Every Answer Has Its Own Path: Agentic Routing for Retrieval-Augmented Table Question Answering

Anonymous ACL submission

Abstract

Documents that users query mix tables with prose, but the QA systems built for them do not: retrieve-and-read pipelines flatten tables, while text-to-SQL pipelines ignore passages. Corporate filings, scientific reports, and Wikipedia articles all sit on this fault line, and a single user question routinely crosses it. We present ATR (AgenticTableRAG), an agentic framework for hybrid table-text question answering. ATR decomposes each question into ordered sub-queries and dispatches each to one of four execution primitives: text reading, cell retrieval, SQL execution, or a hybrid of cell grounding and SQL. When the retrieved evidence does not support the predicted answer, the router is re-invoked with the failed-route history and dispatches a different primitive. Value grounding composed with constrained SQL execution rules out column and value hallucinations by construction. The routing policy is distilled into a lightweight student model that transfers across benchmarks without additional inference-time LLM calls. A single ATR configuration leads on token F1 across three benchmarks without per-dataset tuning.

1 Introduction

In practice, document QA spans both modalities at once. A single user question often combines a table-row lookup with a passage read drawn from the same document, two sub-queries that call for different reasoning primitives.

Because each sub-part of such a question calls for a different primitive, *agentic QA* is a natural framework for this setting: an agentic-QA system decomposes the input and selects among several execution primitives (retrieval, SQL, Python sandboxing) instead of committing to a fixed backbone. Existing agentic-QA systems for hybrid table-text QA, however, still pre-commit to one primitive at the question level, whether sandboxed Python over retrieved indices (Chen et al., 2024), SQL cross-validated with textual retrieval (Yu et al., 2025), or passage retrieval (Yao et al., 2023); none recover when the chosen tool fails on a sub-part that another primitive would have handled. The bottleneck is not the choice of primitive but the *unit of decision*: decomposition, routing, and re-routing must all operate at the sub-query level.

Question

*What historic place is in the city where **U.S. Highways 2 and 169** are the main routes?*

TEXT PASSAGES

The region is known for its mining history.

U.S. Hwys 2 and 169 pass through La Prairie.

The population has grown steadily since 1990.

(1) Multi-hop cross-modal

RELATIONAL TABLE

City	···	Site	Year
Grand Rapids	···	Old Central School	1895
Hwy 2/169 @ La Prairie	···	**Frank Gran Farmstead**	1920
Coleraine	···	Canisteo District	1907

Answer found via cross-modal bridge

(2) Value alignment: `'Highways'` (Q) ∼ `'Hwys'` (text) ∼ `'Hwy 2/169'` (cell)

Figure 1: A HybridQA example exhibiting the two failure modes from §1; **(1)** marks the text→table bridge, **(2)** marks the surface-form mismatch.

This question-level commitment shows up as two recurring failure modes (Figure 1). **(1) Multi-hop cross-modal trajectory.** Retrieval-only agents stop at the text passage, and pure NL2SQL searches the table without the text clue, so neither carries intermediate clues across modalities. **(2) Lexical mismatch between surface forms.** Embedding-based retrieval can rank the correct cell as a candidate, but the SQL executor demands exact surface-form matching that retrieval similarity does not guarantee.

To address this question-level commitment, we propose ATR (AgenticTableRAG), in which routing happens at the sub-query level (Figure 2). ATR

decomposes each input question into ordered sub-queries; for each sub-query, a router selects one of four execution primitives (text reading, cell retrieval, SQL, or a hybrid of cell grounding and constrained SQL); the selected primitive returns a candidate sub-answer; and a verifier validates the sub-answer, re-routing it to a different primitive when the evidence is insufficient. To make sub-query-level routing efficient at inference time, the router itself is distilled from an LLM-prompted teacher into a lightweight student that incurs no per-sub-query LLM call.

Empirically, a single ATR configuration improves over the strongest agentic prior on HybridQA by $+37.2$ F1 and $+15.2$ points on an LLM-judged correctness metric, and matches or exceeds the strongest prior system on TAT-QA and WTQ. Ablations isolate the contribution of routing, value linking, and re-routing on failure (Section 6).

Our contributions can be summarized as follows:

- **Identification of a Key Challenge.** We identify the *soft-retrieval / hard-execution gap* in agentic table-text QA: embedding retrieval can rank the correct cell as a candidate while SQL execution still misses it due to surface-form mismatch between the question and stored values.

- **Retrieval-grounded Constrained Execution.** Before any SQL is issued, ATR runs a value-linking step that grounds each question entity against the cell index, mapping user mentions to canonical cell forms. This eliminates the surface-form mismatch that causes pure NL2SQL to miss the row.

- **Empirical Validation.** Through extensive experiments on hybrid table-text benchmarks, we validate that a single ATR configuration matches or exceeds prior agentic systems and reduces per-sub-query LLM cost to zero via the distilled router.

2 Related Work

Agentic and iterative retrieval. Recent agentic-QA systems extend retrieval-augmented generation (Lewis et al., 2020) with iterative tool selection but typically commit to a single execution strategy at the question level. General-purpose agents such as ReAct (Yao et al., 2023), Self-Ask prompting (Press et al., 2023), and Reflexion (Shinn et al., 2023) explore tool decomposition and self-critique; Self-RAG (Asai et al., 2024) adds reflection-token gating to retrieval; Program-of-Thoughts (Chen et al., 2023) executes reasoning as Python code rather than natural language. Within tables, TableRAG-R (Chen et al., 2024) indexes schema, rows, and cells and executes inside a Python sandbox; TableRAG-S (Yu et al., 2025) cross-validates SQL with textual retrieval. ODYSSEY (Agarwal et al., 2025) reports strong HybridQA numbers by traversing a bidirectional cell/passage graph specific to Wikipedia cell-link tables; we exclude it from the headline comparison because the same graph is undefined on TAT-QA and WTQ. ATR differs in (i) sub-query-level rather than question-level routing, (ii) a fixed four-primitive vocabulary closed under failure-conditioned re-routing, and (iii) distillation of the routing policy into a lightweight classifier.

NL2SQL and constrained execution. Modern NL2SQL maps natural language to executable SQL through pretrained models such as SQL-PaLM (Sun et al., 2023), execution-guided methods like Binder (Cheng et al., 2023) and Codex SQL (Rajkumar et al., 2022), and decomposition approaches like DIN-SQL (Pourreza and Rafiei, 2023) that break SQL generation into sub-tasks with self-correction. These approaches improve the SQL syntax itself but inherit the lexical-mismatch failure of pure NL2SQL: a syntactically perfect query that emits `'tv'` still misses a cell stored as `'television'`. ATR uses NL2SQL as one of four routes and pairs it with the HybridValueLinker, which grounds every value referenced in `WHERE` clauses against the cell index, addressing the data-grounding bottleneck rather than the syntax bottleneck.

Lightweight distilled controllers. Distillation into smaller students is well studied for classification (Sanh et al., 2019; Jiao et al., 2020), and LLM-routing work such as RouteLLM (Ong et al., 2024) and FrugalGPT (Chen et al., 2024) learns to dispatch between strong and weak LLMs to reduce inference cost. ATR's router instead dispatches between execution primitives (text reading, cell retrieval, SQL, hybrid) for a fixed LLM backbone, a different unit of routing decision. Distilling such an LLM-prompted router for an agentic pipeline, where the teacher emits a discrete route label rather than a sequence, is underexplored. ATR's student classifier recovers 98.6% of the teacher's routing

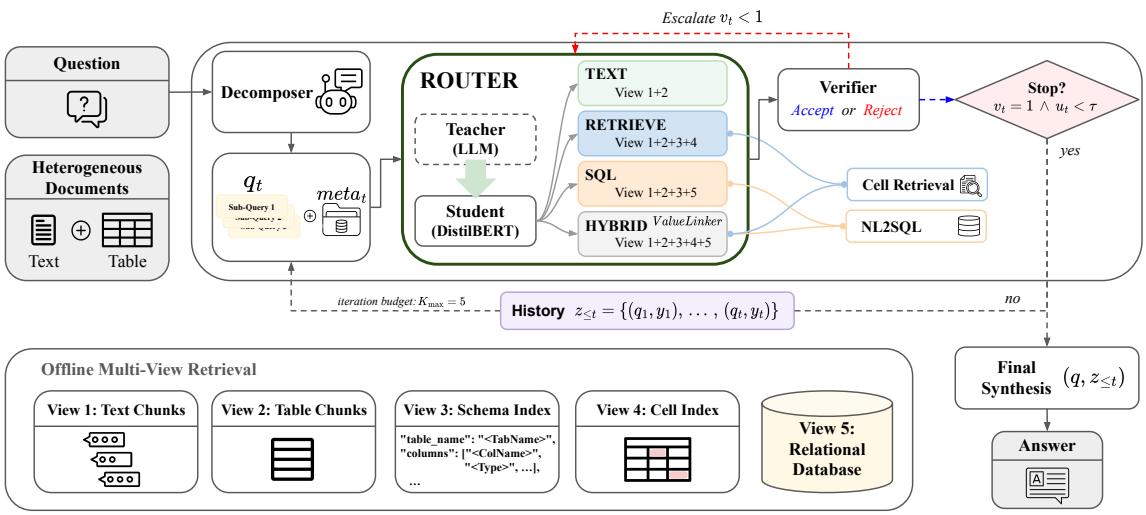

Figure 2: ATR online inference loop (Algorithm 1): Decomposer, distilled Router, four execution primitives over a five-view index, with re-routing on failure.

decisions and transfers across benchmarks without further supervision.

3 Method

3.1 Overview

ATR answers a question q through an iterative loop of decomposition, routing, execution, and verification (Figure 2; Algorithm 1). At each iteration $t \in \{1, \ldots, K_{\max}\}$ the decomposer emits a sub-query q_t conditioned on previous sub-answers $z_{<t}$, the router dispatches it to one of four primitives $r \in \mathcal{R} = \{\textsc{Text}, \textsc{Retrieve}, \textsc{Sql}, \textsc{Hybrid}\}$, the primitive produces an intermediate answer y_t, and a verifier returns $\nu_t \in \{0, 0.5, 1\}$. When $\nu_t < 1$, the router re-selects given the updated failed-route history $H_{<t}$ and the sub-query is re-executed. The loop exits when $\nu_t = 1 \wedge u_t < \tau$ (u_t is the decomposer's residual uncertainty) or after $K_{\max}$ iterations; an optional final synthesis step fuses the sub-answers into the final answer a.

Two design choices distinguish ATR from prior agentic table-QA systems. Routing happens at the sub-query level, so a single question can mix primitives. And when the verifier rejects an answer, the router (rather than a free-form LLM patch) picks the next primitive conditioned on which routes have already failed, making the rescue step principled rather than open-ended. The full pseudocode is given in Algorithm 1 (Appendix A).

3.2 Offline Indices

ATR shares a single set of offline indices across all four execution primitives. **View 1** (text chunks) splits each passage with a recursive character splitter; **View 2** (table chunks) serializes each table into row-aligned chunks together with a mapping f from each chunk back to its source table and schema; **View 3** (schema index) embeds each column header to support schema retrieval; **View 4** (cell index) embeds each `(column, value)` pair up to a per-table budget of B cells; and **View 5** is a relational database served by an external SQL endpoint. Each route consumes a different subset of these views (§3.5); the indices themselves are route-agnostic.

3.3 Decomposition and Sub-Query Schema

At each iteration t, the decomposer samples the next sub-query conditioned on the original question and the accumulated sub-answer history,

$$q_t \sim p_{\text{DEC}}\big(\cdot \mid q,\ z_{<t}\big), \quad z_{<t} = \{(q_s, y_s)\}_{s=1}^{t-1},$$

where the decomposer is a prompted LLM that emits a structured $\textsc{SubQuery}$ object rather than a free-form question; its six fields, types, and downstream consumers are listed in Appendix Table 4, and the full decomposer prompt is given in Appendix Figure 10a. The decomposer also emits the sentinel $\textsc{Terminate}$ when it judges q already answered, which ends the loop early. Multi-hop dependencies, such as a sub-query that references the result of a prior step, are handled implicitly because the previous sub-answers $z_{<t}$ are part of the decomposer's prompt context. The same JSON output additionally carries a scalar `uncertainty` $u_t \in [0, 1]$: the prompted LLM is asked to estimate,

on a fixed rubric, how likely the original question remains under-answered after the sub-queries so far (Appendix Figure 10a). This u_t gates the stop controller in §3.6: when the verifier supports the current sub-answer (ν_t=1) and the decomposer also reports low residual uncertainty (u_t<τ), the outer loop exits.

3.4 Routing

The router maps the tuple $(q_t, meta_t, schema_t, H_{<t})$ to a route $r \in \mathcal{R}$, where $meta_t$ is the SUBQUERY metadata of §3.3, $schema_t$ is the set of candidate columns retrieved from View 3 for q_t, and $H_{<t} \subseteq \mathcal{R}$ is the set of routes already attempted and verifier-rejected at earlier iterations for this sub-query (cf. §3.6). We train the router by teacher/student distillation: a zero-shot LLM router (the *teacher*) labels routes on training questions, and a compact four-class classifier (Sanh et al., 2019) (the *student*) is distilled to match the teacher's distribution. Only the student runs at inference, a lightweight classifier that runs at zero LLM cost per sub-query.

3.5 Execution Primitives

Each route in $\mathcal{R}$ is backed by a different execution primitive that shares the offline indices of §3.2 but exposes a different reasoning shape.

TEXT. Retrieved chunks from View 1 feed an answer-synthesis prompt that grounds the answer in passages; no schema or SQL is involved.

RETRIEVE. For each entity $e \in E_t$, View 4 returns top-K_v `(column, value)` candidates via embedding similarity. Because matched cells alone localize the answer row but not the answer column, we augment View 4 with a per-row index that returns the top-K_r rows R whose concatenated cell text is most similar to q_t. The synthesis prompt receives both the candidate cells and the rows R, letting the LLM resolve lexical variants like *tv* vs. *television* through the row context without committing to SQL.

SQL. The candidate columns $schema_t$ from View 3 feed an NL2SQL prompt that produces a query against View 5.

HYBRID. The HYBRIDVALUELINKER bridges retrieval and SQL: for each entity $e \in E_t$, View 4 returns the top-K_v candidate cells $V_{\text{raw}}^{(e)}$ by embedding similarity, and an LLM verifier LLM_{link} selects a single grounded value $v_e^\star = \text{LLM}_{\text{link}}(e, V_{\text{raw}}^{(e)})$, yielding the value constraint set $V^* = \{(e, c_e, v_e^\star)\}$, where $c_e \in C$ is the column matched to entity e during cell retrieval. When LLM_{link} rejects all candidates, the linker falls back through three weakening levels: *unconstrained* (drop e's constraint), `LIKE` *fuzzy* (substring match), and *reroute* (emit TEXT); routes already in $H_{<t}$ raise the minimum fallback level (e.g., if TEXT is already in $H_{<t}$ the *reroute* fallback is skipped, and the linker proceeds with `LIKE` fuzzy or unconstrained). CONSTRAINEDSQL then samples a query restricted to the columns C retrieved from View 3 and seeded with V^*. Because both C and V^* are derived from the index, the executor cannot emit a column or cell value absent from the table, eliminating column hallucination (forbidden by C) and lexical mismatch (handled by V^*).

All four primitives share a single set of offline indices, and the router's choice picks which API consumes them. The four views are quantitatively complementary on HybridQA, with no single view dominating: row retrieval covers 69% of gold tables at K=10, schema 42%, document chunks 21%, and cell values 6%; a t-SNE projection of the view embeddings (Figure 6) confirms the views encode complementary semantic signals rather than redundant copies.

3.6 Validation, Escalation, and Stop Controller

The intermediate answer y_t is validated by a prompted LLM that consumes $(q_t, y_t, \text{evidence}_t)$ and returns a three-way verdict $\nu_t \in \{0, 0.5, 1\}$ corresponding to *reject*, *insufficient*, and *support*; the rubric and full prompt are given in Appendix Figure 10c. When $\nu_t < 1$, the chosen route is added to $H_{<t}$ and the router is invoked again on the same sub-query (up to three attempts); because $H_{<t}$ is part of the router's input, each retry picks a primitive that consumes a strictly different combination of indices than the failed attempt. The stop controller exits the outer loop as soon as $\nu_t = 1$ and $u_t < \tau$, cutting the mean iteration count without losing accuracy (§6).

If the SQL primitive returns a single scalar (one row and one numeric or short-string column) that value bypasses the synthesis prompt and is recorded verbatim as y_t. This prevents the LLM from paraphrasing a precise value such as `9,461,105` into "about 9.5 million."

3.7 Final Synthesis

The optional synthesis step rereads q together with all sub-answers $\{(q_t, y_t)\}_{t=1}^{T}$ in a single LLM call. The prompt enforces a minimal output format and arithmetic rules for unit preservation and percentage formulas. On HybridQA, it improves strict Exact EM by $+4.59$ and Normalized EM by $+5.24$ points by canonicalizing verbose sub-answers, without changing Relaxed EM (§6).

4 Experimental Setup

Datasets. We evaluate on three hybrid table-text benchmarks. **HybridQA** (Chen et al., 2020) pairs Wikipedia tables with passages linked from each cell, so most questions join cell content with free text; we use the dev split standardised by Yu et al. (2025), the only split with publicly reported numbers for the strongest agentic prior. **TAT-QA** (Zhu et al., 2021) draws from financial filings and demands scale-aware arithmetic over tables adjacent to explanatory paragraphs; we use the official dev set. **WTQ** (Pasupat and Liang, 2015) provides compositional queries over standalone tables; we use the pristine-unseen split, in which the dev tables are disjoint from training.

Baselines. We compare ATR against five reference systems together with floor and oracle references. **Naive LLM** answers from the question alone (no retrieval) and serves as a contamination floor. **NaiveRAG** retrieves the top five document chunks using BGE-M3 (Chen et al., 2024) with a cross-encoder reranker and produces the answer in a single LLM call. **ReAct** (Yao et al., 2023) runs a Thought/Action/Observation loop with the same retrieval primitives and a maximum of ten steps. **TableRAG-R** (Chen et al., 2024) is the original retrieval-only TableRAG that combines BM25 with FAISS indices over schema, cell, and row representations together with a Python sandbox. **TableRAG-S** (Yu et al., 2025) extends TableRAG-R with a relational SQL backend and cross-validates each sub-query between SQL execution and textual retrieval; it is the strongest agentic prior on the LLM-judged metrics (Relax, Judge), and its strict EM and token F1 are reduced by sentence-form output that does not match the brief gold spans these scorers expect. All five baselines are re-run under the same protocol: identical Gemini 2.5 Flash backbone, identical BGE-M3 retrieval, and $sc{=}1$ (no self-consistency (Wang et al., 2023)). Dataset-specific supervised models (e.g., TagOp fine-tuned on TAT-QA) and HybridQA-specialised graph methods (e.g., ODYSSEY) are not included in this head-to-head; we discuss their separate scope in §5 (Scope of comparison).

Metrics. Every cell in Table 1 uses each dataset's official evaluator. For HybridQA we report the SQuAD-style `normalize_answer` Exact Match and token F1. For TAT-QA we report the official TagOp scale-aware Exact Match and numerical F1 from the original evaluator (Zhu et al., 2021). For WTQ we report the official Pasupat and Liang Normalized Exact Match together with token F1. As a complementary semantic signal, we also report an **LLM Judge** score on all three datasets, produced by a Gemini 2.5 Flash call with a fixed correctness rubric (Appendix Figure 10d); this complements the strict EM evaluators because agentic systems often produce semantically correct answers in slightly different surface forms that strict text matching penalises. We do not headline the substring-tolerant Relaxed Exact Match, which sentence-form predictions systematically inflate.

Implementation details. All systems share Gemini 2.5 Flash through Vertex AI as the backbone LLM for every prompted component (an open-weight Llama-3.3-70B backbone swap controlling for closed-source contamination is reported in §6.2); retrieval uses BGE-M3 (Chen et al., 2024) with the BGE-reranker-v2-M3 cross-encoder, and the SQL backend reuses the MySQL service of Yu et al. (2025). Default ATR hyperparameters: $K_{\max}{=}5$, $\tau{=}0.1$, cell-index budget $B{=}10{,}000$ per dataset; only $K_{\max}$ and τ are selected on HybridQA-train and held fixed for TAT-QA and WTQ. The DistilBERT router (66M parameters) is distilled from 9,998 HybridQA-train LLM-router labels with class-weighted cross-entropy and AdamW (Loshchilov and Hutter, 2019); inference is a ~ 1 ms GPU forward pass at zero LLM cost. Full hyperparameter, hardware, and reproducibility details are in Appendix A.

5 Main Results

One pipeline, three benchmarks. As shown in Table 1, a single ATR configuration with the learned router and default hyperparameters leads on token F1 across all three benchmarks. The HybridQA EM/Relax gap is an LLM-pipeline surface-form artifact; the LLM Judge score (54.02) sits between EM and Relax, ruling out Relax-only in-

Table 1: Main results (Gemini 2.5 Flash, BGE-M3, sc=1). **Bold** = best per column. *Judge* = Gemini Flash binary rubric (Appendix Figure 10d).

System	HybridQA				TAT-QA				WTQ			
	EM	Relax	F1	Judge	EM	Relax	F1	Judge	Norm	Relax	F1	Judge
Baselines												
Naive LLM	13.11	16.39	17.76	17.05	0.60	1.26	2.05	0.00	6.90	8.30	8.54	8.80
NaiveRAG	16.39	25.57	22.33	23.28	36.75	69.78	44.18	66.73	49.90	54.50	53.14	52.30
ReAct	18.03	25.25	25.14	25.57	21.34	47.30	27.78	45.20	45.10	47.60	47.56	47.55
TableRAG-R	11.18	18.75	15.28	18.77	13.61	33.77	16.81	34.02	51.30	56.80	55.22	56.90
TableRAG-S	5.90	44.92	16.84	38.82	1.44	38.97	9.11	56.30	16.30	**67.80**	32.06	**66.60**
Ours												
ATR (heuristic router)	38.36	56.39	49.42	49.51	37.53	68.44	43.87	71.05	56.10	62.50	60.99	61.22
ATR (LLM router)	**40.98**	55.74	51.02	52.13	38.07	**69.89**	**44.91**	**72.22**	55.30	62.30	60.63	61.32
ATR (learned router)	40.83	**61.59**	**54.02**	**54.02**	**38.55**	68.32	44.73	71.00	**57.60**	64.00	**62.81**	63.29

flation. Across every non-EM metric, ATR leads on HybridQA and TAT-QA in full and on WTQ token F1. On the two cells we do not lead (WTQ Relaxed and WTQ Judge), TableRAG-S produces sentence-form outputs that average several times the length of ATR's: ATR's short, precise answers (mean length 1.7 tokens) are structurally disadvantaged by substring-tolerant and LLM-judged metrics that reward verbosity, rather than reflecting a substantive accuracy advantage. The aggregate TAT-QA F1 gap to NaiveRAG under-reports ATR's value because a large fraction of TAT-QA dev questions are single-span extractions whose answers appear verbatim in the retrieved passage; the per-`answer_type` breakdown in Table 2 shows ATR ahead by $+5.43$ pp on `arithmetic` and $+6.25$ pp on `count`, demonstrating the structural advantage of sub-query routing on reasoning-heavy question types. Appendix D (Figure 9) traces the

Table 2: TAT-QA dev per-`answer_type` breakdown (ATR learned router vs. NaiveRAG, Gemini 2.5 Flash). n is the question count per type. **Bold** = better per cell.

Type	n	EM		F1		ΔF1
		RAG	ATR	RAG	ATR	
arithmetic	718	36.21	**41.64**	36.21	**41.64**	$+5.43$
count	32	53.13	**59.38**	53.13	**59.38**	$+6.25$
multi-span	217	29.49	**31.34**	**45.11**	43.39	-1.72
span	701	**38.80**	36.66	**51.64**	47.64	-4.01
Agg.	1668	36.75	**38.55**	44.18	**44.73**	$+0.55$

pipeline end-to-end on a HybridQA two-hop question, where ATR mixes HYBRID and TEXT within a single question, the operational signature of sub-query-level routing.

Cross-dataset transfer of the distilled router. The DistilBERT router is distilled from HybridQA-train sub-query labels emitted by the LLM teacher and never observes WTQ or TAT-QA training data. As shown in Table 1 (rows *ATR (LLM router)* vs. *ATR (learned router)*), the student matches or exceeds the LLM teacher on the headline metrics on both unseen benchmarks at zero per-sub-query LLM cost, consistent with the student having internalized operator- and modality-conditioned route preferences rather than dataset-specific lexical patterns.

Scope of comparison. We separate baselines into two categories. *Shared-protocol baselines* (NaiveRAG, ReAct, TableRAG-R, TableRAG-S) run with the same backbone, retrieval stack, and sc=1 as ATR; these are the comparisons reported in Table 1 and the only head-to-head we claim. *Dataset-specific methods*, such as supervised TAT-QA models (e.g. TagOp (Zhu et al., 2021) fine-tuned on TAT-QA-train) and HybridQA-specialised graph methods (e.g. ODYSSEY (Agarwal et al., 2025), which exploits the bidirectional cell/passage graph unique to HybridQA), achieve higher dataset-specific scores by virtue of training or architecture tied to one benchmark and are therefore not directly comparable to a zero-shot, per-dataset-tuning-free pipeline; we list them for context only.

6 Ablations and Analysis

6.1 Component ablation

Table 3 (top block) isolates each component's contribution by removing it one at a time from Full ATR. On HybridQA token F1, capping the loop at $K_{\max}$=1 collapses the system to one-shot QA at -48.8 pp (the floor of the smooth $K_{\max}$ curve

Table 3: Ablation on HybridQA dev. *Top:* component removal; *bottom:* backbone swap (learned router fixed; full matrix in Table 7). **Bold** = best per column; ΔF1 vs Full ATR.

Variant	Exact	Norm	Relax	**F1**	Judge	ΔF1
Full ATR (Gemini Flash + learned router)	37.02	40.83	61.59	54.02	54.02	—
Component removal						
− Escalation	12.46	13.77	23.61	18.32	19.08	−35.70
− Iteration ($K_{\max}$=1)	3.28	3.28	5.25	5.26	3.28	−48.76
− Decomposer (raw question)	8.52	9.18	16.07	13.49	14.43	−40.53
− Final Synthesis	33.77	36.72	58.69	48.77	51.64	−5.25
No Router (forced HYBRID)	33.11	36.72	49.51	45.76	44.26	−8.26
Backbone swap (learned router fixed)						
Gemini 2.5 Pro	38.36	42.30	**63.28**	54.58	54.10	+0.56
Claude Haiku 4.5	35.74	39.67	58.69	50.15	54.28	−3.87
Llama-3.3-70B-AWQ	39.67	**44.92**	56.39	**54.68**	**55.08**	+0.66
Qwen3.5-122B-A10B-GPTQ-Int4	**40.33**	42.62	57.05	52.76	51.15	−1.26

in Figure 5, not a binary on/off effect); removing re-routing on failure costs −35.7 pp; and replacing the decomposer with the raw question costs −40.5 pp. These contributions are not independent: HybridQA questions are multi-hop, so the iterative loop *generates* sub-queries that the router *dispatches* and the verifier *rescues*; the headline F1 reflects the *composed* system. The bottom-block reference *No Router* (forced HYBRID) shows that removing routing alone costs −8.3 pp of F1.

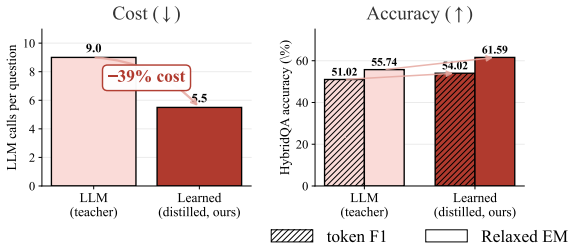

Figure 3: Accuracy and cost trade-off of router variants on HybridQA dev: mean LLM calls per question (left); token F1 and Relaxed EM (right).

6.2 $K_{\max}$, router, and backbone ablations

$K_{\max}$ saturation. Figure 5 reports the full $K_{\max}$ sweep on HybridQA dev. Relaxed climbs steeply from $K_{\max}$=1 to $K_{\max}$=3 and peaks at $K_{\max}$=5; further iterations do not help and slightly hurt on Relaxed. We interpret the regression past $K_{\max}$=5 as later iterations adding sub-queries whose sub-answers dilute rather than extend the evidence the synthesis step has to fuse: once the multi-hop chain is complete, additional decomposer calls produce tangential or already-answered sub-queries whose noisier outputs are not fully filtered by the verifier. Official Norm EM, by contrast, continues to climb slowly past $K_{\max}$=5, consistent with extra iterations buying answer-form canonicalization rather than new coverage, a pattern that mirrors the strict-EM gain we attribute to Final Synthesis (§3.7). We adopt $K_{\max}$=5 as the Pareto-optimal operating point: peak Relaxed, near-peak Norm EM, and the lowest compute among saturating settings.

Teacher vs. student. The distilled student recovers 98.6% of the LLM teacher's routing decisions on HybridQA; the resulting accuracy profile matches or exceeds the teacher on every headline metric on HybridQA, TAT-QA, and WTQ (Table 1, rows *ATR (LLM router)* vs. *ATR (learned router)*). Figure 3 plots the cost/accuracy trade-off: the student reaches the teacher's token F1 while cutting mean LLM calls per question relative to the No Router baseline that pays the linker cost on every sub-query. *A counter-intuitive finding*: the student distilled from the LLM teacher's labels outperforms a student trained on *oracle* routing labels. Oracle labels mark which route *can* solve each sub-query in isolation, but the LLM teacher's labels carry verifier-aware route preferences (which route the teacher would in practice pick given the metadata and the escalation chain, even when several routes are admissible), embedding downstream recoverability into a single label. The student internalises this preference, suggesting that the routing signal in this pipeline is closer to a downstream-cost-aware policy than to an isolated correctness classifier.

Backbone. Table 3 (bottom block) reports backbone sensitivity on HybridQA dev with the learned router fixed across one representative from each major family, spanning closed-weight (Gemini, Claude) and open-weight (Llama, Qwen); frontier

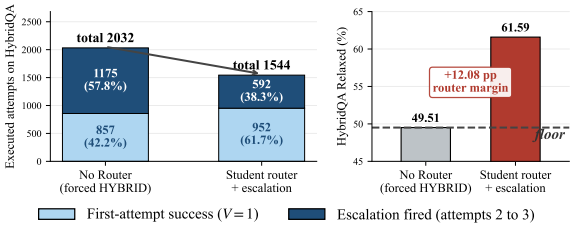

Figure 4: Router value vs. No Router (forced HYBRID): -24% attempts and $+12.08$ pp Relaxed accuracy on HybridQA.

checkpoints are reported in Appendix C (Table 7). Across this set the architectural contribution in the top block dwarfs the backbone effect by an order of magnitude: ATR's routing-and-linking pipeline is the floor that holds across closed-weight (Gemini 2.5 Pro, Claude Haiku 4.5) and open-weight (Llama-3.3, Qwen3.5) backbones alike. A self-hosted Llama-3.3 backbone via vLLM (Kwon et al., 2023) matches or exceeds the Flash headline on most metrics at zero API cost ($+0.66$ pp F1), and Qwen3.5 lands within 1.3 pp of Flash F1 in the same setting. On WTQ the same self-hosted Llama backbone trails Flash by only $\sim$1 pp on both F1 and Judge, showing that ATR's gains are not bottlenecked by closed-weight LLM access on Wikipedia-style tables. The picture is less symmetric on TAT-QA: open-weight backbones still trail closed-weight peers on the strict-arithmetic evaluator there, a gap the architecture alone does not close (see Limitations).

6.3 Where the router's value comes from

The router's contribution is a cost reduction on top of the escalation loop (Figure 4; *No Router* row of Table 3): against the No Router (forced HYBRID) baseline, the student router raises Relaxed accuracy by $+12.08$ pp on HybridQA while cutting escalation firings from 57.8% to 38.3% and mean LLM calls per question from 9.0 to 5.5.

6.4 HybridValueLinker telemetry and value-alignment ablation

On HybridQA and TAT-QA dev, the linker either rewrites the entity to a canonical cell value (REWRITE) or produces a partial alignment (PARTIAL: substring or numeric-format normalisation) on roughly 55% and 42% of linked entities, respectively (Appendix Figure 8). These are exactly the cases that pure NL2SQL would emit verbatim and miss (Table 6); toggling the linker off under No Router costs -0.85 pp F1 on first-attempt successes.

6.5 Failure modes

Three regimes dominate the residual error. **Router disagreements** with the LLM teacher sit mostly on SQL$\leftrightarrow$HYBRID borderline sub-queries where both primitives admit a plausible answer. **Verifier false-support**, the case in which $\nu_t{=}1$ on a sub-answer that is in fact wrong, contributes the larger share and is the immediate bottleneck on accuracy. **HYBRID hurts** when the sub-query is a narrative-attribute lookup with no DB column (no value to ground), which is the cost captured by the *No Router* ablation in Table 3.

7 Conclusion

We presented ATR, an iterative framework built around a sub-query-level router with failure-conditioned re-routing, and a HybridValueLinker that grounds every value referenced by structured execution against the cell index, addressing the large fraction of linked entity mentions on HybridQA and TAT-QA that pure NL2SQL emits verbatim and misses. A single ATR configuration matches or exceeds the strongest agentic prior across HybridQA, TAT-QA, and WTQ with no per-dataset tuning.

A distilled student router trained on HybridQA-train transfers to TAT-QA and WTQ without further supervision, removes the per-sub-query LLM call at inference, and outperforms a student trained on oracle routing labels, evidence that the teacher's labels carry verifier-aware route preferences absent from isolated correctness labels. Routing converts escalation from a frequent rescue path into a rare safety net at no accuracy penalty, and the architectural contribution transfers to self-hosted open-weight backbones. Sub-query-level routing, paired with failure-conditioned re-routing, is a productive replacement for fixed retrieval-or-SQL stacks in heterogeneous table-augmented QA. Two directions follow naturally: composing the router with explicit subgoal planning to extend beyond the current $K_{\max}{=}5$ horizon on long compositional chains, and broadening the primitive vocabulary to multimodal tables that include charts and images alongside text and structured cells.

Limitations

Headline numbers are single-seed point estimates on each benchmark's development set; we do not report multi-seed standard deviations or paired bootstrap significance tests, so ablation deltas under $\sim$5 pp F1 should not be over-interpreted, though the headline gaps to prior agentic baselines are large enough to dwarf typical seed-to-seed variance, and *Judge* (a Gemini Flash call at temperature 0.1) exhibits ± 0.5 pp run-to-run variance. The verifier is itself a prompted LLM (Gemini Flash in the headline configuration; Llama-70B in the open-weight variant) and is therefore subject to false-support and false-rejection that we do not measure against human judgement, so escalation accuracy is bounded by this unmeasured rate. The open-weight Llama backbone matches Flash on Wikipedia-style tables (HybridQA, WTQ) but TAT-QA's strict TagOp evaluator exposes a backbone-capability gap on financial arithmetic that the architecture alone does not close, and ATR caps the outer loop at $K_{\max}$=5 so questions requiring genuinely deeper compositional chains (5+ hops) likely need explicit subgoal planning rather than a larger $K_{\max}$. The scalar-fast-path that records single-scalar SQL outputs verbatim prevents paraphrase of precise numeric values but produces short outputs (mean 1.7 tokens) that lose on substring-tolerant metrics, which is the empirical source of the WTQ Relaxed and Judge gap to TableRAG-S. All benchmarks are English and cover Wikipedia, financial filings, and crowd-written compositional tables only; scientific, biomedical, legal, and government-data tables remain unverified. ATR's retrieval relies on the multilingual BGE-M3 embedder, so the question of cross-lingual transfer is empirical (untested benchmarks) rather than a constraint on the retrieval stack itself. The distilled router pays no LLM cost at inference, but training requires labels from the LLM-router teacher, and reproduction further requires the MySQL-backed SQL service of Yu et al. (2025) per dataset.

Ethics Statement

Use of Scientific Artifacts. We use PyTorch (Paszke et al., 2019), HuggingFace (Wolf et al., 2020), and vLLM (Kwon et al., 2023); open-weight backbones and BGE retrieval checkpoints are obtained through HuggingFace, and closed-source LLMs through Google Vertex AI and OpenAI under each provider's terms for academic research. HybridQA, TAT-QA, and WTQ are publicly released.

Use of AI Assistants. We only used Google Gemini to provide a better expression and to refine the wording. Some of the code used in the experiment was written with the assistance of Anthropic Claude. All AI-assisted outputs (writing and code) were reviewed and verified by the authors, who take full responsibility for the final content of this paper.

References

Ankush Agarwal, Chaitanya Devaguptapu, and Ganesh S. 2025. Hybrid Graphs for Table-and-Text based Question Answering using LLMs. In *Proceedings of NAACL-HLT 2025*, pages 858–875.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *ICLR 2024*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. 2020. HybridQA: A Dataset of Multi-Hop Question Answering over Tabular and Textual Data. In *Findings of EMNLP 2020*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. In *Transactions on Machine Learning Research*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In *Findings of ACL 2024*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. In *Transactions on Machine Learning Research*.

Si-An Chen, Lesly Miculicich, Julian Martin Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. TableRAG: Million-Token Table Understanding with Language Models. In *NeurIPS 2024*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding Language Models in Symbolic Languages. In *ICLR 2023*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *Findings of EMNLP 2020*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP 2023*.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *ICLR 2019*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2024. RouteLLM: Learning to Route LLMs with Preference Data. In *ICLR 2025*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS 2019*.

Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *ACL-IJCNLP 2015*.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *NeurIPS 2023*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of EMNLP 2023*.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the Text-to-SQL Capabilities of Large Language Models. *arXiv:2204.00498*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. In *NeurIPS EMC^2 Workshop*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *NeurIPS 2023*.

Ruoxi Sun, Sercan O. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL. *arXiv:2306.00739*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR 2023*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP 2020: System Demonstrations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR 2023*.

Xiaohan Yu, Pu Jian, and Chong Chen. 2025. TableRAG: A Retrieval Augmented Generation Framework for Heterogeneous Document Reasoning. In *EMNLP 2025*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. In *ACL-IJCNLP 2021*.

A Implementation Details

Backbone LLM. Gemini 2.5 Flash is served through Vertex AI (us-central1) for every prompted component (decomposer, LLM-router teacher, executors, verifier, value linker, final synthesis), with default temperature and top-p. Open-weight variants in Table 3 are served via self-hosted vLLM 0.6 with AWQ-INT4 quantisation on $2\times$ A6000 (tensor-parallel size 2), exposed through an OpenAI-compatible endpoint.

Retrieval stack. BGE-M3 (Chen et al., 2024) provides the dense 1024-dim embedding for Views 1 through 4; the BGE-reranker-v2-M3 cross-encoder reranks the top candidates per query. Document chunks are produced with a recursive character splitter at 512-token chunk size and 64-token overlap. The MySQL SQL service of Yu et al. (2025) is reused without modification.

ATR hyperparameters. Default values: $K_{\max}$=5, τ=0.1, cell-index budget B=10,000 per dataset, schema top-K=5, cell top-K_v=15, row top-K_r=10, escalation cap of three attempts per sub-query, worker-thread count = two. No per-dataset tuning beyond $K_{\max}$ and τ on HybridQA-train.

Router training. The DistilBERT router (66M parameters) is fine-tuned for five epochs with class-weighted cross-entropy on 9,998 HybridQA-train sub-query labels distilled from the LLM-router teacher. We use AdamW (lr 2×10^{-5}, batch size 32); class weights are inversely proportional to route frequencies in the training set. The teacher emits hard route labels (argmax), so the student is supervised on auto-labelled data rather than soft teacher logits. Training completes in under one hour on a single NVIDIA RTX 3080.

Reproducibility. All inference runs are single-seed and deterministic (seed 42, sampling temperature 0 except where noted). Random sub-sets (e.g., the TAT-QA dev sample used in the backbone-swap analysis) use the same seed. Code, training scripts, and prompt templates are released with the camera-ready.

Hardware and wallclock. Inference runs are I/O-bound against the Vertex AI / OpenAI / vLLM endpoints rather than compute-bound on the local machine; the client runs on a single CPU node with eight worker threads and two parallel sub-query workers. End-to-end wallclock per HybridQA dev pass is $\sim$30 minutes for Gemini 2.5 Flash, $\sim$2.5 hours for Claude Opus 4.7, and $\sim$2.5 hours for the self-hosted Qwen3.5-122B-A10B-GPTQ-Int4 / Llama-4-Scout-17B-16E-AWQ runs on $2\times$ A6000. The DistilBERT router adds $\sim$1 ms per sub-query on a single GPU and is therefore a negligible contribution to total wallclock. The MySQL SQL service of Yu et al. (2025) runs on the same node as the client; for self-hosted backbones the vLLM endpoint lives on a separate $2\times$ A6000 box reachable over a 10 Gb/s LAN link, and the per-call round-trip latency is dominated by LLM generation rather than network transport.

Data preprocessing. We follow the preprocessing of Yu et al. (2025) on every benchmark: HybridQA uses their canonical dev split (linked passages held fixed across all backbones; cell-grounded answer spans are not used at training or inference); TAT-QA uses the official dev split with the per-question scale annotations from the TagOp (Zhu et al., 2021) evaluator; WTQ uses the official Pasupat and Liang dev split with the standard `normalize_answer` function. Excel tables are persisted to MySQL by the offline pipeline of Yu et al. (2025), with the table name normalisation rules (`transfer_name` offline / `_normalize_table_key` online) kept in sync between the offline ingestor and the online SQL router.

SubQuery schema. Table 4 enumerates the six fields of the structured SubQuery object emitted by the Decomposer at each iteration t (§3.3). Each field is consumed by exactly one downstream module: the Router uses `expected_operator`, `required_modalities`, and `need_global_table_view`; the HybridValueLinker consumes `entity_mentions`; the stop controller reads `uncertainty`. The schema is therefore the load-bearing interface between the LLM-prompted decomposer and the rest of the pipeline, and constrains the decomposer to a small set of emit-time decisions that the next-stage modules can act on without re-prompting. A typical emitted object on HybridQA looks like `{sub_query: "What is the population of the city where the 2009 final was held?", expected_operator: lookup, required_modalities: both, need_global_table_view: false,`

`entity_mentions: {2009 final}, uncertainty: 0.6}`, which the router reads as "cell-grounded look-up over a table with a linked passage" and dispatches to HYBRID.

B Online Inference Loop

Algorithm 1 gives the line-by-line procedure of the online loop described in §3.1. The pseudocode is a page-wide float because its lines do not fit a single column without breaks; section references in the algorithm point back to the corresponding method subsections.

The loop has two nested iterations. The *outer* loop iterates over decomposed sub-queries (bounded by $K_{\max}$=5); the *inner* loop iterates over routing attempts for a single sub-query (bounded by M=3 escalation attempts). Two early-exit paths short-circuit the outer loop: (i) the decomposer emits the sentinel TERMINATE when it judges the question already answered, and (ii) the stop controller exits when the verifier returns ν_t=1 and the residual uncertainty u_t falls below τ=0.1. Together these two exits save roughly a third of the maximum iteration budget on HybridQA dev: the mean realised iteration count is 3.4 (vs. 5 cap), and only 11% of questions hit the cap.

Inside the inner loop, the failed-route history $H_{<t}$ enters the router's input, so each escalation is conditioned on which primitives have already been rejected, a discrete form of counterfactual routing rather than a free-form LLM patch. The default escalation chain learned by the teacher is SQL → HYBRID → RETRIEVE for operator-heavy sub-queries and RETRIEVE → TEXT for lookup/filter sub-queries; the student internalises both chains. If Final Synthesis is disabled (toggled off in the − Final Synthesis ablation), the loop returns the most recent verifier-accepted answer y_t directly, which works for single-hop questions but loses the multi-hop aggregation that drives the −5.3 pp F1 in Table 3.

C Additional Ablations

This section reports the ablations referenced from §6 that did not fit into the body. We organise them around four questions: how routing trades latency for accuracy, where the router's value actually comes from, how iteration interacts with student/teacher agreement, and how the four offline views overlap.

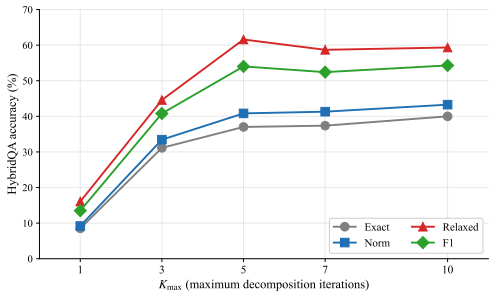

Figure 5: $K_{\max}$ saturation on HybridQA dev: all metrics saturate between $K_{\max}$=5 and $K_{\max}$=7.

Iteration and retrieval views. Figure 5 reports the $K_{\max}$ sweep on HybridQA dev: Relaxed EM climbs from 16.1 at $K_{\max}$=1 to 61.6 at $K_{\max}$=5 and slightly regresses at $K_{\max}$=7 (58.7). The early steep slope ($K_{\max}$=1→3: +28 pp) reflects multi-hop questions that need at least two to three sub-queries to resolve; the diminishing return between 3 and 5 tracks questions whose remaining hops are answered by Final Synthesis rather than further decomposition. The regression at $K_{\max}$=7 is a dilution effect: the decomposer keeps emitting sub-queries past the natural compositional depth of the question, and their loosely related sub-answers push Final Synthesis toward off-topic paraphrases. We therefore use $K_{\max}$=5 as the default. Coupling this with the stop controller (§3.6) keeps the mean realised iteration count to 3.4 even at the 5-iteration cap.

Figure 6 projects the four offline view embeddings (§3.2) into a shared BGE-M3 space via t-SNE; clusters by view are largely disjoint, confirming that the views encode complementary semantic signals rather than redundant copies. Gold-table coverage at K=10 is uneven across views (row 69%, schema 42%, chunks 21%, cells 6%): row retrieval is the highest-recall single view, but no view covers everything, and the cell view is the only one that surfaces the precise `(column, value)` target needed for value linking, even though its standalone gold-table recall is the lowest. This is exactly the asymmetric-utility pattern that a route-conditional consumer can exploit: HYBRID needs cells for grounding, SQL needs schema, TEXT needs chunks, and RETRIEVE needs rows. A single fused index would either over-weight the high-recall but low-precision row view (collapsing schema and cell signal) or pay the cost of maintaining four indices anyway; we therefore keep them separate and let the router pick which to consult.

Table 4: The SUBQUERY object emitted by the DECOMPOSER.

Field	Type	Range / Domain	Role
`sub_query`	string	free text	natural-language sub-question
`expected_operator`	enum	lookup, filter, count, compare, aggregate, sort, arithmetic	coarse reasoning primitive; routing feature
`required_modalities`	enum	table, text, both	evidence modality; routing feature
`need_global_table_view`	bool	true, false	full-table aggregation vs. local lookup
`entity_mentions` E_t	set	$\{e_1, \ldots, e_n\}$	spans grounded by the value linker
`uncertainty` u_t	real	$[0, 1]$	residual fraction of q unanswered after q_t; stop-controller signal

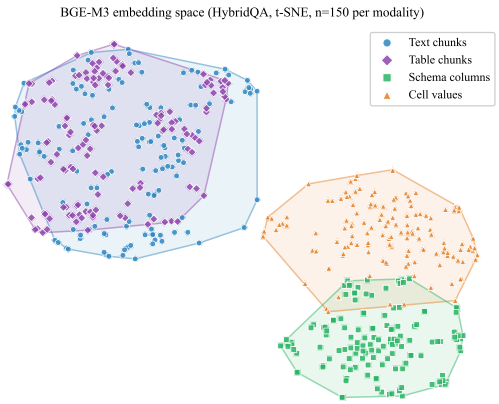

Figure 6: t-SNE projection of the four offline retrieval views (text chunks, table chunks, schema columns, cell values) on a shared BGE-M3 space (HybridQA).

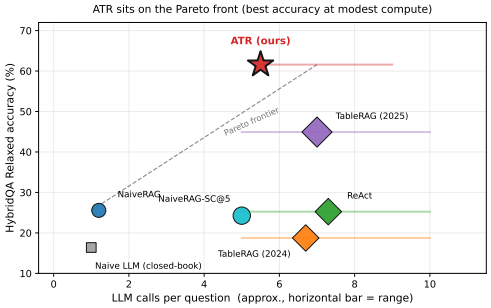

Figure 7: ATR sits on the Pareto front: top accuracy at modest LLM-call cost.

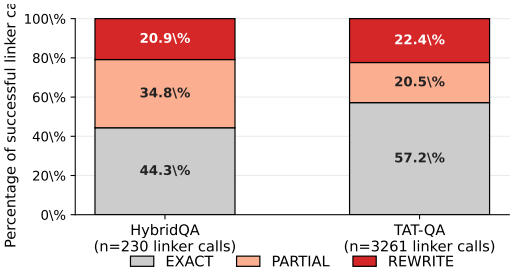

Figure 8: HYBRIDVALUELINKER output distribution. EXACT: entity matches a cell verbatim; PARTIAL: substring or format normalisation; REWRITE: lexical translation (e.g. `'second'`→`'2nd'`). REWRITE+PARTIAL cover 42 to 56% of linker calls on HybridQA and TAT-QA, the mismatches naive NL2SQL would miss.

Design-space comparison. Table 5 maps each shared-protocol baseline against seven load-bearing axes (cell index, NL2SQL, multi-route, verifier + escalation, iterative decomposition, self-consistency, graph traversal). ATR is the only system that combines all five non-graph axes; the dataset-specific ODYSSEY adds graph traversal at the cost of every other axis. Figure 7 translates the same comparison into the (mean LLM calls per question, HybridQA accuracy) plane: ATR sits on the Pareto front with highest accuracy at a moderate call budget; TableRAG-S trades higher calls for similar Relaxed accuracy but worse F1; NaiveRAG and ReAct sit below ATR on accuracy at comparable call cost.

Value-linker telemetry. On the HybridQA dev set, the HYBRIDVALUELINKER classifies its successful outputs as EXACT (44.3%, the entity already matches a cell verbatim), PARTIAL (34.8%, substring or numeric-format overlap), or REWRITE (20.9%, canonical swap). On TAT-QA dev the corresponding fractions are 57.2%/20.5%/22.4%. The REWRITE+PARTIAL share, 42 to 56% across the two benchmarks, quantifies the slice of entities that pure NL2SQL would emit verbatim and miss; toggling the linker off under No Router costs −0.85 pp F1 on first-attempt successes (51.37→50.52), and the gap widens further once escalations are counted because the linker also gates the fallback to TEXT when no candidate survives verification. Figure 8 visualises the same per-class distribution.

Table 6 samples representative alignments. REWRITE covers canonical-name swaps such as the ordinal `second`→`2nd` or the abbreviation

Table 5: Design-space comparison. ●/◐/○ = has / partial / absent. ATR combines all five load-bearing axes.

System	Cell index	NL2SQL	Multi-route	Verifier + Esc.	Iter. decomp.	Self-cons.	Graph traversal
Naive LLM	○	○	○	○	○	○	○
NaiveRAG	○	○	○	○	○	○	○
NaiveRAG-SC$_{@5}$	○	○	○	◐	○	●	○
ReAct ($\leq$10 steps)	○	◐	◐	◐	◐	○	○
TableRAG-R	◐	○	○	◐	●	◐	○
TableRAG-S	○	●	○	◐	●	○	○
ODYSSEY	○	○	◐	○	◐	○	●
ATR (ours)	●	●	●	●	●	○	○

`GP` → `Grand Prix` — cases where the question's lexical form and the cell's storage form are linguistically equivalent but lexically disjoint. PARTIAL covers substring and numeric-format normalisations such as `2011-05-17` → `2011` (year extraction) or `Lindsay` → `Lindsay Gilbee` (first-name expansion), all cases where the question token is a sub-token of the cell or vice versa. TAT-QA's financial filings add a third flavour, `2019` → `19`, where the cell stores a short-form year header that the question uses in long form.

Table 6: Representative HYBRIDVALUELINKER alignments. REWRITE: canonical-name swap. PARTIAL: substring or numeric-format normalization.

Class	Question entity	Cell value
HybridQA		
REWRITE	`GP`	`Grand Prix`
REWRITE	`fifth season`	`Season 5`
REWRITE	`midfielder`	`MF`
REWRITE	`St. Louis`	`Saint Louis`
PARTIAL	`2011-05-17`	`2011`
PARTIAL	`Lindsay`	`Lindsay Gilbee`
PARTIAL	`NY`	`NY Yankees`
PARTIAL	`1.83 m`	`1.83`
TAT-QA		
REWRITE	`Solutions seg.`	`Industrial Sol.`
REWRITE	`net revenues`	`Net sales`
REWRITE	`domestic disc.`	`4.00%`
PARTIAL	`2019`	`19`
PARTIAL	`$1,234 mil.`	`1234`
PARTIAL	`Q4 2019`	`2019-Q4`

TAT-QA per-`answer_type` breakdown. Table 2 decomposes the TAT-QA dev results of Table 1 along TAT-QA's `answer_type` label (`arithmetic`, `count`, `multi-span`, `span`). ATR leads NaiveRAG by $+5.43$ pp F1 on `arithmetic` (n=718, the largest type) and $+6.25$ pp on `count` (n=32) — the two types that require explicit multi-step computation over table values. On `multi-span` (n=217) the two systems are within ± 2 pp F1. On `span` (n=701, 42% of dev) NaiveRAG leads by 4.01 pp F1: these questions have their answer present verbatim in the retrieved passage, so a retrieve-and-extract pipeline with a strong backbone (Gemini 2.5 Flash) already saturates the metric; ATR's iterative decomposition adds little here and occasionally over-paraphrases the span, which TAT-QA's strict TagOp scorer punishes. The aggregate F1 gap ($+0.55$ pp) is therefore a weighted average dominated by the `span` bucket; the architectural contribution lives in the computation-heavy buckets where ATR's value-linking and constrained SQL execution actually matter.

Full backbone catalogue. Table 7 reports the full backbone swap on HybridQA dev across nine backbones, split into closed-weight (Gemini 2.5 Flash anchor, Gemini 2.5 Pro, Gemini 3.1 Pro Preview, Claude Haiku 4.5, Claude Opus 4.7, GPT-4o-mini) and open-weight self-hosted via vLLM (Llama-3.3-70B, Qwen3.5-122B, Llama-4 Scout). Three patterns emerge.

Within-family variation is small at competitive scale. Within the Gemini family, Pro is within $+0.56$ pp F1 of the Flash anchor, while Gemini 3.1 Pro Preview adds another $+9.34$ pp on top of Pro, so almost all of the family's spread comes from the frontier release, not from the Pro/Flash split at the previous generation. Within Claude, Opus 4.7 leads Haiku 4.5 by $+12.02$ pp F1; the gap is wider than the Gemini family because Haiku is positioned as a small/efficient model rather than a frontier sibling. Within Llama, Llama-3.3-70B (dense) leads Llama-4-Scout (MoE, 17B active)

by $+2.89$ pp F1 on this workload despite Scout's $4\times$ larger total parameter count: structured-JSON agentic prompts favour dense compute over sparse MoE at comparable active scale (§6.2).

Cross-family ordering tracks active compute, not total parameters. The top of the table (3.1 Pro Preview, Opus 4.7, Llama-3.3-70B) and the bottom (GPT-4o-mini, Haiku 4.5) split along the active-parameter axis: frontier closed models and dense 70B open-weights win, while small reasoning-tuned checkpoints (GPT-4o-mini's 8B-class budget, Haiku 4.5's small-tier positioning) lose by -4 to -9 pp F1. Total parameters in MoE models (Llama-4-Scout's 109B, Qwen3.5's 122B) do not close this gap because only $\sim$10 to 17 B are active per token on a structured JSON workload.

Open-weight ATR is competitive with closed-weight Flash. Llama-3.3-70B-AWQ self-hosted via vLLM ($+0.66$ pp F1 vs. Flash) and Qwen3.5-122B-A10B-GPTQ-Int4 (-1.26 pp F1) bracket the Flash headline within ± 1.3 pp at zero API cost. This is the empirical claim behind the floor/ceiling reading: ATR's architectural floor (the persistent gap to baselines under *any* backbone choice) does not require closed-weight LLM access on Wikipedia-style tables.

Router error modes. Beyond the global agreement rate (98.6% between DistilBERT student and LLM teacher on HybridQA dev), the residual disagreements concentrate at three boundaries. SQL$\leftrightarrow$HYBRID disagreements ($\sim$0.8% of sub-queries) occur on row filters with at most one entity, where both primitives are plausible and the teacher's preference is sub-tle (whether the cell needs linking or already matches). TEXT$\leftrightarrow$RETRIEVE disagreements ($\sim$0.4%) occur on biographical look-ups whose target attribute might or might not have a dedicated cell. HYBRID$\rightarrow$TEXT false negatives ($\sim$0.2%) are the most consequential failure: the student under-routes to HYBRID when an entity *would* have been linkable but the student's confidence is too low. Re-routing on failure absorbs the first two classes: $V{=}0.5$ on the wrong first choice triggers a re-route within the same outer iteration, and the third class is the main residual error in Table 3's bottom block: a route the student declined but the teacher would have taken.

Cross-dataset transfer. The DistilBERT student is trained only on HybridQA-train and deployed zero-shot on TAT-QA and WTQ. The transfer holds because the four operator $\times$ modality $\times$ entity-mention slots are dataset-agnostic. TAT-QA arithmetic questions surface as `expected_operator` $\in$ {count, compare, aggregate, arithmetic} with `required_modalities` $=$ table; the student dispatches these to SQL/HYBRID without having seen TAT-QA training data. WTQ filter/sort questions surface as `filter`/`sort` with `required_modalities` $=$ table, dispatched to RETRIEVE/HYBRID depending on entity-mention presence. The Decomposer's emit-time decision to fill these fields is what makes the cross-dataset jump possible; an operator-blind question encoder would have to learn the routing boundary separately on each benchmark.

Per-call inference load and dollar cost. Table 8 reports the LLM-call profile for one HybridQA dev run across every backbone in Table 7 for which we logged HTTP call counts. The Full ATR Gemini 2.5 Flash configuration issues 4,505 LLM calls (14.77 per question on average) for a total of 11.26M input and 0.90M output tokens. Backbone choice changes the call budget by less than $1.3\times$ (min 3,442 for Gemini 3.1 Pro Preview, max 4,505 for Gemini 2.5 Flash), and the per-call prompt size is dominated by retrieval context rather than backbone-specific tokenisation.

The rightmost Cost column in Table 8 applies each provider's listed per-token rates (May 2026) to the measured token counts and reports the USD spend per HybridQA dev pass. The Flash headline costs \$1.11, GPT-4o-mini \$1.80, while Claude Opus 4.7 reaches \$188.40, roughly $170\times$ the Flash baseline, driven by output-token pricing ($\$75/M$ vs. $\$0.60/M$ for Flash) compounded across ATR's 5-iteration decomposition. Self-hosted open-weight backbones (Llama-3.3, Qwen3.5, Llama-4 Scout) carry no per-call API charge. The sub-quadratic growth in calls with $K_{\max}$ is what makes a DistilBERT student attractive: a per-sub-query LLM router would add one extra LLM call per iteration, inflating the call count by $\approx K_{\max}$, while the distilled student adds zero.

Cost vs. accuracy interplay. Placing the per-run dollar cost alongside the headline F1 from Table 7 reveals two distinct cost-accuracy regimes. In the proprietary tier, frontier closed models trade a one to two orders of magnitude cost increase for $+8$ to $+10$ pp F1 over the Flash anchor: Gemini 3.1 Pro Preview at $16\times$ cost buys $+9.9$ pp, Claude Opus 4.7 at $170\times$ cost buys $+8.2$ pp. The Opus premium

Table 7: Full backbone swap on HybridQA dev (learned router fixed). Top: closed-weight; bottom: open-weight (self-hosted via vLLM). ΔF1 vs. the Gemini 2.5 Flash anchor.

Variant	Exact	Norm	Relax	**F1**	Judge	ΔF1
Closed-weight						
Gemini 2.5 Flash (anchor)	37.02	40.83	61.59	54.02	54.02	—
Gemini 2.5 Pro	38.36	42.30	63.28	54.58	54.10	+0.56
Gemini 3.1 Pro Preview	**47.87**	**53.11**	**69.51**	**63.92**	**64.92**	+9.90
Claude Haiku 4.5	35.74	39.67	58.69	50.15	54.28	−3.87
Claude Opus 4.7	47.21	51.15	65.90	62.17	62.62	+8.15
GPT-4o-mini	28.20	35.08	47.87	45.03	46.23	−8.99
Open-weight						
Llama-3.3-70B-AWQ	39.67	44.92	56.39	54.68	55.08	+0.66
Qwen3.5-122B-A10B-GPTQ-Int4	40.33	42.62	57.05	52.76	51.15	−1.26
Llama-4-Scout-17B-16E-AWQ	38.36	43.28	53.77	51.79	53.29	−2.23

Table 8: Per-call inference load and per-run USD cost on HybridQA dev (tokens assume 2,500 in / 200 out per call; May 2026 list rates). **Bold** = Flash anchor; open-source rows are self-hosted (no API charge).

Type	**Model**	**Ctx**	**Cutoff**	**Checkpoint / API ID**	**Calls**	**C/Q**	**T_{in}(M)**	**T_{out}(M)**	**Cost ($)**
Proprietary	**Gemini 2.5 Flash**	1M	Jan 2025	gemini-2.5-flash	**4,505**	**14.77**	**11.26**	**0.90**	**1.11**
	Gemini 2.5 Pro	1M	Jan 2025	gemini-2.5-pro	3,818	12.52	9.55	0.76	19.54
	Gemini 3.1 Pro Preview	1M	Jan 2025	gemini-3.1-pro-preview	3,442	11.29	8.61	0.69	17.66
	GPT-4o-mini	128K	Oct 2023	gpt-4o-mini-2024-07-18	3,632	11.91	9.08	0.73	1.80
	Claude Haiku 4.5	200K	Feb 2025	claude-haiku-4-5	4,438	14.55	11.10	0.89	15.55
	Claude Opus 4.7	200K	Jan 2026	claude-opus-4-7	3,582	11.74	8.96	0.72	188.40
Open-Source	Llama-3.3-70B-AWQ	128K	Dec 2023	llama-3.3-70b-instruct-awq	3,736	12.25	9.34	0.75	0.00
	Qwen3.5-122B-A10B-GPTQ-Int4	262K	May 2025	Qwen3.5-122B-A10B-GPTQ-Int4	3,687	12.09	9.22	0.74	0.00
	Llama-4-Scout-17B-16E-AWQ	10M	Aug 2024	Llama-4-Scout-17B-w4a16	4,483	14.70	11.21	0.90	0.00

is particularly steep because output-token pricing dominates ($\$75/M$ output vs. $\$0.60/M$ for Flash) and ATR's intermediate sub-answer chains rack up output tokens across 5 iterations of decomposition, routing, and verification. In the open-weight tier, Llama-3.3-70B-AWQ delivers $+0.66$ pp F1 over Flash at \$0 marginal API cost, which positions the self-hosted variant as a strong default for cost-sensitive deployments. Qwen3.5 sits within -1.3 pp F1 at the same zero-API price and adds $10\times$ longer context support (262K tokens), useful when the routing pipeline must carry long retrieval contexts through multiple sub-queries.

Practical guidance for backbone selection. The four (cost, accuracy) quadrants suggest a default policy. For tight latency and the simplest deployment, Gemini 2.5 Flash sits at the cost-accuracy Pareto frontier among proprietary backbones. For maximum accuracy regardless of cost, Gemini 3.1 Pro Preview's $+9.9$ pp F1 jump is the largest single-knob return in our matrix. For privacy- or sovereignty-sensitive deployments, Llama-3.3-70B-AWQ on $2\times$ A6000 matches the Flash headline at zero per-call dollar cost while keeping inputs and outputs on-premise. Claude Opus 4.7 is the only backbone where the cost premium is not justified by the accuracy delta on this workload; it is worth using when its 200K context window or its non-Google provider story matters, but not as a default for HybridQA-style tasks.

D End-to-End Walkthrough

Figure 9 traces ATR on the HybridQA two-hop question *"What is the nationality of the driver who finished second at the 2001 Brazilian Grand Prix?"*, whose gold answer is *German*. The question requires (i) locating the runner-up row in a Wikipedia results table and (ii) joining the driver name to a biographical attribute that lives only in a linked passage, a canonical multi-hop, cross-modal trajectory that breaks both pure NL2SQL and pure retrieval pipelines.

Iteration 1: cell-grounded SQL. The decomposer emits the sub-query q_1=*"Who finished second at the 2001 Brazilian Grand Prix?"* with operator `filter`, modality `table`, and entity mentions E_1={*second*, *2001 BGP*}. The router selects HYBRID because q_1 asks for a row filter over a structured table and the entity *"second"* would never match the cell verbatim. The HYBRID-VALUELINKER rewrites *"second"* to the canonical cell value `'2nd'` (a REWRITE: confidence 0.60 on

the `Position` column) and partially aligns *"2001 BGP"* to `'2001'` on the `Year` column. CONSTRAINEDSQL then issues `SELECT Driver FROM 2001_Brazilian_Grand_Prix_0 WHERE Position = '2nd'`, restricted to columns retrieved from View 3 and seeded with the linked value. This returns *Ralf Schumacher*. The verifier accepts with ν_1=1. A pure NL2SQL baseline that emits `Position = 'second'` verbatim would have returned an empty row set; the value linker is what turns the table lookup into a hit.

Iteration 2: text fallback. The decomposer now emits q_2=*"What is the nationality of Ralf Schumacher?"* with operator `lookup`, modality `text`, and E_2={*Ralf Schumacher*}. The router selects TEXT because *nationality* is a biographical attribute with no corresponding DB column. The linked Wikipedia passage retrieved from View 1 supplies *"a German former racing driver"*, and the verifier confirms ν_2=1. With both sub-queries verified and residual uncertainty $u_2<\tau$, the stop controller exits the loop at t=2; Final Synthesis aggregates the two sub-answers into the final response *German*, matching the gold answer exactly.

This trace illustrates the operational signature of sub-query routing: a single question mixes HYBRID (table-with-text cell-grounded SQL) and TEXT (passage retrieval) within its two iterations, and verifier-confirmed sub-answers carry over to the final response without an extra LLM-fusion call.

Failure modes of single-primitive baselines. The same question highlights why a fixed-primitive baseline loses on either iteration. A pure NL2SQL system would issue `WHERE Position = 'second'` for iteration 1, miss the row, and have no fallback for the *nationality* attribute in iteration 2 because the table simply does not have that column. A pure retrieval system (chunks + reranker) would surface the linked Wikipedia passage that mentions *Ralf Schumacher* and his nationality, but lose the precise row-filter "second" that is only computable against the structured table; it would either invent a runner-up not present in the chunked text or return the wrong driver. The escalation chain makes ATR's failure recoverable in both directions: if iteration 1 had returned *not found*, the router would have re-routed to RETRIEVE+row index on the second attempt; if iteration 2 had returned a low-confidence answer, escalation would have triggered a HYBRID retry with the runner-up as an anchor entity. Single-primitive systems pay this cost as a hard failure rather than a soft re-route.

Connection to value-linker statistics. This walkthrough's REWRITE (*second* → `'2nd'`) and PARTIAL (*2001 BGP* → `'2001'`) alignments are exactly the two majority classes on HybridQA dev (20.9% + 34.8%; §6.4). The cells the embedding ranks first are correct verbatim in 43% of cases (EXACT); the remaining 56% need an LLM-verified rewrite before the SQL executor accepts the row.

Algorithm 1 ATR online inference (§3.1): decompose, route, verify; re-route on $\nu_t < 1$.

Require: Question q; multiview indices (Views 1 to 5; View 4 hosts both a cell index and a row index, §3.2); max iterations $K_{\max}$; max escalations per sub-query M; verifier threshold τ

Ensure: Final answer a

```
z_{<1} ← ∅                                                    ▷ sub-query / answer history
for t = 1 to K_max do
    (q_t, meta_t, E_t, u_t) ← DECOMPOSE(q, z_{<t})
    if q_t = TERMINATE then
        break
    end if
    chunks ← DOCUMENTRETRIEVAL(q_t)
    schema ← RESTORESCHEMAVIAMAPPING(chunks)
    H_{<t} ← ∅
    for k = 1 to M do
        r ← ROUTER(q_t, meta_t, schema, H_{<t})
        y_t ← EXECUTE(r, q_t, schema, E_t, chunks)
        ν_t ← VERIFIER(q_t, y_t)
        if ν_t ≥ 1 then
            break
        end if
        H_{<t} ← H_{<t} ∪ {r}
    end for
    z_{≤t} ← z_{<t} ∪ {(q_t, y_t)}
    if ν_t = 1 and u_t < τ then
        break                                                 ▷ stop controller
    end if
end for
if FINALSYNTHESIS enabled then
    a ← FINALSYNTHESIS(q, z_{≤t})
else
    a ← y_{last verifier-accepted}
end if
return a
```

Iteration 1 | V_1=1 ✓

Decomposer: q_1 = "Who finished second at the 2001 Brazilian Grand Prix?"

`op=filter` `mod=table` `ents=[second, 2001 BGP]` ⟶ **HYBRID**

Why: structured row-filter with lexical mismatch on "second" (cells store ordinals as `'2nd'`).

ValueLinker: `ents=[second]` $\xrightarrow{\text{REW}}$ `'2nd'` (`Position`, 0.60)

`ents=[2001 BGP]` $\xrightarrow{\text{PAR}}$ `'2001'` (`Year`, 0.80)

⇒ V^*={(`Position`,`'2nd'`)}.

Generated SQL (CONSTRAINEDSQL)

```
SELECT Driver FROM 2001_Brazilian_Grand_Prix_0 WHERE Position = '2nd';
```

Verifier: accepts. **Sub-answer:** *Ralf Schumacher*.

Iteration 2 | V_2=1 ✓

Decomposer: q_2 = "What is the nationality of Ralf Schumacher?"

`op=lookup` `mod=text` `ents=[Ralf Schumacher]` ⟶ **TEXT**

Why: nationality is biographical, no DB column; linked passage is the only source.

Top retrieved passage (View 1)

"Ralf Schumacher (born 30 June 1975) is a German former racing driver…"

Verifier: accepts. **Sub-answer:** *German*.

Stop: V_2=1 ∧ u_2<τ exits at t=2; **Final Synthesis** aggregates {*Ralf Schumacher*, *German*} into the final answer *German*.

Figure 9: End-to-end walkthrough of a HybridQA two-hop question (gold: *German*). Iteration 1 routes to HYBRID (lexical mismatch on a table-row filter); iteration 2 routes to TEXT (attribute lives only in a linked passage).

Prompt 1: Decomposer

Decompose the question into atomic sub-queries, each answerable by a single retrieval step.

Input: {question}, {history}.

If history suffices → sub_query=TERMINATE. Otherwise emit one new sub-query.

Output JSON:
sub_query (string),
required_modalities ∈ {text, table, both},
expected_operator ∈ {lookup, filter, count, compare, aggregate, sort, arithmetic},
need_global_table_view (bool),
entity_mentions (set of spans),
uncertainty ∈ $[0, 1]$.

Strategy hints:
BRIDGE: find X, then look up Y.
MULTI-HOP LOOKUPS: chain across rows / passages.
COMPARATIVES: each comparand separately.
NEVER repeat an already-answered sub-query.

(a) Decomposer (§3.3): emits the next sub-query q_t as a structured SUBQUERY object.

Prompt 2: LLM Router (teacher)

Select the single best execution route for the sub-query.

Input features:
sub_query,
expected_operator,
required_modalities,
entity_mentions,
schema_available (bool),
already_failed_routes (set).

Routes: TEXT / RETRIEVE / SQL / HYBRID.

Selection rules:
R1: modality=text or no schema → TEXT.
R2: aggregation + schema + no entities → SQL.
R3: aggregation + entities → HYBRID.
R4: lookup/filter + entities → RETRIEVE.
R5: lookup/filter, no entities → TEXT.
R6: any route in already_failed_routes is excluded.

Output one token from {TEXT, RETRIEVE, SQL, HYBRID}.

(b) LLM Router (§3.4): teacher for the DistilBERT student distillation.

Prompt 3: Verifier

Judge whether the candidate answer is faithfully supported by the evidence.

Input: question, candidate answer, retrieval evidence, SQL result (if any).

Three verification dimensions:
D1 *slot alignment*: answer type matches the question slot (name / number / date / count).
D2 *evidence-answer consistency*: every claim in the answer is derivable from the retrieval evidence or the SQL result.
D3 *route faithfulness*: every entity name and cell value in the answer appears verbatim in some evidence source (no hallucinated spans).

Verdict:
V=1: all three pass.
V=0.5: partial support, or one minor failure (also when SQL returns empty but evidence is consistent).
V=0: answer contradicts evidence or names an unseen entity.

(c) Verifier (§3.6): returns $V(r) \in \{0, 0.5, 1\}$ for the escalation chain.

Prompt 4: LLM Judge (evaluation only)

Rate whether the predicted answer is suitable for the question against the gold answer.

Criteria:
Accuracy / Hallucinations: the prediction must be semantically consistent with the gold; numerical values and ordering must match; no fabricated entities or facts.
Completeness: the prediction must contain all key points required by the question; further elaboration may be omitted.

Overall score on a $\{0, 1\}$ scale (0=wrong, 1=correct). Output a single line.

Format: Rating: [[score]].

Input fields: {question}, {golden}, {gen}.

(d) LLM Judge: evaluation-only prompt for the supplementary *Judge* metric in Tables 1 and 3; retained verbatim from Yu et al. (2025) so the metric is strictly comparable.

Figure 10: ATR prompt templates (abridged). Prompts 1–3 drive the inference loop; Prompt 4 (LLM Judge) is used only for the supplementary *Judge* metric.