backfills the result into a template or performs one additional synthesis step.
- **Limit LLM calls:** HYVE is not an open-ended iterative agent. Instead, it operates as a service that preserves the same interface as a standard LLM endpoint. Beyond the primary LLM call, the system makes at most one additional LLM call by default; this can be configured to a small fixed number for SQL refinement, and is only needed when semantic synthesis over the retrieved evidence is required.

Three requirements guide the design: retain omitted evidence in the datastore; expose ordered same-field sequences and whole-record relationships through column and row views; and recover missing information through deterministic backfilling or bounded synthesis based on session-scoped queryable state.

Accurately generating SQL in a general open-ended environment remains challenging for LLMs. Our setting, however, is controlled: the preprocessor and postprocessor are tightly coupled around a shared datastore. HYVE exploits the local context to guide the LLM toward generating high-quality SQL queries tailored to each request.

HYVE can complement general-purpose tool-using agents; Appendix E.16 compares their scopes.

1.3 "Everything Is a File," and Some Are Databases

Many successful harness systems make extensive use of a virtual file-system abstraction, often summarized as "everything is a file." This abstraction has worked remarkably well. In principle, almost any data operation can be built on files. Yet database systems emerged because a higher level of abstraction and a generic structured query language matter: some inputs could be treated not merely as files, but as databases. HYVE implements this lifecycle by discovering compatible structures inside an opaque mixed request, retaining the complete structured payload in session-scoped queryable state, and exposing complementary row/column views. Crucially, all of these steps are coordinated coherently according to the detected structure and semantics of the request itself. Appendix C.6 expands the comparison.

1.4 High-level design

As illustrated in Figure 1, HYVE wraps a primary LLM call with a preprocessor and a postprocessor. By default, the postprocessor may trigger at most one additional LLM call.

The *preprocessor* performs three tasks: (1) parse the raw input and detect repetitive structures in embedded JSON/AST literals; (2) build or reuse an in-memory datastore that retains the full data together with inferred schema; and (3) transform the data into hybrid column and row views, exposing only a selected subset of table entries to the LLM. The *postprocessor* then operates in one of three modes: (1) return the LLM output directly when the visible context is sufficient; (2) query the datastore and backfill the retrieved data into an LLM-generated template via schema-aware mappings; or (3) query the datastore, append the results to the context, and make one additional LLM call to synthesize the final answer.

The result is a compact prompt that still preserves task-critical information: although some table entries are hidden from the visible context, the schema, selected entries, surrounding instructions, and non-tabular key-value pairs remain available to the model. Appendix C.6 discusses how HYVE relates to agent design and database-style management of structured context.

2 Main design

We formalize analytics-oriented context engineering for LLM inputs containing embedded semi-structured segments, using hybrid views and an on-demand delayed query process implemented through coupled preprocessors and postprocessors around an LLM call.

2.1 Preprocessor design

The preprocessor uses a robust parser to detect repetitive structures together with their schemas, transform them into column and row views, and selectively expose only a subset of table entries to the LLM. The full dataset remains available in a queryable datastore.

Raw input The input is a single opaque string s[2] that may contain an arbitrary mixture of natural language and machine data, such as job descriptions, instructions, command outputs, log fragments, and configuration snippets. No explicit structural boundary markers are guaranteed: JSON or AST literals may appear inline, span multiple lines, be partially malformed, or be string-encoded within other JSON fields.

Parsing and structure detection A resilient parser decomposes s into an interleaved sequence of free-form text segments and structured objects:

$$\mathsf{Parse}(s) = (t_1, J_1, t_2, J_2, \ldots, t_m, J_m, t_{m+1}),$$

where each t_i is a text segment and each J_i is a JSON or Python literal object. The parser handles deeply nested objects, heterogeneous arrays, partially malformed JSON, string-encoded nested blobs,

[2] In practice, inputs often contain semi-structured objects, for example as JSON messages conforming to the interaction schema of the target LLM service. Before inference, the provider serializes this structure into a single token stream, potentially inserting vendor-specific control or delimiter tokens. As this serialization detail does not affect the design of HYVE, we omit it for simplicity.

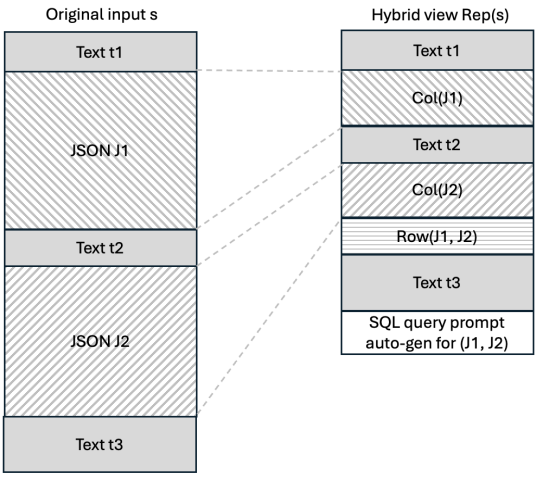


Figure 4: Layout of $\mathsf{Rep}(s)$ showing preserved text segments, transformed columns, and the row view. If table truncation is triggered, SQL query prompt is automatically generated.

and Python AST literals (Appendix D). We write $\mathbf{J} = (J_1, \ldots, J_m)$ for the extracted objects and $\mathbf{t} = (t_1, \ldots, t_{m+1})$ for the text segments.

Using JSON's key-value structure together with JSONPath [1], a standard path notation for addressing values inside nested JSON objects, we identify *repetitive patterns*, such as collections of dictionaries or lists that share a common JSONPath prefix and a common inferred schema. For example, in a JSON object whose field `items` stores a list of records, the path `$.items[0].name` refers to the `name` field of the first record, starting from the root node `$` and traversing through the `items` field. These patterns can be organized into tables with well-defined column names and value types; they form the basis of both the hybrid view and the queryable datastore. We use JSONPath notation as a compact way to refer to values inside nested JSON objects, and write $\mathsf{Val}(J, p)$ for the value at path p when it exists.

The key requirement is that entries grouped into the same table or column family must be structurally compatible. HYVE therefore uses a recursive schema-consistency check before aligning repeated records: fields are merged only when their key structure and compatible leaf types match. Formal definitions of schema consistency, column-oriented views, and row-oriented views are given in Appendix D.1.

Representation operator Let $\mathsf{Tok}(\cdot)$ denote the token-count function under the target tokenizer. The system transforms the raw input s into a prompt string π that preserves sufficient information for the LLM while substantially reducing token count. We denote this preprocessing step by the *representation* operator

$$\mathsf{Rep} : s \mapsto \pi,$$

which operates on s via $\mathsf{Parse}(s)$. It is governed by subset-selection parameters, such as the number of list items retained per array and the number of top-ranked records surfaced in the row view. These parameters can substantially reduce the visible context when a request contains large repeated structures, while requests with little compressible structure change less. Section 3 reports the resulting end-to-end model-token usage.

Hybrid View For each object J_i, the preprocessor generates an ordered column view $\mathsf{Col}(J_i)$. The sequence $\mathsf{Col}(J_1), \mathsf{Col}(J_2), \ldots$ is interleaved with the text segments $\mathbf{t}$ in original order. It also generates a combined, unordered row view $\mathsf{Row}(\mathbf{J})$ across all extracted objects. As illustrated in Figure 4, the resulting prompt $\mathsf{Rep}(s)$ is assembled as the following *hybrid view*:

$$\mathsf{Rep}(s) = t_1 \oplus \mathsf{Col}(J_1) \oplus t_2 \oplus \mathsf{Col}(J_2) \cdots \oplus t_m \oplus \mathsf{Col}(J_m) \oplus \mathsf{Row}(\mathbf{J}) \oplus t_{m+1},$$

where $\oplus$ denotes string concatenation.

The two views serve complementary purposes. The column view groups homogeneous scalar values by repeated context and field path, preserving traver-

sal order so the model can inspect value sequences directly. For example, `clusters[].nodes[].metrics[].cpu` and `clusters[].nodes[].metrics[].mem` share the repeated context `clusters[].nodes[].metrics[]` but form different columns. The row view instead keeps all fields of each schema-compatible repeated record together, so entity-level correlations such as which `status` belongs to which `id` remain explicit. Together, these views support both analytics-oriented extraction and retrieval-oriented correlation while avoiding unsafe merges across schema-inconsistent regions. Formal definitions and a detailed illustration of these hybrid views are given in Appendix D.1.

Table Entry Truncation The challenge is that both $\mathsf{Col}(J_i)$ and $\mathsf{Row}(\mathbf{J})$ may still be too large to expose in full. HYVE therefore truncates only *table entries* while preserving schema. For the column view, the retained subset must preserve the original order to maintain semantic continuity. For the row view, ordering is not needed, because its primary purpose is to expose representative entities and their relationships.

HYVE uses reference-aware term-weighting methods from information retrieval to decide which entries in $\mathsf{Col}(J_i)$ and $\mathsf{Row}(\mathbf{J})$ are most relevant to the task. To do so, it constructs a *query string* q by concatenating the surrounding text segments $t_1, \ldots, t_{m+1}$, which typically contain the user's question or instruction, with a slim representation $\mathbf{Slim}(\mathbf{J})$ that retains all key-value pairs outside the detected tables. The role of $\mathbf{Slim}(\mathbf{J})$ is important: non-tabular key-value pairs often encode request-level semantics that are needed to rank table entries correctly.

For $\mathsf{Col}(J_i)$, HYVE keeps a fixed-length prefix and postfix plus items selected by a reference-aware ranking score with respect to q, preserving their original order. For $\mathsf{Row}(\mathbf{J})$, it ranks complete records by the same method and presents a subset that exposes entity-level correlations without preserving record order.

To preserve access to the full dataset, the preprocessor uses a *datastore* $\mathcal{D}(\mathbf{J})$, in which the untruncated data are loaded into in-memory relational tables with schemas inferred from the input, including column names, value types, and parent-child foreign keys for nested arrays. Here $\mathcal{D}(\mathbf{J})$ denotes data materialized from the parsed objects in the shared datastore. Specifically, the repeated-context path c helps determine the relational table to which a value belongs, while the flattened field path p determines the corresponding field name; nested arrays are normalized into parent-child tables.

2.2 Datastore and SQL Query

When table entry truncation is triggered, HYVE automatically injects tool-call prompts to guide SQL generation.

SQL Prompt Auto-Generation Reliable SQL generation remains challenging for LLMs in open-ended settings, but our environment is controlled and provides exact knowledge of the schema and parent-child relationships among tables. HYVE synthesizes a schema-aware tool prompt from $\mathcal{D}(\mathcal{J})$, listing the available tables, columns, parent-child keys, and the output contract of the permitted tool call. The same query mechanism can use data retained from earlier calls in the session. It also constructs targeted few-shot examples to guide the LLM toward generating high-quality SQL queries for each request.

Because these prompts are generated per request and may include corner-case details, HYVE adopts a gradual-disclosure design. In the primary LLM call, the model only needs to identify the task intent and choose the appropriate tool. When a follow-up call is needed, HYVE provides more contextualized prompts to guide SQL generation and data backfilling. Thus, HYVE incurs at most one additional LLM call by default.

In practice, such guidance can be supplied, for example, through the `instructions` field in the OpenAI Responses API [2] or the Chat Completions API [3]; alternatively, the same design can be implemented through Anthropic's API or its OpenAI SDK compatibility layer [4, 5].

A concrete prompt/datastore example showing $\mathsf{Rep}(s)$ and the injected schema metadata is provided in Appendix C.4.

2.3 LLM Call and Postprocessor

If the visible prompt is insufficient for the primary LLM call, HYVE completes the result through either template-based data backfilling or query-based evidence augmentation.

LLM Output Given a prompt π, the LLM generates $\mathsf{LLM}(\pi) \rightarrow y$, where y may be either a free-form answer or a structured object conforming to a schema, such as a tool-call output or a visualization specification.

Postprocessor Working Modes If the visible prompt contains sufficient evidence, the LLM operates directly on π and produces the final result y; no datastore access or postprocessing is required. If needed evidence is retained in the session datastore but not visible to the LLM, HYVE synthesizes a schema-aware SQL prompt from $\mathcal{D}$, enabling the model to emit SQL statements that are executed by the postprocessor. This yields three operating modes:

- *Mode 1 (Direct Output).* When the visible context is sufficient, for example for a point lookup, the LLM output y is returned directly.
- *Mode 2 (Template Backfill).* This mode corresponds to the `GenTemplateAndBackfill` branch introduced above and targets rendering tasks in which the LLM first generates the schema of a structured answer, typically under a JSON specification. HYVE uses gradual disclosure in this branch: the primary prompt presents `GenTemplateAndBackfill` only as a routing choice, while the detailed `BackfillData` tool contract is revealed only after Mode 2 has been selected. The follow-up prompt then instructs the LLM to produce both the answer template and a `BackfillData` tool call containing SQL queries together with column-to-path mappings. The postprocessor executes the query against $\mathcal{D}$ and injects the full result into the template according to those mappings. In this way, the visible portion is generated by the LLM, while the hidden portion is restored through template-based expansion that preserves the model-generated schema.
- *Mode 3 (SQL-Augmented Synthesis).* This mode corresponds to the `QueryDatastore` branch introduced above. Unlike Mode 2, the primary prompt already includes the full `QueryDatastore` tool specification, so no second tool-disclosure step is required. When the task requires semantic synthesis over both the queried data and the natural-language prompt, such as aggregation, counting, filtering, or joins across multiple tables, the primary LLM emits a `QueryDatastore` tool call containing one or more SQL queries. The postprocessor executes these queries against $\mathcal{D}$, appends the formatted results to the conversation context, and makes one bounded additional LLM call to produce a synthesized answer grounded in the complete evidence.

Under the default design, HYVE requires only the primary LLM call in Mode 1 and at most one additional LLM call in Modes 2 and 3. As discussed in Appendix C.3, Mode 3 can also be configured to allow a bounded number of additional calls to improve the fidelity of the final answer. A concrete Mode 2 example showing template generation, SQL queries, and column-to-path backfilling is given in Appendix C.5. Full system architecture and implementation details are provided in Appendix D; operator-level details for ranking, rendering, and time-series analysis are provided in Appendix E.

3 Experimental Evaluation

We evaluate HYVE on a diverse collection of real-world networking datasets involving machine data and structured outputs. Most of these datasets are proprietary, so we describe them in the appendix. We also evaluate HYVE on the public TOON-QA benchmark [6]. The TOON-QA column uses all 154 questions from the official four-dataset release, rendered as beautified JSON; Appendix E.4 describes its coverage and the v4.1.0 release.

Detailed dataset descriptions are provided in Appendix E.4. The performance metrics and evaluation methodologies are provided in Appendix E.7.

Table 1: Comparison of answer quality, token usage, and latency across benchmarks

	Model	Cert-QA^{G}	Runbook^{G}	Line^{S}	Bar^{S}	Anom^{R}	Sum^{G}	Canvas^{R}	RB-Text^{G}	RB-JSON^{R}	Hard^{R}	TOON-QA^{E}
Score↑	GPT-5	**4.49**	4.55	0.68	0.85	3.28	3.82	4.96	4.67	4.84	4.33	0.96
	GPT-5 + HYVE	**4.49**	**4.58**	**0.97**	**1.00**	**3.77**	**3.85**	**4.96**	**4.77**	**4.92**	**5.00**	**0.98**
	GPT-4.1	4.36	4.26	0.41	**0.99**	3.22	4.43	4.94	4.88	4.78	4.04	0.47
	GPT-4.1 + HYVE	**4.38**	**4.32**	**0.95**	**0.99**	**4.03**	**4.46**	**4.95**	**4.89**	**4.92**	**5.00**	**0.93**
	Claude Sonnet 4.5	**4.39**	4.33	0.33	0.71	3.16	4.08	4.97	4.67	4.78	3.87	0.49
	Claude Sonnet 4.5 + HYVE	**4.39**	**4.37**	**1.00**	**0.97**	**3.44**	**4.14**	4.96	**4.71**	**4.88**	**5.00**	**0.99**
Token↓	GPT-5	**3.1M**	1.1M	4.2M	1.8M	18.5M	381.3K	122.8M	71.7K	142.8K	80.5K	1.4M
	GPT-5 + HYVE	3.1M	**1.1M**	**748.0K**	**251.8K**	**10.9M**	**372.4K**	**38.2M**	**56.8K**	**132.7K**	**20.3K**	**182.4K**
	GPT-4.1	**1.1M**	289.7K	3.0M	1.5M	15.8M	209.2K	123.4M	61.5K	116.1K	75.1K	1.3M
	GPT-4.1 + HYVE	1.1M	**286.9K**	**369.4K**	**132.5K**	**8.9M**	**206.0K**	**35.1M**	**48.4K**	**107.3K**	**15.1K**	**153.8K**
	Claude Sonnet 4.5	1.3M	401.1K	3.5M	1.8M	16.6M	253.2K	128.0M	76.2K	137.5K	84.5K	1.5M
	Claude Sonnet 4.5 + HYVE	1.3M	**400.2K**	**1.4M**	**931.3K**	**15.9M**	**246.7K**	**52.1M**	**68.2K**	**122.2K**	**29.9K**	**800.4K**
Lat.(s)↓	GPT-5	8.10	38.89	125.11	75.27	28.73	23.66	10.45	20.60	7.39	19.45	5.48
	GPT-5 + HYVE	**7.34**	**36.79**	**39.79**	**12.73**	**20.71**	**22.23**	**8.99**	**12.73**	**7.19**	**16.96**	**2.71**
	GPT-4.1	3.21	7.58	47.86	32.91	7.49	5.57	3.22	**2.74**	2.40	8.07	**1.35**
	GPT-4.1 + HYVE	**2.91**	**7.18**	**8.93**	**3.11**	**4.38**	**4.99**	**3.00**	4.53	**2.11**	**5.43**	2.33
	Claude Sonnet 4.5	7.25	24.00	100.63	87.62	21.03	15.48	3.21	7.97	6.55	13.28	1.91
	Claude Sonnet 4.5 + HYVE	**6.99**	**23.60**	**23.73**	**9.53**	**18.75**	**15.01**	3.87	**7.78**	**6.42**	**11.43**	3.26

GPT-5 / GPT-4.1: Baseline models with standard JSON serialization; structured data passed directly to the LLM without optimization. All model–method conditions were evaluated in ten runs; the table reports means (Appendix E.5). **+ HYVE**: Models augmented with the full HYVE pipeline (hybrid-view transformation, reference-guided truncation, queryable data backfilling and bounded SQL-augmented reasoning). ^{S}Similarity (max 1.0); ^{E}ExactMatch (max 1.0); ^{G}GenericJudge, ^{R}ReasoningJudge (max 5.0). Token values in K (thousands) or M (millions) with 1 decimal precision.

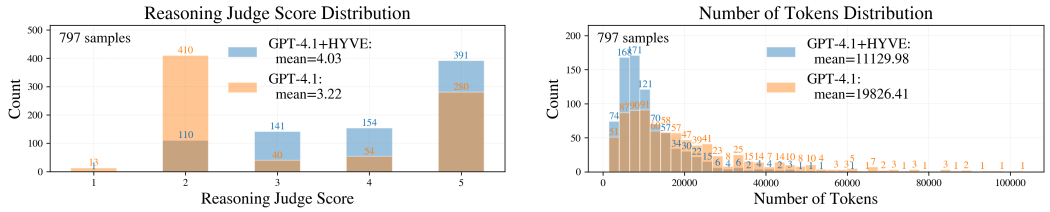


Figure 5: Per-sample distributions on the Anom dataset (n=797). HYVE improves reasoning quality while reducing token consumption through the anomaly operator's high-signal summarization.

3.1 Baselines

We compare HYVE with Raw GPT-4.1, GPT-5, and Claude Sonnet 4.5. Raw models receive standard JSON-serialized input without HYVE recovery. All conditions in Table 1 were evaluated in ten runs, and the table reports their means; Appendix E.5 provides the evaluation details.

We conduct ablation experiments to isolate the contribution of individual HYVE components (truncation strategies, rendering formats, and postprocessing recovery); these are detailed in Appendix E.14.

3.2 Main Results

Table 1 compares three model families across 11 task groups. On several large-payload tasks, HYVE reduces total model-token use by about 50–90% while maintaining or improving quality; savings are smaller on moderate-context QA and summarization. Chart generation and structured reasoning show the largest GPT-family gains: GPT-5 Line similarity rises from 0.68 to 0.97 and GPT-4.1 Line from 0.41 to 0.95. Appendix E.8 gives per-task analysis.

Structured reasoning For TOON-QA [6], GPT-4.1 exact match rises from 0.47 to 0.93 with 88% fewer tokens. In the Sonnet rows, quality improves or differs by at most 0.01 across the 11 groups, while Canvas and TOON-QA incur additional latency. Figure 5 illustrates the quality and token trends on Anom. Appendix E.9, E.11, E.10, and E.12 give the distinct TOON release, configured agent baselines, public-benchmark experiments, and independent-judge results.

Table 2: Ablation study of HYVE with GPT-4.1

	Variant	Bar^S	Anom^R	Canvas^R	RB-Text^G	Hard^R	TOON-QA^E
Score↑	Full	**0.99**	**4.03**	**4.95**	**4.89**	**5.00**	**0.93**
	Full − rank	**0.99**	**4.03**	**4.95**	4.56	**5.00**	0.91
	Full − trunc	0.73	3.83	4.93	4.56	4.10	0.87
	Full + TOON	0.97	4.02	4.85	4.67	**5.00**	0.92
	Full − post	0.06	3.41	4.94	4.78	3.93	0.38
Token↓	Full	132.5K	8.9M	35.1M	48.4K	15.1K	153.8K
	Full − rank	115.1K	8.9M	37.8M	48.5K	16.2K	255.8K
	Full − trunc	928.7K	15.7M	118.8M	57.8K	74.7K	361.2K
	Full + TOON	**91.7K**	8.9M	**28.9M**	45.5K	**12.8K**	143.2K
	Full − post	132.3K	**6.2M**	34.2M	**33.7K**	14.8K	**94.5K**
Lat.(s)↓	Full	3.11	4.38	3.00	4.53	5.43	2.33
	Full − rank	**2.65**	4.91	4.12	**2.39**	8.54	2.56
	Full − trunc	24.48	5.73	3.33	3.31	10.80	1.91
	Full + TOON	3.94	4.76	3.03	3.27	5.39	1.86
	Full − post	3.14	**3.20**	**2.98**	2.68	**5.25**	**0.97**

Each variant changes one component; all others remain at their defaults.
Full: Complete HYVE pipeline (Mode 1+2+3).
Full − rank: Ranking disabled; postprocessing SQL queries remain enabled; truncation keeps only the first k elements (prefix-only).
Full − trunc: Truncation disabled; full-length arrays passed to the LLM.
Full + TOON: TOON encoding [6] instead of beautified JSON.
Full − post: Postprocessing disabled; preprocessing, including ranking, remains enabled; output is taken directly from the LLM.
^{S}Similarity (max 1.0); ^{E}ExactMatch (max 1.0); ^{G}GenericJudge, ^{R}ReasoningJudge (max 5.0). Tokens in K/M.

3.3 Ablation Study

We conduct ablation studies to isolate the contribution of each HYVE component. Table 2 organizes ablations along two dimensions. *Preprocessing ablation*: we selectively disable ranking (Full − rank), truncation (Full − trunc), or swap the serialization format (Full + TOON), while keeping the full postprocessing pipeline (Mode 1+2+3) active. *Postprocessing ablation*: we disable all postprocessing (Full − post). We evaluate on tasks where these components have measurable impact: chart generation (Bar), anomaly detection (Anom), slot-filling (Canvas), text summarization (RB-Text), multi-hop reasoning (Hard), and structured-data question answering (TOON-QA).

Detailed ablation analysis is provided in Appendix E.14. Appendix E.15 discusses the complementary direction of combining HYVE with domain-adaptive training. Appendix Table 9 isolates Modes 2 and 3 using matched controls; Appendix E.13 reports repeated-run uncertainty and preprocessing sensitivity. Near-ceiling GPT-4.1 Cert-QA and GPT-5 Runbook intervals include zero.

4 Related Work and Limitations

4.1 Related Work

DATER [7] and TAP4LLM [8] start with supplied tables; Table Meets LLM [9] studies table presentation; FACTS [10] assumes a known schema; SchemaRAG [11] a predefined output schema; and LLMLingua/LongLLMLingua [12, 13] compress prompts. HYVE instead discovers structure in opaque mixed requests and coordinates queryable retention, row/column views, and recovery. To the best of our knowledge, prior work does not jointly provide these properties for context engineering of machine-data-heavy requests. Appendix E.16 details the distinctions.

4.2 Limitations

Short-Term vs. Long-Term Memory: HYVE maintains a queryable datastore across related calls within a session. By contrast, long-term memory systems such as LongMem [14], MemGPT [15], and Mem0 [16] maintain or retrieve information across longer horizons using external memory stores, controllers, or retrieval policies. These approaches are orthogonal to HYVE: HYVE focuses on faithful within-session organization, truncation, and recovery of machine data, while long-term memory focuses on persistence across sessions. A natural future direction is to combine the two.

Beyond JSON Objects: HYVE assumes that repetitive input structures are already represented as JSON objects or Python/AST literals. However, some system logs contain structured components embedded in free-form text. A useful extension would be to automatically convert such components into JSON so HYVE can process them more effectively.

5 Conclusion

HYVE applies structural, recoverable context reduction to machine-data-heavy requests: row/column views expose selected evidence, a session-scoped datastore retains exact values, and bounded postprocessing retrieves what the primary response needs. On several large-payload workloads, this context-reduction path cuts model-token use by about 50–90% while maintaining or improving quality. The no-truncation ablations motivate future model–harness co-design for more effective use of structured evidence, not merely larger visible context.

Acknowledgements and Disclosures

Funding. This work was fully funded by Cisco Systems, Inc.

Competing interests. The authors declare no competing interests.

References

[1] JSONPath: Query Expressions for JSON. `https://www.rfc-editor.org/rfc/rfc9535`, 2024.

[2] OpenAI API Reference: Responses. `https://platform.openai.com/docs/api-reference/responses`, 2026. Accessed: 2026-03-25.

[3] OpenAI API Reference: Chat Completions. `https://platform.openai.com/docs/api-reference/chat/create-chat-completion`, 2026. Accessed: 2026-03-25.

[4] Anthropic. Anthropic api: Messages examples. `https://docs.anthropic.com/en/api/messages-examples`, 2026. Accessed: 2026-03-25.

[5] Anthropic. Openai sdk compatibility. `https://docs.anthropic.com/en/api/openai-sdk`, 2026. Accessed: 2026-03-25.

[6] TOON Format Contributors. TOON: Token-oriented object notation. `https://github.com/toon-format/toon`, 2025. Includes the TOON Retrieval Accuracy Benchmark. Accessed: 2026.

[7] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, pages 174–184, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394086. doi: 10.1145/3539618.3591708. URL `https://doi.org/10.1145/3539618.3591708`.

[8] Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. TAP4LLM: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10306–10323, 2024. doi: 10.18653/v1/2024.findings-emnlp.603. URL `https://aclanthology.org/2024.findings-emnlp.603/`.

[9] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. Table meets LLM: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654, 2024. doi: 10.1145/3616855.3635752.

[10] Ye Yuan, Mohammad Amin Shabani, and Siqi Liu. FACTS: Table summarization via offline template generation with agentic workflows. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 9298–9315, 2026. doi: 10.18653/v1/2026.findings-acl.453. URL https://aclanthology.org/2026.findings-acl.453/.

[11] Sin Yu Bonnie Ho, Arlie Coles, Erik Larsson, Eric Marshall, Nathan Bodenstab, and Paul Vozila. SchemaRAG: Dynamic large schema reduction for LLM-driven structured information extraction. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 1114–1127, 2026. doi: 10.18653/v1/2026.acl-industry.78. URL https://aclanthology.org/2026.acl-industry.78/.

[12] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, 2023. doi: 10.18653/v1/2023.emnlp-main.825.

[13] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

[14] Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *arXiv preprint arXiv:2306.07174*, 2023.

[15] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[16] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[17] Anthropic. Claude code overview. https://docs.anthropic.com/en/docs/claude-code/overview, 2026. Official documentation. Accessed: 2026-03-30.

[18] Vercel. Building filesystem agents. https://vercel.com/academy/filesystem-agents, 2025.

[19] Hannes Mühleisen and Mark Raasveldt. Duckdb: An embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, 2019.

[20] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

[21] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[22] LangChain, Inc. LangSmith. https://www.langchain.com/langsmith, 2023. Accessed: 2026.

[23] Liyao Li, Jiaming Tian, Hao Chen, Wentao Ye, Chao Ye, Haobo Wang, Ningtao Wang, Xing Fu, Gang Chen, and Junbo Zhao. LongTableBench: Benchmarking long-context table reasoning across real-world formats and domains. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 11927–11965, 2025. doi: 10.18653/v1/2025.findings-emnlp.638.

[24] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.398. URL https://aclanthology.org/2020.acl-main.398/.

[25] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, 2020. doi: 10.18653/v1/2020.acl-main.745.

[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 Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

[27] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[28] Zhao Tan, Xiping Liu, Qing Shu, Xi Li, Changxuan Wan, Dexi Liu, Qizhi Wan, and Guoqiong Liao. Enhancing text-to-SQL capabilities of large language models through tailored promptings. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6091–6109, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.539/.

[29] Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6342–6353, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.391. URL https://aclanthology.org/2023.emnlp-main.391/.

[30] OpenClaw. Openclaw. https://openclaw.ai/, 2026. Official website. Accessed: 2026-03-30.

[31] OpenAI. OpenAI Codex CLI – Getting Started. https://help.openai.com/en/articles/11096431, 2026. Official help documentation. Accessed: 2026-03-30.

[32] Google. Gemini CLI. https://github.com/google-gemini/gemini-cli, 2026. Official repository. Accessed: 2026-03-30.

[33] OpenCode. OpenCode. https://opencode.ai/, 2026. Official website. Accessed: 2026-03-30.

[34] Pi. pi.dev. https://buildwithpi.com/, 2026. Official website for the Pi coding agent. Accessed: 2026-03-30.

[35] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

[36] Jinyang Li et al. SWE-SQL: Illuminating LLM pathways to solve user SQL issues in real-world applications. In *Advances in Neural Information Processing Systems*, 2025.

[37] Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. NL2SQL-BUGs: A benchmark for detecting semantic errors in NL2SQL translation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025. doi: 10.1145/3711896.3737427.

[38] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, 2021. doi: 10.18653/v1/2021.emnlp-main.779.

[39] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7 (PLDI):1946–1969, 2023.

[40] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.674. URL `https://aclanthology.org/2023.emnlp-main.674/`.

[41] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022.

[42] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. C-store: a column-oriented dbms. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 553–564. VLDB Endowment, 2005. ISBN 1595931546.

[43] Chao Zhang, Guoliang Li, Jintao Zhang, Xinning Zhang, and Jianhua Feng. Htap databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6410–6429, 2024.

[44] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. Tidb: a raft-based htap database. *Proceedings of the VLDB Endowment*, 13(12):3072–3084, 2020.

[45] Jiacheng Yang, Ian Rae, Jun Xu, Jeff Shute, Zhan Yuan, Kelvin Lau, Qiang Zeng, Xi Zhao, Jun Ma, Ziyang Chen, et al. F1 lightning: Htap as a service. *Proceedings of the VLDB Endowment*, 13(12):3313–3325, 2020.

This appendix presents representative benchmark examples, evaluation prompts, implementation details, and additional experimental analyses. Unless otherwise noted, code and prompt snippets are shortened for presentation.

A Representative Dataset Examples

A.1 Cert-QA Dataset

This appendix provides representative examples from each certification level in the Cert-QA dataset.

A.1.1 Entry-Level Example

Question: What command is used on a Windows PC to display IP-to-MAC address mappings?

Answer: `arp -a`

A.1.2 Advanced-Level Example

Question: What is Generic Routing Encapsulation (GRE), and what was its original purpose?

Answer: GRE is a tunneling protocol that encapsulates packets over an IP-based network. It was originally created to provide transport for non-routable legacy protocols (like IPX) across an IP network.

A.1.3 Expert-Level Example (Open-Ended)

Question: Clients of **[Redacted]** need to be able to reach several different applications to provide access to services such as voicemail, meetings, directories, and other functions. Which profile must be configured to provide these services?

Answer: Service profile

A.1.4 Expert-Level Example (Multiple-Choice)

Question: What is the deployment model of **[Redacted]**?

(a) as a plug-in in **[Redacted]**
(b) in the public cloud (SaaS)
(c) on-premise as a dedicated appliance
(d) on-premise as a virtual machine

Answer: (b) in the public cloud (SaaS)

Explanation: [Redacted] is offered as a Software as a Service product. The service is deployed in the provider cloud and other deployment options are not possible.

A.1.5 Expert-Tiered Example (Advanced Topics)

Question: In BGP implementations, what attribute is used to influence inbound traffic from neighbouring ASes?

Answer: AS-PATH prepending.

A.2 Runbook Dataset

This appendix provides a representative example from the Runbook dataset.

A.2.1 StackWise Upgrade Troubleshooting

Problem Description:

1. Install mode in stackwise standard upgrade procedure.

2. Install mode in stackwise, if one machine (non-active switch) prompts v-mismatch. How to correctly upgrade this device?
3. If using a 4-switch stack, but these 4 switches have different versions, can all switches be upgraded directly to the specified version through the active switch?

Ground-Truth Runbook (excerpt):

StackWise Upgrade Troubleshooting

Summary: This playbook outlines the troubleshooting steps for upgrading a Catalyst 9300 series switch stack. The focus is on addressing potential issues encountered during the upgrade process, such as version mismatches and ensuring successful upgrades across all stack members.

1. Initial Assessment:

- Access the Active Switch CLI: Establish connectivity to the active switch within the stack. Use Telnet or SSH to access the CLI.
- Verify Stack Status: Execute `show stack status` to confirm the operational state of the stack.
- Check Software Versions: Run `show version` on the active switch to identify the current software version.

2. Version Mismatch Resolution:

- If a non-active switch shows v-mismatch, use `install add file <image> activate commit` from the active switch.
- The active switch will propagate the image to mismatched members automatically.

3. Multi-Version Stack Upgrade:

- Yes, all switches can be upgraded from the active switch using install mode.
- Execute `install add file flash:<image> activate commit` to upgrade all stack members simultaneously.

A.3 Line Chart Dataset

This appendix provides a representative example from the Line chart dataset.

A.3.1 Line Chart Example

Input Data (excerpt):


```
[
  {"endTs": "2025-06-14T22:32:00Z",
   "jitter": 0.23, "goodput": 100000,
   "startTs": "2025-06-14T22:31:00Z",
   "latencyMs": 9.07, "lossPercent": 0},
  {"endTs": "2025-06-14T22:34:00Z",
   "jitter": 0.11, "goodput": 100000,
   "startTs": "2025-06-14T22:33:00Z",
   "latencyMs": 9.02, "lossPercent": 0},
  ...
]
```


Expected Output (excerpt):


```
{"data": {
  "name": "CDSAILineChart",
  "props": {
    "id": "[Redacted]",
    "axes": {"xAxis": {"type": "time"},
             "yAxis": {"format": {}}},
    "data": [
      {"id": "loss", "data": [
        {"x": "2025-06-14T22:31:00Z", "y": 0},
        {"x": "2025-06-14T22:33:00Z", "y": 0},
        ...]},
      {"id": "latency", "data": [
        {"x": "2025-06-14T22:31:00Z", "y": 9.07},
        ...]}
    ]}}}
```

A.4 Bar Chart Dataset

This appendix provides a representative example from the Bar chart dataset.

A.4.1 Bar Chart Example

Input Data (excerpt):


```
[
  {"peer": 59, "expansionism": "sediment"},
  {"peer": 98, "expansionism": "nonsense"},
  {"peer": 81, "expansionism": "cloud"},
  ...
]
```


Expected Output (excerpt):


```
{"data": {"props": {"data": [
  {"stacks": [{"key": "peer", "value": 59}],
   "category": "sediment"},
  {"stacks": [{"key": "peer", "value": 98}],
   "category": "nonsense"},
  {"stacks": [{"key": "peer", "value": 81}],
   "category": "cloud"},
  ...
]}}}
```


A.5 Anomaly Detection Dataset

This appendix provides a representative example from the Anom dataset.

A.5.1 Network Path Latency Analysis

Task Instruction:

> Use the Network path data to identify IF there are any specific high latency nodes. Provide a summary to the user of impacted nodes.

Input Data (excerpt):


```
{"test": {"testId": "281208",
  "testName": "Synthetic Network Test",
  "type": "network"},
 "pathVis": [{
   "agent": {"agentName": "agent-89",
             "countryId": "JP"},
   "server": "service26.example.com:443",
   "pathTraces": [{
     "hops": [
       {"hop": 1, "ipAddress": "180.137.137.113",
        "hostname": "srv-34.fernandez.com",
        "responseTime": 2},
       {"hop": 2, "ipAddress": "132.239.231.168",
        "hostname": "lt-24.davies.com",
        "responseTime": 2},
       ...
     ]}]}]}
```


Expected Output:


```
{"Reasoning": [
  "Each path trace contains hops with response times.",
  "Any node with responseTime > 10ms is high latency.",
  "Iterate through each path trace and check hops.",
  "No node has responseTime exceeding 10ms threshold.",
  "Based on the data, there are no high latency nodes."
 ],
 "variables": {
  "Summary of high latency nodes":
    "No nodes with high latency were detected."
 }}
```

A.6 Summarization Dataset

This appendix provides a representative example from the Sum dataset.

A.6.1 Board Report Generation

Task Instruction:

> Given a JSON object that contains the current context of the 'board', generate a holistic report that derives key data points, insights, timelines, incident details, resolutions, or root cause analysis. Organize your response with clearly defined sections and a table of contents.

Input Data (excerpt):

```
{"board": {
  "name": "AIC 10 - Sep 27, 2025 5:42 PM",
  "metadata": {},
  "canvas": {"id": "canvas-12345", ...},
  "cards": [...],
  "conversations": [...]
}}
```

Expected Output (excerpt):

```
# Report on Board "AIC 10 - Sep 27, 2025"

## Table of Contents
1. [Introduction](#introduction)
2. [Board Overview](#board-overview)
3. [Canvas Details](#canvas-details)
4. [Cards Analysis](#cards-analysis)
5. [Conversations Historical Context]
6. [Conclusion](#conclusion)

## Introduction
This report presents an analysis of the board
titled "AIC 10 - Sep 27, 2025 5:42 PM,"
providing insights into its components...

## Board Overview
- **Name**: AIC 10 - Sep 27, 2025 5:42 PM
- **Description**: The board is identified by
  its timestamp, suggesting it may be part of
  a series or larger project...
```

A.7 Canvas Dataset

This appendix provides a representative example from the Canvas slot-filling dataset.

A.7.1 Runbook Step: Network Identification

Task Context:

> You are an expert in networking with an expert-level networking certification. You are helping with running a network troubleshooting run-book. Steps involve data gathering, analyzing the data and setting variables, running commands against the network, and making decisions based on the data.

Input Data (excerpt):

```
{"context": {
  "board": {
    "board_id": {"id": "cf53d6d9-ef2a-..."},
    "name": {"name": "[Redacted] - Aug 15, 2025"},
    "metadata": {...},
    "canvas": {...},
    "cards": [...],
    "conversations": [...]
  }
},
"runbook_step": "Get the list of all networks
```

```
  in the org. Find the network that best matches
  'Seattle' location and return the networkId
  and network name."
}
```

Expected Output:

```
{"core_skill_input": {
  "orgId": "928177",
  "query": "Get the list of all the networks
    in the org. Find the network that best
    matches 'Seattle' location and return
    the networkId and network name.",
  "baseUrl": "[Redacted]",
  "networkId": "L_4729533",
  "organizationId": "928177"
}}
```

A.8 RB-Text Dataset

This appendix provides a representative example from the RB-Text API response summarization dataset.

A.8.1 API Response Summary for [Redacted]

Task Context:

> You are an expert in networking with an expert-level networking certification. You are helping with running a network troubleshooting run-book. Please summarize the following response that was obtained calling an API and output it in a markdown format.

API Response (excerpt):

```
{"status": "success",
 "message": "The analysis cannot be completed
   due to invalid start or end timestamps.
   This typically occurs when the specified
   time range is not available in the test
   data or the timestamps are outside the
   valid data retention period..."}
```

Expected Output:

```
**[Redacted]**

- Analysis could not be completed due to
  invalid start or end timestamps.
- The specified time range is either
  unavailable in test data or outside
  the data retention period.
- To resolve:
  - Use the "Explain Selection" button in
    the [Redacted] for valid timestamps.
  - Ensure test ID 7802744 has data for
    the selected period.
  - Verify timestamps are within your
    account's data retention period.
```

A.9 RB-JSON Dataset

This appendix provides a representative example from the RB-JSON conditional expression evaluation dataset.

A.9.1 Variable Existence Check

Task Context:

> You are an expert in networking with an expert-level networking certification. We are executing a flow chart that corresponds to a run-book used for network

troubleshooting. We need to figure out the truth value of the expression in the reasoning instruction to execute the flow chart.

Reasoning Instruction:

```
If "Location" is set, go to step 3.
```

Context Variables:

```
{"Application": "video server",
 "Location": "[Redacted]",
 "StartTime": "2025-11-10T00:00:00Z"}
```

Expected Output:

```
{"Reasoning": [
   "The variable 'Location' is present
    in the provided data.",
   "The value of 'Location' is '[Redacted]',
    which means it is set (not null or empty).",
   "Therefore, the condition 'Location is set'
    is true."
 ],
 "expression": "Location is set",
 "truth_value_of_expression": true}
```

A.10 TOON-QA Dataset

This appendix provides representative examples from the TOON-QA dataset [6]. Questions span field retrieval, aggregation, and multi-condition filtering over structured JSON payloads.

A.10.1 Order Filtering and Counting

Input Data (excerpt):

```
{"orders": [
  {"orderId": "ORD-0001",
   "customer": {"id": 1, "name": "Valerie Braun",
     "email": "name.jones@gmail.com"},
   "items": [
     {"sku": "SKU-OOH73G", "name": "Widget A",
      "quantity": 2, "price": 29.99},
     {"sku": "SKU-PLM12X", "name": "Gadget B",
      "quantity": 1, "price": 49.99},
     ...
   ],
   "status": "processing",
   "total": 109.97},
  ...
]}
```

Question:

> How many orders have status "processing" and at least 3 items? Provide only the direct answer, without any additional explanation or formatting.

Expected Answer: `5`

A.10.2 Employee Aggregation

Question:

> How many active employees have more than 5 years of experience? Provide only the direct answer, without any additional explanation or formatting.

Expected Answer: `63`

A.10.3 Time-Series Lookup

Question:

What was the revenue on 2025-01-04? Provide only the direct answer, without any additional explanation or formatting.

Expected Answer: `8357.79`

A.11 Hard Reasoning Dataset

This appendix provides a representative example from the Hard multi-hop reasoning dataset.

A.11.1 Multi-Hop Test Discovery

Query:

Find a **[Redacted]** DNS test (type: dns-server) that is related to the Application (**[Redacted]**) and is run from an agent in the Location (**[Redacted]**).

Input Data (excerpt):

```
{"tests": [
  {"testId": 264343,
   "testName": "A - [Redacted] - DNS - Internal",
   "type": "dns-server",
   "target": "[Redacted]",
   "agents": [
     {"agentName": "pdx-linux-ea", "location": "Portland"},
     {"agentName": "[Redacted]", "location": "San Jose"},
     {"agentName": "newark-agent", "location": "Newark, NJ"}
   ]},
  {"testId": 264347,
   "testName": "A - [Redacted] - DNS - External",
   "type": "dns-server",
   "target": "[Redacted]",
   "agents": [
     {"agentName": "phoenix-agent", "location": "Phoenix, AZ"},
     {"agentName": "[Redacted]", "location": "[Redacted]"}
   ]},
  ...
]}
```

Expected Output (excerpt):

```
{"Reasoning": [
  "The instruction is to find a [Redacted] DNS
   test (type: dns-server) related to [Redacted]
   running from an agent in [Redacted].",
  "I found tests with '[Redacted]' in the name:
   'A - [Redacted] - DNS - Internal' (264343),
   'A - [Redacted] - DNS - External' (264347).",
  "'A - [Redacted] - DNS - Internal' has agents in
   Portland, San Jose, Newark - none in SF.",
  "'A - [Redacted] - DNS - External' has agents in
   Phoenix and [Redacted]. This matches!",
  "The matching test is testId 264347."
 ],
 "Answer": {
   "testId": "264347",
   "testName": "A - [Redacted] - DNS - External"
 }}
```

B Evaluation Prompts

This appendix provides the complete prompts used by our LLM-as-a-judge evaluators.

B.1 GenericJudge Prompt

The following prompt is used by the GenericJudge evaluator (GPT-4o) for open-ended quality assessment:

```
You are an evaluator tasked with comparing a Ground Truth Answer,
which represents the correct solution, to a Generated Answer,
which is the system's response to a given input question.
```

```
Follow these steps to conduct your evaluation:

1. **Read Carefully**
   Review the Question, Ground Truth Answer, and Generated Answer
   thoroughly.

2. **Assign a Score (1-5)**
   Evaluate the Generated Answer against the Ground Truth Answer
   using the following rubric:

   * **5 - Excellent**: Fully correct, complete, and clearly
     articulated. Matches the Ground Truth in factual content
     and intent with no significant errors or omissions.
   * **4 - Good**: Mostly correct and covers most key points.
     Minor inaccuracies or omissions that don't significantly
     affect overall quality.
   * **3 - Fair**: Partially correct. Captures some key elements
     but misses others. May include inaccuracies or lack clarity.
   * **2 - Poor**: Largely incorrect or incomplete. Contains
     major errors, omissions, or misinterpretations.
   * **1 - Inadequate**: Completely incorrect, irrelevant, or
     nonsensical. Shows no meaningful understanding of the query.

3. **Output Final JSON**
   Return a valid JSON object with exactly two keys:
   * "score": an integer from 1 to 5
   * "justification": a brief explanation for the score

Here are the inputs for you to conduct your evaluation:

Question:
[BEGIN QUESTION]
{question}
[END QUESTION]

Ground Truth Answer:
[BEGIN GROUND TRUTH ANSWER]
{ground_truth}
[END GROUND TRUTH ANSWER]

Generated Answer:
[BEGIN GENERATED ANSWER]
{response}
[END GENERATED ANSWER]
```

B.2 ReasoningJudge Prompt

The following prompt is used by the ReasoningJudge evaluator (GPT-4.1) for structured JSON output assessment:

```
You are an evaluator tasked with assessing a System Under Test (SUT)
JSON output against a Ground Truth JSON and a provided JSON Schema.
Base your judgment strictly on the Ground Truth and Schema--do not
use outside knowledge.

Follow these steps:

1. **Parse and Validate JSON**
   * Extract JSON from the SUT Output (strip code fences if present).
   * If parsing fails, assign a score of 1.
   * If parsing succeeds, validate against the Schema.

2. **Compare to Ground Truth**
   * Ignore extra fields not in the Ground Truth.
   * For each Ground Truth field, check:
     * Field exists in the SUT.
     * Type matches.
     * Value matches, using these rules:
       - Strings: exact match after trimming whitespace.
       - Numbers/booleans: exact equality.
       - Arrays (structured data): same length, element-wise
         equality, same order.
       - Arrays (reasoning): allow grouping/splitting with
         semantic matching; penalize missing or contradictory
         points.
       - Objects: compare recursively.
   * For free-text fields: paraphrasing is acceptable only if
     all factual points are preserved and no contradictions
     are introduced.
```

```
3. **Assign a Score (1-5)**
   * **5 - Excellent**: Valid JSON; schema-valid; all fields
     match (or only negligible paraphrasing).
   * **4 - Good**: Valid JSON; schema-valid; most fields correct
     with minor omissions; no contradictions.
   * **3 - Fair**: Valid JSON; schema-valid; some fields correct,
     but notable errors or omissions.
   * **2 - Poor**: Valid JSON but major mismatches, schema invalid,
     or reasoning contradicts Ground Truth.
   * **1 - Inadequate**: Not valid JSON, or completely inconsistent.

   Note: Schema invalidation caps the score at 2 maximum.

4. **Output Final JSON**
   * "score": integer 1-5.
   * "justification": brief explanation citing specific issues.

Here are the inputs for you to evaluate:

Ground Truth JSON:
[BEGIN GROUND TRUTH]
{ground_truth}
[END GROUND TRUTH]

SUT Output (to be parsed and validated):
[BEGIN SUT OUTPUT]
{sut_output}
[END SUT OUTPUT]

JSON Schema to validate the SUT Output against:
[BEGIN SCHEMA]
{json_schema}
[END SCHEMA]
```

C Additional Implementation Details

C.1 SQL Prompt Auto-gen

HYVE generates SQL tool prompts directly from the active session datastore rather than from hand-written templates. This design keeps the prompt aligned with the *actual* tables created during preprocessing and avoids inconsistent schema descriptions. At a high level, prompt generation has two goals: to expose the live relational structure that is available for querying, and to provide task-appropriate guidance for the next tool-enabled LLM step.

For the primary LLM call, HYVE appends a unified prompt that exposes the two high-level entry points available at that stage: `GenTemplateAndBackfill`, which enters Mode 2 template backfill, and `QueryDatastore`, which executes Mode 3 evidence-query reasoning. The prompt includes a schema description derived from the live datastore together with guidance and examples grounded in the tables, columns, and relations available for the request. However, the two branches are not disclosed symmetrically. For `QueryDatastore`, the primary-call prompt already provides the full tool specification and the SQL-oriented guidance needed for the model to issue evidence queries immediately. By contrast, `GenTemplateAndBackfill` is exposed in the primary call only as an entry-point decision, not with its full backfill instructions. This asymmetry follows the gradual-disclosure design described earlier: detailed prompt instructions are revealed only when they are needed.

If the primary call selects `GenTemplateAndBackfill`, HYVE generates a second prompt specialized to Mode 2. This follow-up prompt introduces the `BackfillData` tool contract and provides the detailed instructions and auto-generated examples needed to map queried values back into the model-generated template in a schema-consistent way. If the primary call instead selects `QueryDatastore`, HYVE enters Mode 3. In this branch, no new SQL prompt is generated, because the full `QueryDatastore` specification and SQL guidance were already provided in the primary-call prompt. Instead, HYVE executes the SQL tool call against the same session datastore, appends the returned evidence to the prior conversation context, and issues the bounded follow-up LLM call for final synthesis described in Section C.3.

Because every prompt is synthesized from the same live schema, both the tool instructions and the few-shot examples are grounded in the actual tables, columns, and relations available for the

Algorithm 1 Request-Specific SQL Prompt Generation

Input: Datastore $\mathcal{D}$, base message m, prompt stage $\sigma \in \{\text{primary-call}, \text{mode2}, \text{mode3}\}$
Output: Tool prompt or follow-up context π
Extract the live schema and relational structure from $\mathcal{D}$
Build a request-specific schema description for the current tables and relations
if $\sigma =$ primary-call **then**
    Expose `GenTemplateAndBackfill` as the Mode 2 entry-point choice
    Specify the full `QueryDatastore` contract for Mode 3 evidence-query reasoning
    Add guidance and examples for evidence-oriented SQL generation
else if $\sigma =$ mode2 **then**
    Specify the `BackfillData` tool contract
    Add detailed guidance and examples for schema-consistent template backfilling
else
    Execute the `QueryDatastore` SQL tool call
    Append the returned evidence to the prior message context
    Prepare the bounded follow-up LLM call for final synthesis
end if
Assemble the resulting prompt or follow-up context π
return π

current request, rather than in generic placeholders. The prompt is therefore both *schema-grounded* and *branch-aware*: the primary-call prompt fully supports Mode 3 evidence-query reasoning while exposing only the Mode 2 entry point, the Mode 2 follow-up prompt is disclosed only after template backfill has been selected, and the Mode 3 follow-up stage reuses the SQL guidance already provided in the primary call by appending queried evidence for final synthesis. Together, these features help HYVE guide the LLM toward richer and more accurate SQL tool calls.

C.2 Mode 2: `BackfillData` Execution

The postprocessor inspects the primary LLM output and, when the request enters Mode 2, executes the deterministic `BackfillData` path. In this mode, the LLM generates both a partially visible answer template and the SQL-based backfill specification, including the query and its column-to-template mappings. The postprocessor then executes that specification against the active session datastore and restores the hidden values while preserving the template structure produced by the model. The task-specific prompt is synthesized from the *actual* datastore schema and teaches the model how SQL results should be mapped back into the partially visible template.

If the datastore contains a multi-series table with a `series_idx` column and attached series labels, the prompt may include an example such as:

```
{
  "tool_name": "BackfillData",
  "query": "SELECT x, y, series_idx
            FROM data ORDER BY series_idx, _row_id",
  "mappings": [
    {
      "sql_column": "x",
      "template_path":
          "path.to.list.(series_idx).data.(N).x"
    },
    {
      "sql_column": "y",
      "template_path":
          "path.to.list.(series_idx).data.(N).y"
    }
  ]
}
```

This means: execute the SQL query on the full datastore, group the returned rows by `series_idx`, navigate to the corresponding series object in the template, and rebuild its `data` array from the SQL values. Here `(series_idx)` is not a literal field name in the output; it is a placeholder that the

Algorithm 2 Multi-Series `BackfillData` Execution

1: **Input:** Template JSON τ, SQL query q, mappings M
2: **Output:** Backfilled output τ
3: $R \leftarrow \text{EXECUTESQL}(q)$
4: $g \leftarrow \text{DETECTGROUPCOLUMN}(M, \text{COLUMNS}(R))$
5: **if** $g = \text{UNSUPPORTED}$ **then**
6: **return** τ
7: **else if** $g \neq \varnothing$ **then**
8: **for** each distinct group value u in $R[g]$ **do**
9: $R_u \leftarrow \text{ROWSWHERE}(R, g = u)$
10: **for** each mapping $(c, p) \in M$ **do**
11: $p_u \leftarrow \text{REPLACE}(p, (g), (u))$
12: $p_u \leftarrow \text{VALIDATEANDCORRECTPATH}(\tau, p_u)$
13: $V \leftarrow R_u[c]$ in SQL order
14: **for** each expanded path $\hat{p}$ in $\text{EXPANDNESTEDN}(\tau, p_u, |V|)$ **do**
15: $\text{BACKFILLPATHWITHSCHEMA}(\tau, \hat{p}, V)$
16: **end for**
17: **end for**
18: **end for**
19: **else**
20: **for** each mapping $(c, p) \in M$ **do**
21: $p \leftarrow \text{VALIDATEANDCORRECTPATH}(\tau, p)$
22: $V \leftarrow R[c]$ in SQL order
23: **for** each expanded path $\hat{p}$ in $\text{EXPANDNESTEDN}(\tau, p, |V|)$ **do**
24: $\text{BACKFILLPATHWITHSCHEMA}(\tau, \hat{p}, V)$
25: **end for**
26: **end for**
27: **end if**
28: **return** τ

executor replaces with concrete group indices such as `(0)`, `(1)`, and `(2)`. The token `(N)` denotes the list dimension that should be cleared and rebuilt in the order of the SQL result. This positional convention is necessary because, once the SQL query filters or reorders the original rows, the original `_row_id` values no longer determine the correct positions in the partially visible template. Instead, backfilling must rely on the ordered result sequence itself to reconstruct the list faithfully. Because the rebuilt list items are cloned from the visible template items, the final result preserves the LLM-generated nested schema while replacing only the truncated arrays with full-fidelity datastore values.

Line 4 detects whether the SQL result contains a grouping field, such as `series_idx`, that distinguishes multiple repeated substructures. The algorithm then follows three cases: grouped backfill, flat backfill, or a fallback when grouping is ambiguous or unsupported.

C.3 Mode 3: Bounded LLM Invocations

HYVE provides an explicit architectural bound on model invocations. After the primary LLM call selects a postprocessing branch, Mode 3 performs, by default, at most one additional LLM call: the system executes the `QueryDatastore` tool call, appends the returned SQL evidence to a slim context, and asks the model for a final grounded synthesis.

Consequently, HYVE uses at most two LLM calls per request by default: one primary call and, only when necessary, one follow-up call. This keeps latency predictable and is easier to operationalize than open-ended agentic loops, which may issue an unbounded number of tool calls.

As an optional optimization, HYVE also allows a small configurable number of additional repair calls for SQL generation. For example, when a generated query fails due to an execution error or incomplete evidence, the system can supply the error message or missing-context hints and request a revised SQL query. This mechanism remains bounded and is used only to improve query quality under controlled retry limits.

C.4 Illustrative Prompt/Datastore Example

We illustrate this process with a real example. The following snippet shows $\mathsf{Rep}(s)$ after preprocessing the input s. For clarity, we retain only the first three data entries in the column view and omit non-essential descriptions. The transformed prompt π exposes only a small visible slice of the original chart while injecting relational metadata for SQL generation. In this example, the user asks for a multi-series line chart over path-performance telemetry, and the preprocessor retains truncated column views, presents a short row-view sample, and then appends the tool-selection prompt and datastore schema:

```
<language descriptions/instructions omitted...>
  "(CARD 1)": {
    "name": "CDSAILineChart",
    "props": {
      "data": {
        "id": [
          "Newark, NJ", "pdx-linux-ea", "New York, NY"
        ],
        "data": {
          "x": [
            [1770163200000,1770163500000,1770163800000],
            [1770163200000,1770163500000,1770163800000],
            [1770163200000,1770163500000,1770163800000]
          ],
          "y": [
            [2.1, 1.8, 2.2],
            [17.7, 18.0, 16.0],
            [28.1, 26.3, 22.9]
          ]
        }
      },
      "axes": {
        "xAxis": {
          "type": "time"
        }
      }
    }
  }
<other instructions/intents/questions omitted...>
### Row View:
    data:
      x | y_0 | y_1 | y_2
      ------------------
      1770163200000 | 2.1 | 17.7 | 28.1
      1770163500000 | 1.8 | 18.0 | 26.3
      1770163800000 | 2.2 | 16.0 | 22.9
      1770164100000 | 2.1 | 15.3 | 23.1
      1770164400000 | 2.4 | 16.1 | 24.7
      1770164700000 | 2.0 | 15.9 | 24.0
      ... (762 more rows)

###Tool Selection (data is truncated, with the complete data in datastore):
<tool call specifications omitted...>
Available tables in datastore:
Table: data
  Source: CARD 1.props.data.data
  Total rows: 2304
  Rows per series: 768
  series_idx values: 0='Newark, NJ', 1='pdx-linux-ea', 2='New York, NY'
  Columns:
    - series_idx (int)
    - x (int)
    - y (float)
    - _row_id (int)
### Few-Shot Query Examples:
<auto-gen to guide LLM SQL generation; details omitted...>
```

This excerpt clarifies the role of π: it is not merely a compressed serialization of the raw input, but a hybrid prompt that combines visible evidence with truncation metadata. When data-entry truncation is triggered, additional SQL-generation prompts, table-schema information, and few-shot guidance are also provided. The LLM therefore reasons over the visible slice while using the injected table description to decide whether it can answer directly, should call `QueryDatastore`, or should initiate `GenTemplateAndBackfill`.

C.5 Illustrative Mode 2 Answer-Template and SQL Example

Below is an illustrative output for the preceding example, produced in response to `GenTemplateAndBackfill`. It contains an answer template derived from the visible data together with a `BackfillData` tool call. This tool call includes SQL statements and mappings that specify how the query results should be backfilled into the template. By default, this process uses one additional SQL query.

The answer template generated by the LLM

```
{
  "CDSAILineChart": {
    "data": {
      "list": [
        {
          "series_name": "Newark, NJ",
          "data": [
            {
              "x": 1770163200000,
              "y": 2.1
            }
            // ... will be filled in for Newark, NJ
          ]
        },
        {
          "series_name": "pdx-linux-ea",
          "data": [
            // ... will be filled in for pdx-linux-ea
          ]
        },
        {
          "series_name": "New York, NY",
          "data": [
            // ... will be filled in for New York, NY
          ]
        }
      ]
    },
    "axes": {
      "xAxis": {
        "type": "time"
      }
    }
  }
}
```

The SQL tool-call specification

```
{
  "tool_name": "BackfillData",
  "query": "SELECT x, y, series_idx FROM data ORDER BY series_idx, _row_id",
  "mappings": [
    {
      "sql_column": "x",
      "template_path": "CDSAILineChart.data.list.(series_idx).data.(_row_id).x"
    },
    {
      "sql_column": "y",
      "template_path": "CDSAILineChart.data.list.(series_idx).data.(_row_id).y"
    }
  ]
}
```

This output requests a `BackfillData` operation over the full datastore. The SQL query retrieves the complete `x` and `y` series together with `series_idx` and `_row_id`, ordered so that each series can be reconstructed in its original order. Each mapping specifies how one SQL column should be written back into the output template. Here, `(series_idx)` identifies the target series, while the final indexed position `(_row_id)` denotes the element within that series. The postprocessor can therefore reconstruct the full chart data while preserving the schema produced by the LLM.

C.6 Complementarity with File-Oriented Harnesses

Although HYVE is not itself an agent, it can support network troubleshooting agents that combine a virtual file system, which maps observability endpoints to files and transparently intercepts Bash tools, with database-style management of structured data. The file-system abstraction provides intuitive and stateful high-level organization, while the datastore provides precise control over fine-grained structured entries. This paper focuses exclusively on HYVE.

Claude Code [17] and Vercel's filesystem-agent approach [18] illustrate agentic workflows that use file access and shell tools. In machine-data-heavy settings, however, database principles become equally important. Shell tools such as `grep` and `awk` are useful for pattern matching, but they are poorly suited to structured search involving filtering, aggregation, and joins over multiple nested machine-data segments scattered across the input. HYVE therefore stores full-fidelity copies in a *datastore* that supports formal query processing, while presenting hybrid views better aligned with LLM reasoning.

D System Architecture and Implementation

This section presents the system architecture and the main implementation details.

D.1 Formal Definitions for Hybrid Views

The following definitions make precise the schema-consistency and hybrid-view notions used in the main paper.

Definition 1 (Schema Consistency)**.** *Let $L = [e_1, \ldots, e_n]$ be a list of structured JSON values. We say that L is* schema-consistent *if all elements of L share the same recursive schema signature, i.e., they have the same field structure and compatible leaf types at corresponding positions. In particular, for object-valued elements, this requires that they have the same key set and that corresponding child fields are themselves schema-consistent recursively. Otherwise, L is* schema-inconsistent*.*

Definition 2 (Column-Oriented Representation)**.** *Given a JSON object J, let $\mathcal{S}(J)$ be the set of scalar-value occurrences in J. Each occurrence is represented as a tuple (c, p, v), where v is the scalar value, p is its flattened field path relative to its enclosing repeated record, and c is its repeated-context path, defined as the ordered sequence of array-valued ancestor paths from the root of J to that enclosing record.*

The column-oriented representation of J is

$$\mathsf{Col}(J) = \{((c_i, p_i), V_i)\}_{i=1}^{m},$$

where each (c_i, p_i) is a column identifier, and $V_i = [v_1, \ldots, v_{n_i}]$ is the ordered list of all scalar values whose occurrences in $\mathcal{S}(J)$ have the same repeated-context path c_i and the same relative flattened field path p_i. The ordering of V_i follows the original traversal order of the corresponding occurrences in J. Thus, two values are placed in the same column iff they are from schema-consistent repeated records, share the same repeated-context path, and have the same relative flattened field path.

A simple example clarifies this. Consider the path `clusters[].nodes[].metrics[].cpu` and a corresponding one `clusters[].nodes[].metrics[].mem`. They both have the same repeated-context path `clusters[].nodes[].metrics[]`, but with different relative flattened field paths, i.e., `cpu` and `mem`, respectively. Therefore, their column identifiers can be represented as tuples (`clusters[].nodes[].metrics[]`, `cpu`) and (`clusters[].nodes[].metrics[]`, `mem`). By contrast, for a path `clusters[].nodes[].status`, the enclosing repeated record is an element of `nodes[]`, so its column identifier is (`clusters[].nodes[]`, `status`).

Definition 3 (Row-Oriented Representation)**.** *Given a collection of JSON objects $\mathbf{J} = (J_1, \ldots, J_m)$, its row-oriented representation is a set of records*

$$\mathsf{Row}(\mathbf{J}) = \{r_1, \ldots, r_n\},$$

where each r_i is a complete schema-consistent repeated record extracted from some J_k. Each row corresponds to one original occurrence of that record in the input and retains all of its fields together as a single tuple, preserving the co-occurrence relationships among those fields. The row view may

Algorithm 3 Pre-processing and Hybrid-View Construction

Input: Raw input string s, query text q for ranking
Output: Prompt-ready hybrid view and full datastore
$(T, O) \leftarrow \text{EXTRACTTEXTANDJSONOBJECTS}(s)$
$\mathcal{D} \leftarrow \text{NEWDATASTORE}()$
$\mathcal{C} \leftarrow [\,]$
for each object $o \in O$ **do**
  $J_o \leftarrow \text{PARSE}(o)$
  $\text{BUILDDATASTOREFROMOBJECT}(\mathcal{D}, J_o)$
  $\tau \leftarrow \text{BUILDJSONTREE}(J_o)$
  $\text{NORMALIZEEXPANDEDUNPARSEDNODES}(\tau)$
  $\text{MARKSCHEMAINCONSISTENCY}(\tau)$
  $C \leftarrow \text{CLUSTERBYJSONPATHPATTERN}(\tau)$
  $E \leftarrow \text{MERGEANDRESHAPECOLUMNS}(C)$
  $R \leftarrow \text{RANKSCHEMACOMPATIBLEINDEXES}(E, q)$
  $E' \leftarrow \text{APPLYTRUNCATIONANDRANKING}(E, R)$
  $\mathcal{C}.\text{APPEND}(\text{RENDERCOLUMNVIEW}(E'))$
end for
$\mathsf{Row}(\mathbf{J}) \leftarrow \text{RENDERRANKEDROWS}(\mathcal{D}, q)$
$\pi \leftarrow \text{ASSEMBLEPROMPT}(T, \mathcal{C}, \mathsf{Row}(\mathbf{J}))$
return $(\pi, \mathcal{D})$

aggregate records from multiple objects and rank them by relevance to the request context; unlike the column view, it does not preserve the original global ordering of records.

For example, if `clusters[].nodes[]` is a schema-consistent repeated record, then one row could be (`id="node-1"`, `status="healthy"`, `zone="us-west"`) and another row could be (`id="node-2"`, `status="degraded"`, `zone="us-east"`). Each row keeps the fields of a single original node record together, so correlations such as which `status` belongs to which `id` remain explicit.

D.2 System design

At a high level, HYVE consists of three coordinated components (Figure 1) organized around a session-scoped queryable datastore. To support an in-memory datastore and expressive SQL features (e.g., extraction operators for nested JSON objects), HYVE uses DuckDB [19]. Concretely, it instantiates an unnamed in-memory DuckDB database for the session via the special `:memory:` target. This yields an efficient transient datastore. No data are written to disk; the connection and its materialized tables remain available across related calls within the session.

HYVE can be integrated with standard LLM APIs in a provider-agnostic manner, for example through OpenAI-style interfaces such as the Responses API [2] or Chat Completions API [3], as well as Anthropic's Messages API [4]. HYVE remains responsible for structured parsing, datastore construction, truncation-aware representation, and query-based output repair, while the underlying LLM API handles request execution and tool-calling transport.

D.3 Preprocessor Implementation

Given a raw input string, HYVE extracts structured text segments and parses each into a canonical object. It materializes these objects in the active session datastore before any truncation. HYVE then builds the column view by constructing a JSON tree, flattening it into JSONPath-like dotted keys, and clustering paths that share the same repetitive structural pattern after abstracting away index values. These clusters are merged into schema-aware columns whose values are reshaped into ordered lists. When lists are large, HYVE applies truncation and ranking so that high-signal table entries can still appear in the visible prompt. HYVE similarly derives the row view by ranking records against the request context, selecting a small representative subset, and rendering the result as a compact pipe-delimited table. Algorithm 3 shows how a datastore is first constructed for a session; subsequent related calls reuse that datastore.

JSON Tree and Schema Consistency The JSON tree is the intermediate representation that makes this transformation precise. Each node stores a key, a value, whether it originated from a dictionary or a list, and links to its children. Dictionary edges preserve semantic field names, whereas list edges are represented by numeric child keys. During tree construction, HYVE also attempts to expand long string values that themselves contain JSON or Python-literal payloads, so that nested structure is exposed rather than treated as opaque text. Flattening the tree then yields dotted paths such as `data.0.x`, `data.1.x`, and `data.2.x`. Clustering collapses these paths to a shared pattern such as `data.*.x`, while retaining capture cardinalities so that the collected values can be reshaped into aligned lists.

Schema consistency is required because clustering is valid only when different list items instantiate the same record schema. HYVE therefore performs a bottom-up check over every list node. For each list item, it computes a *deep schema signature* that recursively records child keys and normalized leaf types, treating `int` and `float` uniformly as `number`. A list is declared schema-consistent only if all items share the same signature and no descendant leaf exceeds a configured maximum length for column-view rendering. This check prevents false alignment. Without it, unrelated objects such as `{x,y}` and `{name,status,error}` could be merged into the same column family simply because they occupy the same list position, producing malformed columns and invalid row correspondences.

When this check fails, HYVE marks the list node as schema-inconsistent and preserves the numeric indices of all list items explicitly. This disables clustering across those items and ensures that only entries with the same inferred schema are merged into a single logical table or column family. The same safeguard applies to enumerated dictionary children when they contain sufficiently long payloads, as determined by a configuration parameter. HYVE preserves their indices rather than forcing them into a shared columnar pattern. As a result, schema-consistent regions are compressed into tables, whereas schema-inconsistent regions remain as indexed records.

Relationalization of Nested Lists Datastore construction follows a complementary invariant. HYVE recursively scans the parsed object and promotes sufficiently large lists of dictionaries to parent tables. Inside each parent row, list-valued fields are split into child tables using foreign keys of the form `parent_<id>`; if those child rows still contain nested lists, the same rule is applied recursively to produce grandchild tables. Flat one-to-one dictionaries are expanded into columns of the current table, while residual nested objects are serialized as JSON strings when they cannot be normalized safely. The implementation also handles two important multi-series cases. First, for column-store dictionaries whose values are parallel lists of lists, HYVE converts them into a single flat table with a `series_idx` column and attaches sibling label arrays as series metadata. Second, when a list of sibling objects contains parallel sublists under the same key, HYVE combines those sublists into one shared table, again indexed by `series_idx`. These rules ensure that lists at different depths are either mapped to legitimate relational tables with explicit join keys or preserved as indexed nested structures when no sound relational alignment exists.

Robust Parsing Given the raw input string s, the preprocessor first identifies candidate structured segments as part of the Parse operator (Section 2). Each candidate segment is then passed to a multi-strategy robust parser that attempts to recover a canonical structured object. This parser is designed to handle malformed JSON, string-encoded nested objects, and Python AST literals.

The parser follows a two-stage design. The *fast parsing stage* targets common cases with low overhead. It first strips surrounding Markdown code fences, then tries a small sequence of lightweight parsers that move from strict to more permissive formats: canonical JSON parsing, followed by Python-literal parsing with only minimal syntax normalization needed to expose the literal itself. This stage does not attempt broad repair. Whenever one of these parsers succeeds, HYVE recursively traverses the resulting dict/list structure. If a string-valued field itself parses as a JSON object or array, HYVE replaces that string with the parsed structure and continues the traversal on the newly exposed subtree until no further expansion applies. This recursive expansion is necessary because real inputs often contain nested payloads that are string-encoded inside outer JSON fields; without it, deeper records would remain opaque text and could not be normalized into the hybrid view or datastore. This stage covers many practical inputs, including code-fenced JSON, Python dict/list literals, and stringified nested payloads.

If the fast stage fails, HYVE enters a *repair stage*. Before invoking more permissive recovery, the parser applies targeted normalization to make the candidate structurally coherent: it normalizes

Algorithm 4 Robust Parsing of a Candidate Segment

Input: Candidate structured segment s
Output: Parsed structured object O or the original segment s
$s_0 \leftarrow$ StripMarkdownFences(s)
Stage 1: Fast parsing
Try, in order: ParseCanonicalJSON, ParsePythonLiteral
for each parser P in the above order **do**
  $O \leftarrow P(s_0)$
  if $O \neq failure$ **then**
    $O \leftarrow$ ExpandNestedJSON(O)
    if ValidateSourceCoverage(O, s_0) **then**
      return O
    end if
  end if
end for
$s_1 \leftarrow$ NormalizeControlCharacters(s_0)
$s_1 \leftarrow$ NormalizeKeyQuoting(s_1)
$s_1 \leftarrow$ StabilizeNestedJSONStringSyntax(s_1)
Stage 2: Repair-oriented recovery
Try, in order: RepairMalformedJSON, DecodeEncodedJSON, RecoverEmbeddedJSONSpan
for each repairer R in the above order **do**
  $O \leftarrow R(s_1)$
  if $O \neq failure$ **then**
    ReconcileWithOriginal(O, s_0)
    PreserveUnrecoveredContent(O, s_0)
    $O \leftarrow$ ExpandNestedJSON(O)
    if ValidateSourceCoverage(O, s_0) **then**
      return O
    end if
  end if
end for
return s

control characters that would invalidate JSON strings, inserts quotes around unquoted field names, and escapes quotes inside nested JSON fragments embedded within string values. HYVE then proceeds through a small sequence of increasingly permissive recovery strategies. It first attempts structural repair of malformed JSON. If that still fails, it handles wrapper cases in which the payload is itself encoded as a quoted or escaped JSON string. Finally, when the candidate contains surrounding free-form text, HYVE extracts the most plausible embedded JSON span and, when necessary, restores missing closing or opening braces/brackets so that the recovered span forms a complete object or array. Because permissive recovery may silently drop or rewrite malformed content, each recovered candidate is reconciled against the original segment: content that is confidently recovered is kept in structured form, while any unrecovered residue is retained under a dedicated `unparsed_string` field rather than discarded. Finally, each candidate parse is validated against the original input by measuring overlap in alphabetic-word substrings. HYVE rejects parses that violate the word-overlap coverage criterion, thereby reducing silent truncation. Algorithm 4 summarizes this control flow.

E Operator Design

HYVE includes both generic and specialized operators, including ranking, rendering, and time-series analysis.

E.1 Ranking-Based Subset Selection

HYVE uses ranking in two settings. The row view supports *retrieval-oriented* tasks, where the goal is to select the most relevant records from a large collection. The column view preserves a prefix

and suffix together with a small number of additional elements selected by a ranking function, while maintaining the original order.

HYVE uses two related ranking procedures with different candidate corpora. For the column view, ranking is performed independently within each *schema-compatible* group. Each list position defines one candidate record, and ranking statistics are computed using only the records in that group. Because all records in the same group share the same field paths, schema tokens contribute no discriminative signal. Accordingly, column-view ranking uses only value tokens.

For the row view, candidate rows are drawn from the datastore. Here, each candidate is represented by both schema tokens and value tokens. Value tokens provide the fine-grained matching signal, while schema tokens identify the table and field family from which the values originate. This distinction matters when the same request contains multiple heterogeneous tables or repeated structures whose values may overlap lexically.

We score these candidates with a standard BM25-style ranker [20]. The query is a reference string formed from the surrounding input context together with $\mathbf{Slim}(\mathbf{J})$. The key point is not the scoring function itself, but how HYVE maps structured data into retrieval units. BM25 is applied not to free-form documents, but to structured records. For column-view ranking, IDF is computed within each schema-compatible group; for row-view ranking, it is computed over the cross-table candidate set. Token overlap with the reference context determines relevance, and repeated occurrences in the reference are saturated in the usual BM25 manner.

Relative to standard document retrieval, HYVE differs in two respects. First, its retrieval units are structured records augmented with schema information rather than free-form documents. Second, ranking is performed against a reference string derived from the request context. HYVE therefore applies BM25 in a structured setting, matching request-derived context to candidate records rather than ranking free-text documents.

E.2 Time-Series Analysis Operators

For structured inputs containing numerical time series or metric sequences, we provide optional statistical and anomaly-detection operators that convert raw numerical arrays into compact, high-signal natural-language summaries.

Long metric sequences, such as per-node network delays collected along a path, can consume substantial tokens while hiding the few data points that matter most. Blind truncation may discard the rare high-latency events that determine the diagnosis. HYVE therefore supports time-series analysis operators that compress such sequences into compact, semantically meaningful evidence, such as anomaly reports and trend descriptions.

As a simple illustration, we implement an n-sigma anomaly detector. Given a numerical sequence $X = [x_1, \ldots, x_n]$, it flags observations whose deviation from the mean exceeds a fixed multiple of the standard deviation (with default threshold $k = 2$), while applying a lightweight periodicity check to suppress false positives [21]. We expose this analysis as a query operator over the datastore. For instance, an LLM can invoke a custom SQL-style function, `DETECT_ANOMALY(data, y)`, to identify abnormal behavior in metric column `y` of table `data`. Rather than returning the full raw time series, the operator produces compact textual evidence.

For example, it may return a concise summary such as:

```
Anomaly detected in data.y:
  latency: 293.0 ms
Normal observation:
  mean: 3.08 ms, max: 11.19 ms
Context:
  timestamp: 2025-07-15T10:30:00
  pathTrace: 4, round: 9, hop: 4
  node: "pdx-linux-ea"
```

This summary surfaces the anomalous value together with relevant contextual fields and a brief statistical characterization of normal behavior. By transforming raw numerical sequences into compact, semantically meaningful summaries, such operators help the LLM focus on the most salient evidence without being overwhelmed by token-intensive raw data.

E.3 Rendering Operators

After flattening and truncation, we reconstruct a serialized representation suitable for the LLM prompt. This rendering step converts the internal column-oriented representation into a human-readable, LLM-consumable format.

We explored multiple output formats:

- **Beautified JSON (default):** Reconstruct the nested JSON object with consistent indentation for readability.
- **Raw JSON:** Preserve the JSON in minified form, eliminating unnecessary whitespace to maximize token density at the expense of readability.
- **TOON encoding:** Use a compact, lossless representation of the JSON data model for LLM input [6], combining YAML-like indentation with CSV-style tabular layouts for uniform arrays of objects.

TOON encoding offers strong token efficiency for table-like structured data. Our ablation study (Section 3.3) shows that it yields up to 31% token savings, with the largest gains on chart generation and slot-filling tasks (18–31%), though savings are smaller (6–15%) or negligible on other workloads. However, TOON can degrade structured-generation quality. We therefore use beautified JSON as the default rendering format, prioritizing broad task compatibility over maximum token savings. TOON remains a viable option when stronger token efficiency is desired.

As work orthogonal to the main theme of this paper, we also introduce operators for other input formats. For example, an HTML operator can transform HTML-formatted text into a browser-like rendered view, reducing formatting metadata that might otherwise obscure the key information for the LLM.

E.4 Datasets

The datasets include expert-authored and workflow-oriented benchmarks spanning question answering, slot filling, anomaly detection, chart generation, and network troubleshooting. We group our benchmarks into the following categories:

- **Cert-QA:** A collection of 3,524 network engineering questions spanning entry, advanced, and expert certification levels. This includes entry-level questions on basic commands and protocols, advanced questions on routing (OSPF, BGP) and switching (STP, MST), and expert-level questions covering unified communications, wireless security (802.1X, WPA2/3), and network analytics. The dataset combines both multiple-choice exam questions with detailed feedback and open-ended questions requiring precise technical answers. Representative examples are provided in Appendix A.1.
- **Runbook:** A collection of 260 network-operations troubleshooting scenarios derived from real-world support workflows. Each example contains a problem description, such as a stack upgrade failure or port connectivity issue, together with a ground-truth runbook specifying step-by-step diagnosis and verification procedures. The task requires the model to generate or complete a structured troubleshooting playbook from the problem context. A representative example is provided in Appendix A.2.
- **Line:** 100 line-chart generation tasks that require the model to transform time-series network telemetry, such as latency, jitter, packet loss, and goodput, into multi-series line-chart JSON specifications with appropriate axis configurations and data mappings. A representative example is provided in Appendix A.3.
- **Bar:** 100 bar chart generation tasks requiring the model to convert categorical key-value data into bar-chart JSON schemas with stacks and category labels. A representative example is provided in Appendix A.4.
- **Anom:** 797 anomaly detection tasks requiring the model to analyze network path visibility data and identify high-latency nodes. Each entry contains hop-by-hop response times from synthetic network tests, and the model must apply a latency threshold (e.g., $>$10ms) to flag anomalous nodes, then provide structured reasoning and a summary of impacted hosts. A representative example is provided in Appendix A.5.

- **Sum:** 84 report generation tasks requiring the model to synthesize structured board/canvas data into comprehensive Markdown reports. Each entry contains a JSON object with board metadata, canvas details, cards, and conversation history; the model must produce a well-organized report with a table of contents, key insights, incident timelines, and root cause analysis where applicable. A representative example is provided in Appendix A.6.
- **Canvas:** 4,096 slot-filling tasks aggregated from 8 runbook step scenarios. Each entry simulates an expert-level network troubleshooting workflow where the model receives a deeply nested board context (metadata, canvas, cards, conversation history) along with a runbook step instruction, and must extract or compute the required variables (e.g., `orgId`, `networkId`, time ranges, API query parameters) to proceed. A representative example is provided in Appendix A.7.
- **RB-Text:** A collection of API response summarization tasks derived from network troubleshooting runbook executions. Each entry contains a raw API response from commercial network management and monitoring platforms (e.g., device alerts, monitored targets, connectivity status) along with an expert-level system prompt; the model must produce a concise, well-structured Markdown summary that highlights critical information such as alert severity, affected devices, and actionable insights. A representative example is provided in Appendix A.8.
- **RB-JSON:** Representative conditional expression evaluation tasks requiring the model to assess boolean predicates over runbook execution context. Each entry presents a condition (e.g., "Location is set", "Fault Domain is local-network", "Any of the alerts are critical") along with structured context data from network troubleshooting workflows; the model must output a JSON object containing a step-by-step reasoning array and a boolean decision indicating whether the condition holds. A representative example is provided in Appendix A.9.
- **Hard:** A curated set of challenging multi-hop reasoning problems designed to stress-test structured data comprehension. Each entry presents a complex query (e.g., "find a DNS test related to **[Redacted]** running from an agent in **[Redacted]**") over deeply nested JSON payloads containing network test configurations, agent metadata, and location hierarchies. The model must perform multi-step filtering and cross-referencing to produce a detailed reasoning chain and the correct answer. A representative example is provided in Appendix A.11.
- **TOON-QA:** All 154 question-answer pairs from the official four-dataset release of the TOON Retrieval Accuracy Benchmark [6], rendered as beautified JSON. The four datasets contain 50 tabular, 40 nested, 34 analytics, and 30 GitHub-repository questions. This is the complete four-dataset release, not a selected subset of the 244-question v4.1.0 release. The questions span field retrieval, aggregation, and multi-condition filtering. Prompt lengths range from approximately 10K to 43K characters (mean 26K). Some questions test locating a single value; others require aggregation or filtering across records in a long structured context. A representative example is provided in Appendix A.10.

E.5 Evaluation Protocol and Reproducibility

The Raw and HYVE results for GPT-4.1, GPT-5, and Claude Sonnet 4.5 in Table 1 were produced with an automated LangSmith-based evaluation pipeline. Each dataset–model–method condition was evaluated in ten runs, and the table reports the mean across those runs. GPT-4.1 and GPT-5 were invoked through OpenAI-compatible chat-completion endpoints without explicitly setting `temperature`, `top_p`, or `max_tokens`; provider defaults were used.

Unless otherwise specified, the experiments used these preprocessing settings: list truncation threshold 30, retained entries 11, row-view top-k 8, SQL row-view top-k 5, ranking top-k 30, and beautified JSON as the default rendering format. The TOON variant in the ablation replaces this default rendering with TOON encoding while keeping the same evaluation protocol. Token usage is measured from the model API's reported prompt and completion token counts. Latency is measured as wall-clock request latency from request submission to response completion, including preprocessing, model inference, and postprocessing for HYVE.

Exact external reproduction of the proprietary benchmark rows requires access to the underlying internal datasets. To make the results auditable within that constraint, Appendix E.4 reports dataset

sizes and task definitions, Appendix E.7 specifies the metrics and judges, and Appendix B provides the evaluation prompts. TOON-QA is public and serves as an externally reproducible structured-data benchmark using the same evaluation protocol.

E.6 Compute Resources

HYVE is deployed as a production service on a managed Kubernetes cluster. The server is a Python 3.12 FastAPI application served by Uvicorn in hardened container images. The service runs across multiple CPU-only pods, each with multiple Uvicorn worker processes; no local GPUs are used. The GPT-4.1, GPT-5, and Claude Sonnet 4.5 comparisons use hosted model APIs, so accelerator resources for these evaluations are managed by the respective providers. Latency is measured as wall-clock time from request submission to response completion, encompassing HYVE preprocessing, remote model inference, and HYVE postprocessing. Token counts are taken from the model API's reported usage fields. Reproducing the proprietary benchmarks requires access to internal datasets and API credentials for the commercial model endpoints; the public TOON-QA benchmark can be tested independently using the same evaluation protocol described in Appendix E.5.

E.7 Metrics

We evaluate system performance along three dimensions: *answer quality*, *token efficiency*, and *latency*. All evaluations are conducted using an automated evaluation pipeline built on LangSmith [22].

Quality Metrics We employ task-appropriate quality metrics tailored to each dataset's characteristics:

- **GenericJudge (5-point scale):** An LLM-as-a-judge evaluator that compares the generated answer against a ground-truth reference. A GPT-4o judge assigns an integer score from 1 to 5 based on factual correctness, completeness, and clarity: *5 (Excellent)*—fully correct and complete; *4 (Good)*—mostly correct with minor omissions; *3 (Fair)*—partially correct with notable gaps; *2 (Poor)*—largely incorrect or incomplete; *1 (Inadequate)*—completely incorrect or irrelevant. This metric is applied to **Cert-QA**, **Runbook**, **Sum**, and **RB-Text** datasets.
- **Similarity (0–1 scale):** For chart generation tasks, we compute a composite similarity score between the model's output data series and the ground-truth series. The metric combines three components: (1) normalized Dynamic Time Warping (nDTW) similarity, which measures sequence alignment under temporal distortion; (2) cosine similarity, which captures directional agreement between value vectors; and (3) Pearson correlation, which measures linear relationship strength. The final score is the geometric mean of these three components (each normalized to $[0, 1]$), producing a holistic measure of data fidelity. This metric is applied to **Line** and **Bar** chart generation tasks, where the model must produce JSON specifications containing numerical data arrays.
- **ReasoningJudge (5-point scale):** A specialized LLM-as-a-judge evaluator for structured JSON outputs. A GPT-4.1 judge first validates that the model's output is well-formed JSON conforming to a provided schema, then performs field-by-field comparison against the ground truth. String fields require exact matches; reasoning arrays allow semantic equivalence with paraphrasing permitted if all factual points are preserved. Schema violations cap the maximum score at 2. This metric is applied to **Anom** anomaly detection, **Canvas** slot-filling, **RB-JSON** conditional evaluation, and **Hard** multi-hop reasoning tasks, which require structured outputs with explicit reasoning chains.
- **ExactMatch (0–1 scale):** A deterministic evaluator that compares the model's answer against the ground truth via exact string matching (case-insensitive, after whitespace trimming). A score of 1.0 is assigned if the answer matches exactly, and 0.0 otherwise. The final score is the mean accuracy across all examples. This metric requires no LLM judge and is fully reproducible. It is applied to **TOON-QA**, where answers are short deterministic values (integers, floats, or short strings) that admit unambiguous verification.

The complete evaluation prompts are provided in Appendix B.

Token Usage We measure the *total token consumption* for each method by summing the prompt and completion tokens across all examples in a dataset. Token counts are obtained directly from the model's API response. This metric captures the end-to-end cost of processing an entire benchmark, enabling direct comparison of encoding efficiency across methods.

Latency We report the *mean end-to-end latency* per query, measured as the wall-clock time from request submission to response completion. For each dataset, we compute the average latency across all examples, providing a practical measure of user-perceived response time under each encoding strategy.

E.8 Detailed Main Results Analysis

Table 1 in the main paper summarizes our main results across 11 benchmarks. On several data-intensive benchmarks, HYVE reduces total model-token use by approximately 50–90%, while maintaining or improving answer quality; other tasks show smaller savings or little change. Across the three model families, HYVE reduces latency on most task groups. The exceptions are GPT-4.1 TOON-QA and RB-Text, and Claude Sonnet 4.5 Canvas and TOON-QA.

Knowledge QA and Text Summarization The Cert-QA, Runbook, Sum, and RB-Text datasets evaluate domain knowledge and text-generation capabilities, where inputs contain moderate context and outputs are free-form text. On these tasks, HYVE achieves comparable quality to the baselines: for GPT-5, Cert-QA remains at 4.49 and Runbook improves slightly from 4.55 to 4.58; for GPT-4.1, Cert-QA changes from 4.36 to 4.38 and Runbook from 4.26 to 4.32. The reported 95% intervals for GPT-4.1 Cert-QA and GPT-5 Runbook in Appendix E.13 include zero. We therefore describe these near-ceiling differences as comparable quality, not reliable improvements; the other small differences are presented descriptively without significance claims. Token usage and latency show minimal change on Cert-QA, Runbook, and Sum, as these datasets do not contain the long arrays that benefit from hybrid-view compression. RB-Text is the exception: while quality remains comparable, GPT-4.1 latency increases from 2.74s to 4.53s due to Mode 3 overhead.

Structured Chart Generation The Line and Bar datasets require generating JSON chart specifications containing long data-point arrays. Here HYVE yields substantial efficiency gains and, most visibly on Line, higher chart similarity. For GPT-5, Line chart similarity increases from 0.68 to 0.97 (+43%), and Bar chart similarity from 0.85 to 1.00 (+18%). Token usage drops dramatically: Line from 4.2M to 0.75M tokens (–82%), Bar from 1.8M to 0.25M tokens (–86%). Latency decreases as well: Line from 125s to 40s (–68%), and Bar from 75s to 13s (–83%). The quality gains come from data backfilling, which restores arrays that would otherwise be truncated during generation. For GPT-4.1, Table 1 shows Line similarity rising from 0.41 to 0.95 and reported latency declining from 47.86s to 8.93s. Claude Sonnet 4.5 shows the same chart-generation pattern: Line similarity rises from 0.33 to 1.00 and Bar from 0.71 to 0.97, while total model tokens fall from 3.5M to 1.4M and from 1.8M to 931.3K, respectively.

Reasoning over Large Structured Context The Anom, Canvas, RB-JSON, and Hard datasets involve deeply nested JSON inputs and require structured reasoning outputs. For GPT-5, Anom increases from 3.28 to 3.77, benefiting from the anomaly operator that extracts high-signal summaries from raw time-series data, and Hard improves from 4.33 to 5.00, attributing to the relational structure preserved in the row view for multi-hop reasoning. On Canvas and RB-JSON, quality remains broadly stable: Canvas stays in the 4.94–4.96 range, and RB-JSON improves modestly from 4.84 to 4.92. Token savings are substantial: Canvas drops from 122.8M to 38.2M tokens (–69%), Anom from 18.5M to 10.9M (–41%), and Hard from 80.5K to 20.3K (–75%). These results demonstrate that hybrid-view compression effectively reduces large nested contexts without sacrificing the information needed for accurate reasoning. Figure 5 shows per-sample distributions on the Anom dataset: HYVE shifts reasoning scores toward higher values while using significantly fewer tokens per sample.

Structured Retrieval and Aggregation with Exact-Match Evaluation TOON-QA [6] combines field retrieval, aggregation, and multi-condition filtering, and is evaluated with a deterministic exact-match metric that requires no LLM judge. The mix includes direct lookups as well as questions requiring aggregation or filtering across records; it is not exclusively a range-query benchmark.

Table 3: Complete TOON v4.1.0 evaluation with GPT-4.1 (244 questions). Model tokens are the average per question.

Method	Exact match	Model tokens/question
GPT-4.1	0.607	6,905
GPT-4.1 + HYVE	0.943	4,084

TOON-QA highlights two complementary benefits of HYVE. For GPT-4.1, HYVE raises exact-match accuracy from 0.47 to 0.93 (+98%) while reducing token usage from 1.3M to 153.8K (−88%). For GPT-5, the quality gain is smaller, from 0.96 to 0.98, because the baseline is already near ceiling. Even so, HYVE still substantially improves efficiency: token usage drops from 1.4M to 182.4K (−87%), and latency falls from 5.48s to 2.71s (−51%). For Claude Sonnet 4.5, TOON-QA exact match rises from 0.49 to 0.99 and total model tokens fall from 1.5M to 800.4K, while latency rises from 1.91s to 3.26s.

On this mixed structured-data QA benchmark, HYVE substantially improves answer accuracy with GPT-4.1 and Claude Sonnet 4.5. GPT-5 starts near ceiling, so its main benefit is lower latency and token cost. Notably, GPT-4.1 + HYVE (0.93) approaches GPT-5 without HYVE (0.96), suggesting that structured context engineering can partially offset model-capability differences on these tasks.

The aggregate ablation in Table 2 shows that postprocessing as a whole matters for TOON-QA: disabling both recovery modes (Full − post) drops accuracy from 0.93 to 0.38. This intervention does not isolate Mode 3; Table 9 isolates its effect against a matched Full control. In contrast, preprocessing ablations have milder effects: removing ranking yields 0.91 (−2%) and disabling truncation yields 0.87 (−6%). Together, these results indicate that bounded recovery contributes strongly on this mixed question set.

TOON-QA also illustrates the latency trade-off of Mode 3. For GPT-5, HYVE reduces latency from 5.48s to 2.71s (−51%) because token savings outweigh the cost of the extra LLM call. For GPT-4.1, however, latency rises from 1.35s to 2.33s because the baseline is already fast and the second call dominates. A similar trade-off appears on RB-Text, where HYVE's latency (4.53s) exceeds the baseline (2.74s) for the same reason; in the GPT-4.1 and GPT-5 rows, latency decreases on the remaining benchmarks.

E.9 Complete TOON v4.1.0 Retrieval Benchmark

The TOON-QA rows in Table 1 use all 154 questions of the four-dataset release. We also evaluate all 244 questions of the official v4.1.0 release in beautified-JSON format. Quality is deterministic exact-match accuracy; the two releases are reported separately rather than pooled.

On this version, HYVE improves exact match while using 40.9% fewer model tokens per question.

E.10 LongTableBench Exact-Matching Subsets

We evaluate LongTableBench [23] using the released single-table 32k and infinite-context exact-matching subsets, with 41 and 20 questions, respectively. Within each pair, Raw and HYVE use the same question, table, output instruction, model snapshot, and decoding settings. Quality is macro-F1 over answer lists.

On the 32k subset, HYVE reduces tokens by 73.1% with GPT-4.1 and 73.3% with GPT-5, while maintaining or improving macro-F1; latency increases in both pairs. On the infinite-context subset, it improves macro-F1 under both models; with GPT-5 it reduces tokens by 95.2% and latency by 67.6%.

E.11 Generic Tool Agents on TOON-QA

We test whether a generic tool-using agent can replace HYVE's focused harness on the same 154-question, beautified-JSON TOON-QA set used for the TOON-QA column of Table 1. The agent can inspect the complete, unmodified request in a read-only file using metadata and bounded range reads, grep-style regular-expression search, and sandboxed Python. The SQL-enhanced variant additionally

Table 4: LongTableBench exact-matching subset results. Tokens and latency are reported per example.

Subset	Method	Macro-F1	Tokens/example	Latency/example (s)
32k/EM ($n = 41$)	GPT-4.1 Raw	0.888	19,677	2.53
	GPT-4.1 + HYVE	0.906	5,297	3.73
	GPT-5 Raw	0.906	20,358	10.16
	GPT-5 + HYVE	0.906	5,431	15.81
inf/EM ($n = 20$)	GPT-4.1 Raw	0.691	69,735	6.06
	GPT-4.1 + HYVE	0.749	4,967	6.56
	GPT-5 Raw	0.707	73,529	51.41
	GPT-5 + HYVE	0.750	3,534	16.65

Table 5: The 154-question TOON-QA comparison. Raw and HYVE rows reproduce the TOON-QA aggregates in Table 1; each tool-agent row evaluates all 154 questions. Quality is deterministic exact match; model tokens are dataset totals, and latency is reported per example, including tool execution for the agents.

Model	Configuration	Exact match	Total model tokens	Latency/example (s)
GPT-4.1	Raw	0.47	1.3M	1.35
	Generic tools	0.721	6.4M	16.37
	Generic tools + SQL	0.747	3.2M	18.10
	HYVE	0.93	153.8K	2.33
GPT-5	Raw	0.96	1.4M	5.48
	Generic tools	0.961	5.1M	23.18
	Generic tools + SQL	0.929	5.7M	28.14
	HYVE	0.98	182.4K	2.71

uses a benchmark-agnostic ingester that materializes detected JSON into request-scoped SQLite tables for schema inspection and read-only queries. Both configurations use a fixed eight-turn budget.

The SQL arm's database is constructed from the request rather than supplied as an existing task database. Generic tools improve GPT-4.1 accuracy over Raw, but HYVE remains more accurate and uses substantially fewer model tokens and less latency than either configured tool agent under both model families.

E.12 Claude Sonnet Model and Judge Generality

Table 1 reports paired Claude Sonnet 4.5 Raw/HYVE results alongside the GPT-4.1 and GPT-5 rows. Each Sonnet pair uses the same items, model snapshot, and decoding settings. Quality uses the task-specific metrics and judges specified for the corresponding columns; total model tokens are dataset totals and latency is reported per example.

Across these 11 groups, Sonnet+HYVE quality improves or stays within 0.01 of Raw; model tokens decrease in ten groups and are unchanged at displayed precision for Cert-QA. Latency decreases in nine groups. Canvas and TOON-QA retain the added-call latency trade-off shown in the table.

We separately rescore the eight groups requiring an LLM judge using a Claude Sonnet judge. This test changes the evaluator family; its 1–5 scores must not be mixed with the scores in Table 1. Each Raw/HYVE output pair is assessed by the same Sonnet judge.

Under this independent judge, HYVE improves seven of eight groups and matches Cert-QA within 0.01 at the displayed precision.

E.13 Repeated-Run Uncertainty and Threshold Sensitivity

Table 7 reports ten-run means and standard deviations (SD) for selected comparisons. The two-sided 95% Welch confidence intervals are for the HYVE-minus-Raw difference, calculated from the run-level means.

Table 6: Independent Claude Sonnet-judge quality scores (1–5). These are not the Table 1 judge scores.

Task	Sonnet Raw	Sonnet + HYVE
Cert-QA	4.40	4.39
Runbook	3.81	3.87
Anom	2.50	2.72
Sum	4.43	4.47
Canvas	4.40	4.84
RB-Text	4.79	4.85
RB-JSON	4.51	4.58
Hard	3.21	4.77

Table 7: Ten-run variability and 95% confidence intervals.

Comparison	Raw mean $\pm$ SD	HYVE mean $\pm$ SD	Difference (95% CI)
GPT-4.1 Cert-QA	4.36 ± 0.027	4.38 ± 0.032	$+0.020$ $[-0.008, 0.048]$
GPT-5 Runbook	4.55 ± 0.034	4.58 ± 0.031	$+0.030$ $[-0.001, 0.061]$
GPT-4.1 Line	0.41 ± 0.029	0.95 ± 0.012	$+0.540$ $[0.518, 0.562]$
GPT-4.1 Anom	3.22 ± 0.023	4.03 ± 0.030	$+0.810$ $[0.785, 0.835]$

The Cert-QA and Runbook intervals include zero, so we describe their near-ceiling mean scores as comparable, not as reliable improvements or proof of statistical equivalence. The Line and Anom intervals remain above zero under the reported ten-run analysis.

We evaluate five capacity-like settings with a one-at-a-time, ten-repetition TOON-QA sweep (Table 8). Analogous to cache capacity in traditional systems, these settings determine the size of the visible working set and therefore affect the amount of data processed. They do not control retention: the complete structured payload remains in session-scoped state. The prompt acts as a datastore-backed cache, and misses remain recoverable through SQL backfilling (Mode 2) or one bounded synthesis call (Mode 3). Each accuracy change is in percentage points relative to the 0.93 default, with entries aligned to the tested values in the same row.

Within the normal operating ranges shown, accuracy changes remain between -1.3 and $+0.6$ points. The truncation-threshold stress test at 100 instead drops by 18.8 points, showing that delaying recovery activation too far is not benign.

E.14 Ablation Study

We conduct ablation studies to isolate the contribution of each HYVE component. Table 2 in the main paper organizes ablations along two dimensions. *Preprocessing ablation*: we selectively disable ranking (Full $-$ rank), truncation (Full $-$ trunc), or swap the serialization format (Full $+$ TOON), while keeping the full postprocessing pipeline (Mode 1+2+3) active. *Postprocessing ablation*: we disable all postprocessing (Full $-$ post). We evaluate on tasks where these components have measurable impact: chart generation (Bar), anomaly detection (Anom), slot-filling (Canvas), text summarization (RB-Text), multi-hop reasoning (Hard), and structured-data question answering (TOON-QA).

Ranking Disabling reference-guided ranking reduces quality on tasks that require selective attention to relevant data. Without ranking, query-relevant records are less likely to appear in the visible context, so the system must rely more heavily on SQL-based recovery when the needed evidence lies beyond the retained subset. RB-Text drops from 4.89 to 4.56 (–7%) and TOON-QA from 0.93 to 0.91 (–2%), indicating that exposing the most relevant records directly in context is slightly more robust than recovering them indirectly through additional SQL queries.

Token usage changes less consistently. It decreases slightly on Bar (132.5K$\rightarrow$115.1K) but increases on Canvas (35.1M$\rightarrow$37.8M), Hard (15.1K$\rightarrow$16.2K), and especially TOON-QA (153.8K$\rightarrow$255.8K), likely because more cases require SQL-based recovery once relevant records are no longer surfaced in the prompt.

Table 8: Sensitivity of TOON-QA exact-match accuracy to five preprocessing settings. Deltas are percentage points relative to the 0.93 default.

Parameter (default)	Tested values	Accuracy change (pp)
Truncation threshold (30)	20 / 30 / 40 / 100	-1.0 / 0.0 / -0.5 / -18.8
Retained entries (11)	7 / 11 / 15 / 29	-1.3 / 0.0 / 0.0 / $+0.6$
Row-view top-k (8)	4 / 8 / 12 / 20	-0.9 / 0.0 / $+0.1$ / $+0.3$
SQL row-view top-k (5)	3 / 5 / 10 / 50	$+0.3$ / 0.0 / $+0.2$ / $+0.1$
Ranking top-k (30)	15 / 30 / 60 / 200	-0.1 / 0.0 / -0.7 / $+0.2$

Table 9: Mode-specific ablations with matched Full controls on a distinct system revision. Quality uses chart similarity on Bar and exact-match accuracy on TOON-QA; missing or unscorable outputs count as zero in the full example denominator.

Task	Examples	Disabled path	Full quality	Disabled quality
Bar	100	Mode 2 backfilling	1.000	0.310
TOON-QA	154	Mode 3 retrieval/synthesis	0.896	0.338

Truncation Without truncation, token usage increases dramatically: Canvas rises from 35.1M to 118.8M (+239%), Bar from 132.5K to 928.7K (+601%), Hard from 15.1K to 74.7K (+395%), and Anom from 8.9M to 15.7M (+76%). Latency increases accordingly. Disabling truncation also degrades quality on Bar (0.73 vs. 0.99) and Hard (4.10 vs. 5.00, –18%), suggesting that overly long contexts may induce context rot, causing the model to miss relevant structure. On Canvas, by contrast, the no-truncation variant nearly matches HYVE's quality (4.93 vs. 4.95) despite substantial token increases, likely because Canvas inputs often contain relatively clear answers. These results confirm that truncation primarily serves token efficiency; the large overhead on data-intensive tasks (3–7$\times$ increases) underscores the cost of removing it in long-context settings.

Encoding Replacing beautified JSON with TOON encoding yields token savings: Canvas (-18%, 35M$\rightarrow$29M), RB-Text (-6%, 48K$\rightarrow$46K), and Hard (-15%, 15K$\rightarrow$13K). Despite these token savings, TOON can degrade output quality; HYVE therefore uses beautified JSON by default.

Postprocessing Disabling all postprocessing (Full $-$ post, i.e., Mode 1 only) reveals the combined impact of Mode 2 and Mode 3. Chart generation degrades severely: Bar chart similarity drops from 0.99 to 0.06 (–94%), confirming that LLMs often preserve the overall output schema while truncating data arrays mid-generation; without Mode 2's template-based backfilling, the missing values cannot be recovered. Tasks relying on SQL-augmented reasoning (Mode 3) also degrade substantially: Anom drops from 4.03 to 3.41 (–15%), as the anomaly-detection operator (exposed as a `DETECT_ANOMALY` SQL function; see Section E.2) is no longer available. Hard drops from 5.00 to 3.93 (–21%), confirming that multi-hop reasoning benefits from the SQL query interface. TOON-QA, which includes retrieval, aggregation, and filtering questions that can benefit from datastore access, falls from 0.93 to 0.38 (–59%). Because Anom triggers Mode 3 on nearly every sample, disabling postprocessing also reduces token usage from 8.9M to 6.2M (–30%) and latency from 4.38s to 3.20s (–27%), reflecting the overhead of the additional LLM call.

Mode-specific recovery-path ablations The aggregate Full $-$ post condition above removes both recovery paths. To distinguish their effects, we evaluate two paired GPT-4.1 controls on a distinct system revision, using the same examples and configuration within each pair. On Bar (100 chart-generation examples), we disable only Mode 2 deterministic template backfilling while leaving preprocessing and the other completion paths available. On the 154-question TOON-QA release, we disable only Mode 3 datastore retrieval plus synthesis while leaving preprocessing and the other completion paths available. These controls have their own Full scores and are not directly interchangeable with the aggregate-ablation Full values in Table 2.

The Mode 2-off Bar score includes four missing or unscorable outputs counted as zero; the Mode 3-off TOON-QA score includes 15 missing or unscorable outputs counted as incorrect. Disabling Mode 2 uses 27.7% fewer model tokens but increases latency by 169.6%, while disabling Mode 3 uses 7.6%

more tokens and increases latency by 10.5%. These matched ablations quantify the outcome-level roles of deterministic reconstruction and evidence-assisted reasoning.

E.15 Aligning Context Engineering and Domain Adaptation

The preceding experiments pair HYVE with general-purpose LLMs. A complementary direction is to combine HYVE with domain-adaptive post-training.

In anonymized internal experiments, a domain-specialized networking LLM trained on networking data outperforms frontier general-purpose models on representative networking benchmarks, with especially strong results on an expert-level networking dataset (91% vs. 88% for GPT-5). This advantage is consistent with its domain-specialized training corpus.

Adding HYVE on top of the domain-specialized model yields further gains, especially on analytics-intensive benchmarks. On Runbook, the domain-specialized model + HYVE reaches 4.61, compared with 4.58 for GPT-5 + HYVE and 4.32 for GPT-4.1 + HYVE. These results suggest that context engineering at inference time and domain-adaptive training at training time are complementary. HYVE restructures the input representation, while post-training equips the model with priors that better exploit that representation. This alignment could be strengthened further by incorporating HYVE-style intermediate representations directly into the post-training data.

E.16 Related Work

We situate HYVE at the intersection of structured-data reasoning, prompt construction, and database-inspired data organization. The most relevant prior work spans the following threads.

LLMs for Structured Data Reasoning: Large language models have been applied to a variety of structured-data tasks. Table question-answering systems such as TAPAS [24] and TaBERT [25] pre-train on table-text pairs to learn joint representations. Text-to-SQL benchmarks such as Spider [26] and BIRD [27] evaluate cross-domain semantic parsing, while recent prompting strategies [28] improve LLM performance on SQL generation. DATER [7] uses LLMs to decompose large tables and complex questions for more tractable reasoning. DATER also uses SQL-based parsing–execution–filling for question decomposition; TAP4LLM [8] samples, augments, and packs a table supplied by a table manager. These methods start from supplied tables or databases, even when their preprocessing or reasoning uses operations also found in HYVE. HYVE instead discovers compatible structured regions inside an opaque mixed request and retains the detected payload in a session-scoped datastore so omitted evidence remains available for bounded post-call recovery.

Schema-dependent tasks: FACTS [10] generates reusable offline SQL/Jinja2 templates for query-focused summaries of tables sharing a known schema. SchemaRAG [11] retrieves and prunes a predefined target output schema for information extraction from unstructured text. They address different input/output contracts from HYVE's discovery and retention of structured regions embedded in an opaque mixed input.

Structured Data Serialization: The way structured data is serialized into text can strongly affect LLM performance. Table Meets LLM [9] benchmarks structural understanding of supplied tables under different formats, content orders, prompts, and partition marks, and studies self-augmentation for table reasoning. TaBERT [25] linearizes tables row by row with special tokens to preserve structure. HYVE instead first discovers compatible tabular regions inside a mixed request, then couples complementary row/column views with a queryable store for post-call recovery of omitted structured evidence.

Prompt Compression and Context Reduction: Several techniques reduce prompt length to fit within context windows. LLMLingua [12] and LongLLMLingua [13] use smaller language models to identify and prune less informative tokens, while Selective Context [29] applies entropy-based filtering to remove redundant content. LongLLMLingua additionally supports subsequence recovery of entity strings in generated answers. HYVE differs in retaining the complete detected structured payload in a session-scoped queryable store, so omitted records can be retrieved for deterministic backfilling or bounded synthesis rather than relying on token selection as the principal data-retention mechanism.

Agent Context Management: General-purpose coding agents and agent runtimes, including OpenClaw [30], Claude Code [17], Codex [31], Gemini CLI [32], OpenCode [33], and Pi [34], support multi-step, tool-using workflows. HYVE addresses a distinct context-management problem: at the model-call boundary, it reduces the model-visible representation of machine-data-heavy input while retaining the detected structured payload in a queryable datastore for exact recovery. This session-scoped mechanism can complement general-purpose agent workflows.

Evaluation boundaries: SWE-bench [35] evaluates repository-level software issue resolution, BIRD-CRITIC [36] SQL issue debugging in database applications, and NL2SQL-BUGs [37] semantic-error detection for supplied natural-language queries, schemas, and SQL. These tasks differ from organizing an opaque mixed request before model inference.

Constrained Generation and Output Repair: Ensuring well-formed structured outputs from LLMs has received significant attention. Constrained decoding methods such as PICARD [38] enforce grammar constraints during autoregressive generation. LMQL [39] provides a query language for declarative prompting with type constraints, and grammar-constrained decoding [40] uses formal grammars to guide generation toward valid structured outputs. Synchromesh [41] uses constrained semantic decoding to enforce syntactic and semantic constraints during code generation. These approaches constrain generation to improve the validity of model outputs. HYVE addresses a different failure mode: outputs that are structurally valid yet *data-incomplete* because arrays have been truncated.

Hybrid Storage in Database Systems: Column-oriented storage [42] organizes data by attribute rather than by record, enabling efficient analytical queries over large datasets. Hybrid transactional/analytical processing (HTAP) systems [43–45] combine row stores for transactional workloads with column stores for analytical workloads, thereby avoiding costly ETL pipelines. HYVE draws inspiration from this principle of workload-adaptive data organization: its hybrid view uses a column-oriented representation for analytics-oriented tasks such as anomaly detection and statistical summarization, and a row-oriented representation for retrieval-oriented tasks such as selecting relevant entities and relationships. This parallel suggests that workload-aware data organization remains valuable even when the execution engine is an LLM rather than a database system.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contribution as analytics-oriented context engineering for machine-data-heavy settings, with emphasis on token reduction and preserved or improved quality. These claims are supported by the design description in Sections 1 and 2 and by the experiments in Section 3.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated Limitations subsection discussing HYVE's within-session datastore in relation to long-term memory and its current focus on JSON / Python-AST-like structures; see Section 4.2. It specifies the public evaluation's structured-data question-answering scope and distinguishes this task from repository repair and SQL debugging over a supplied database.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present theorem-proof style theoretical results. It introduces formal definitions and system design components, but no claims requiring formal proofs are made. Formal definitions of schema consistency, column-oriented views, and row-oriented views are given in Appendix D.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The main experiments are reproducible within the stated data-access constraints: Appendix E.5, Appendix E.4, and Appendix E.7 document the evaluation pipeline, benchmark sizes, task definitions, metrics, model configuration, 10-run averaging protocol, and token and latency accounting. Because proprietary internal datasets cannot be released under applicable privacy and corporate policies, we use the public TOON-QA benchmark to provide an externally verifiable evaluation. Appendix E.10 specifies the evaluated public LongTableBench subsets, their exact task coverage, and the macro-F1 metric. Implementation details for the core components are documented in Appendices C.1, C.2, D, and E, including Algorithms 1, 2, 3, and 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Most evaluated datasets are proprietary and cannot be publicly released under applicable corporate policies, as stated in Section 3 and Appendix E.4. The implementation has not yet been approved for open-source release because it has recently been integrated into a commercial product. Therefore, this submission does not provide an open-source release of the system, but the paper documents the design and evaluation details needed to support reproducibility within these constraints.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper describes the baselines, evaluated datasets, quality metrics, token accounting, latency measurement, and ablation settings in Sections 3–3.3 and Appendix E.4, Appendix E.7, and Appendix E.14. These details are sufficient to understand how the reported results are produced. Appendices E.10–E.13 specify the public-benchmark subsets, generic-tool configurations, Sonnet judge distinction, and threshold sweeps.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Appendix E.13 reports ten-run means and standard deviations for four key comparisons, together with two-sided 95% Welch confidence intervals for HYVE-minus-Raw differences computed from run-level means. The evaluation protocol and metric definitions are described in Appendices E.5 and E.7.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Detailed compute-resource information is provided in Appendix E.6.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: To the best of our knowledge, the research described in the paper conforms to the NeurIPS Code of Ethics. The work studies context engineering and evaluation for machine-data reasoning and does not involve human-subject experimentation or unsafe data-collection procedures.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The current submission does not include a dedicated broader-impacts discussion. The paper focuses on technical design and empirical evaluation of context engineering for machine-data reasoning.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a high-risk generative model or a scraped dataset requiring a special safeguards discussion. This study investigates context engineering techniques to facilitate the organization of structured machine data during inference.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Existing third-party assets and services used in the paper are credited through the corresponding citations and used under their stated licenses or service terms. The TOON encoding and TOON-QA benchmark are cited via the TOON project [6], which is released under the MIT License (`https://github.com/toon-format/toon/blob/main/LICENSE`). DuckDB is cited as the embedded analytical database used by HYVE [19] and is also distributed under the MIT License. LangSmith is cited as the hosted evaluation platform used for experiment orchestration [22] and is used subject to LangChain's LangSmith Platform Terms of Service (`https://www.langchain.com/terms-of-service`). The evaluated OpenAI and Anthropic models are documented in their respective sources [2–5]; model inference uses hosted services under the applicable access terms. We do not redistribute third-party model weights, proprietary service assets, or proprietary datasets as part of this submission; the paper reports aggregate results and limited illustrative benchmark examples only.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper includes dataset descriptions and illustrative examples, but no new dataset, code, or model asset is released with it. The complete evaluated datasets and implementation are not released under applicable corporate policies.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not include crowdsourcing experiments or research involving human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research on human subjects, so IRB review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a central part of both the method and the evaluation. The paper explicitly describes the use of GPT-4.1, GPT-5, and Claude Sonnet 4.5 as task models in Section 3, and it also documents LLM-as-a-judge evaluators in Appendix E.7 and Appendix B.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.