TabRL-Summarizer: Two-Stage Text-to-Table Generation with MONA Multi-Policy Reinforcement Learning

Arya Patil Thanishka Bolisetty Simran Sharma
Hriday Pradhan Mohammed Usman Mehboob
Arizona State University, Tempe, USA
{apatil45, tboliset, sshar465, hpradha3, musmanme}@asu.edu

Abstract

Structured tables are the standard interface for analysis and decision making, yet most information is written as unstructured text. We study text-to-table generation in which a system must infer an appropriate schema and then populate a faithful table instance from a document. We present **TabRL-Summarizer**, a two-stage architecture that first predicts a schema and then generates a table conditioned on that schema. Both stages are instantiated with LLaMA 3.1 8B Instruct and trained by supervised fine tuning on text-to-table pairs. We evaluate this supervised backbone on the LiveSum and Rotowire benchmarks and show that it achieves strong team-level completeness on Rotowire but struggles with fine-grained player statistics, and that naively adding multi-step Text-Tuple-Table structure can harm LiveSum performance when it is not coupled with suitable rewards. Motivated by these observations, we design a *MONA* (Multi-Objective Non-Aggregating) reinforcement learning architecture in which schema and table policies are optimized with separate rewards and no cross-stage credit. The present work contributes the supervised backbone, evaluation protocol, and a detailed MONA-based reinforcement learning blueprint for future optimization of text-to-table systems.

1 Introduction

Relational tables are the workhorse representation for databases, business intelligence, and many downstream machine learning systems. In contrast, humans naturally produce long, unstructured narratives such as sports recaps, financial reports, and scientific articles. Automatically converting these documents into high quality tables would enable large-scale analytics and monitoring, but it requires reliable schema design and cell-level extraction.

The system must decide which attributes to include, how to group them into rows and columns, and which types and units to use. It must then populate each cell with values that are supported by the document and respect the chosen schema. Realistic applications also require strict output formats such as valid JSON or CSV, fixed column order, and consistent treatment of missing values.

Large language models (LLMs) are a natural fit for this task because they can read long documents and generate structured text. However, purely supervised learning on text-to-table pairs does not directly optimize table quality metrics, does not penalize subtle schema mistakes, and does not enforce formatting constraints reliably. Multi-step pipelines, such as Text-Tuple-Table (T3), introduce additional points of failure when they are used only through prompting.

This paper has two goals. First, we build and evaluate **TabRL-Summarizer**, a two-stage supervised backbone that explicitly separates schema prediction from table generation. Second, we design a reinforcement learning extension based on the MONA principle, where the schema policy and the table policy are treated as separate agents with their own reward functions and without cross-stage credit.

Our supervised experiments on LiveSum and Rotowire highlight both the strengths and weaknesses of the current backbone. On Rotowire, TabRL-Summarizer achieves the highest completeness for team-level statistics but fails on player tables, where row alignment is more challenging. On LiveSum, a naive T3 prompting strategy hurts a strong LLaMA baseline but slightly improves our table-aware backbone.

These results motivate the MONA reinforcement learning architecture that we outline later in the paper. In that design, schema and table policies are optimized separately using group relative preference optimization with KL regularization, and table rewards are computed only on tables that are generated from sufficiently good schemas.

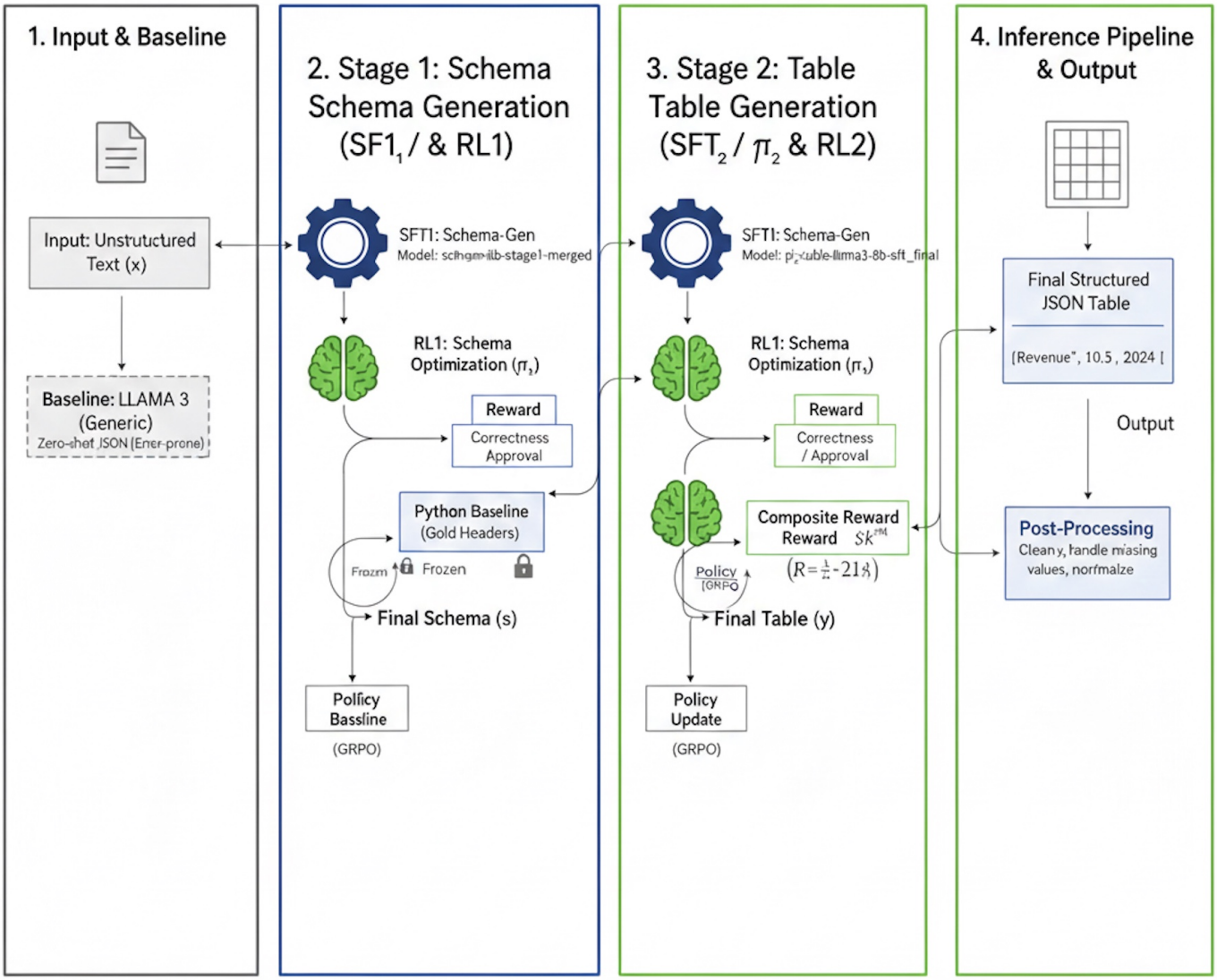

Figure 1: End-to-end TabRL-Summarizer pipeline. Stage 1 predicts a schema from unstructured text. Stage 2 generates a table conditioned on the predicted schema. Both stages are trained with supervised fine tuning. In future work the same stages will be refined with MONA-style reinforcement learning.

2 Task and Data

2.1 Problem formulation

Given an input document x describing an entity or an event, the goal is to produce a table y that satisfies three properties:

1. It conforms to a well-defined schema s with explicit column names and types.
2. Each cell value is supported by the document and does not hallucinate information.
3. The representation is machine readable and compatible with downstream tools and evaluators.

Supervision is provided as text-to-table pairs (x, y); the schema is not given separately. The system must therefore infer a schema from the gold tables during training and from the document at inference time.

2.2 Training corpora

We construct a unified training corpus from several datasets that contain aligned text and tables:

- **E2E** contains restaurant descriptions and attribute tables.
- **Rotowire** [10] contains basketball game summaries paired with box-score tables for teams and players.
- **WikiBio** contains short Wikipedia biographies with infobox tables.

Each dataset is converted into a JSONL format in which each line stores a narrative field `text` and a table field `table`. The combined training file is referred to as `training_data.jsonl`. During schema training we

extract the header of each gold table to construct the target schema.

2.3 Evaluation benchmarks

We evaluate on two held-out benchmarks that stress different aspects of text-to-table generation.

LiveSum. LiveSum [11] focuses on soccer commentary. Each input is a full match commentary, and the target output is a 2×9 statistics table that includes goals, cards, shots, and other counts. The benchmark partitions the cells into Easy, Medium, and Hard subsets and reports mean absolute error (MAE), root mean squared error (RMSE), and exact match error (EM) at both split and macro levels.

Rotowire. For Rotowire we follow the setting of Wiseman2017. We evaluate both team tables and player tables. Team tables summarize game-level statistics for each team, while player tables include per-player rows with points, rebounds, minutes, and other statistics. We use a string similarity metric for team tables and the TabEval metric [16] for both team and player tables.

3 Two-Stage Supervised Architecture

3.1 Base model and data encoding

Both stages use LLaMA 3.1 8B Instruct as the base model. Training examples are represented as chat-style sequences of messages with roles `system`, `user`, and `assistant`. If dataset metadata is available (for example, a brief description of the dataset or domain), it is inserted as a synthetic system message before the user content.

We use the official chat template to serialize messages and train with a causal language modeling objective. Sequences are truncated from the left to respect a maximum context length of 8,192 tokens. Left truncation keeps the most recent parts of long documents, which often contain the highest density of statistics.

3.2 Stage 1: Schema policy π_1

Stage 1 maps input text x to a schema s. We represent the schema as a JSON array of field objects with `name` and `type` keys. For each example we construct:

- a system message that instructs the model to read the document and output a schema that is sufficient to represent its content;
- a user message that contains a short natural-language policy, optional dataset metadata, the raw document x, and a description of the output format;
- an assistant message that contains the gold schema extracted from the header of the gold table y.

The schema policy is trained by supervised fine tuning to minimize negative log likelihood on the assistant message. The resulting schema model is available at `https://huggingface.co/mohdusman001/schema-gen-llama3-8b-stage1-merged`.

As a non-LLM baseline we implement a simple Python extractor that reads schemas from gold tables in `training_data.jsonl`. For a test example from a known dataset, this baseline can reproduce the exact schema and provides an upper bound on schema accuracy for that dataset.

3.3 Stage 2: Table policy π_2

Stage 2 maps a pair (x, s) to a table y. The table is emitted as JSONL, one JSON object per row, with keys that must match the schema fields and appear in the same order.

For each training example we provide:

- a system message that explains the task of generating a table that respects a given schema and remains faithful to the document;
- a user message that includes the document x and a schema section that contains s, clearly delimited;
- an assistant message that contains the gold table y in JSONL format.

We again optimize the causal language modeling loss. The trained table policy is published at `https://huggingface.co/mohdusman001/pi2-table-llama3-8b-sft_final`.

During evaluation we also compute a structural sanity score on sample generations. We check that each line parses as JSON, that columns exactly match the schema in content and order, that numeric columns contain well-formed numbers, and that dates and booleans respect simple regular expressions. These checks provide a fast and deterministic measurement of formatting reliability.

3.4 Training configuration

Both π_1 and π_2 start from LLaMA 3.1 8B Instruct and are adapted using LoRA. We attach LoRA adapters to all attention and MLP projection modules (`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`). The LoRA rank is 128, the scaling factor is 256, and the dropout rate is 0.05. Base model weights remain frozen. This configuration adds targeted capac-

ity for schema and table behavior without significantly increasing memory or inference cost.

We use the AdamW optimizer with weight decay 0.01 and a cosine learning-rate schedule. Peak learning rates lie between $5 \cdot 10^{-5}$ and $1.5 \cdot 10^{-4}$, depending on the stage, and warmup ratios fall between 0.03 and 0.08 of the total steps. The cosine schedule decays the learning rate smoothly to ten percent of its maximum. Gradient clipping is disabled; with bfloat16 precision and fully sharded data parallelism we observe stable training.

To balance memory and gradient noise, we use micro-batch sizes between 2 and 8 sequences per GPU and 4 to 16 gradient-accumulation steps, depending on stage and hardware. This yields effective global batch sizes of 16 to 64 sequences. We train in bfloat16 precision with FSDP and gradient checkpointing enabled.

Validation loss and perplexity are monitored on held-out splits. For qualitative analysis we sample 128 generations per checkpoint using nucleus sampling with temperature 0.8 and top-p of 0.95. For the table policy we also track structural sanity rates.

3.5 Inference pipeline

At inference time the system operates as follows:

1. Given a document x, run the schema policy π_1 to produce a candidate schema $\hat{s}$.
2. Run the table policy π_2 conditioned on $(x, \hat{s})$ to generate a table $\hat{y}$.
3. Apply light post-processing to enforce type constraints, remove duplicate rows, and normalize categorical labels.

Figure 1 visualizes this flow, including the baseline zero-shot LLaMA configuration and potential reinforcement learning updates.

4 MONA Multi-Policy Reinforcement Learning Blueprint

Supervised fine tuning provides a solid starting point but does not explicitly optimize table quality metrics or capture user preferences over schemas. The LiveSum and Rotowire results show that multi-step structure can be harmful when it is not rewarded, and that fine-grained row alignment in player tables remains a major challenge.

We therefore design a reinforcement learning extension that follows the **MONA** principle, which stands for *Multi-Objective Non-Aggregating*. In a MONA design, each policy has its own reward function and there is no mixing or redistribution of reward across policies. In our setting, this means that the schema policy and table policy are optimized independently and never receive credit for the other stage.

4.1 Policies and reference models

We maintain two trainable policies and two frozen reference models:

- π_1, the schema policy, initialized from the supervised schema model, with a frozen reference π_1^{ref}.
- π_2, the table policy, initialized from the supervised table model, with a frozen reference π_2^{ref}.

During reinforcement learning, each policy is regularized toward its reference through a token-level KL divergence penalty. The reference encodes the behavior learned during supervised training and helps prevent degenerate outputs.

4.2 Schema rewards

Schema rewards are composed of an immediate component and a slower approval component.

Immediate schema rewards. These rewards reflect structural properties that can be checked deterministically from the schema:

- JSON validity and absence of parsing errors.
- Unique and non-empty column names.
- Use of allowed types and units and reasonable numbers of columns.

Schemas that fail any of these checks receive low immediate reward.

Approval schema rewards. When gold schemas are available, we compute the F1 score between predicted and gold column names, with a small bonus when types and units match. When gold schemas are not available, we use a rubric-based judge that scores the schema on coverage, faithfulness, auditability, and future-proofing given the document.

Total schema reward. We combine the two components as

$$R_{\text{schema}} = \text{clip}\left(0.3R_{\text{imm}} + 0.7R_{\text{appr}}, 0, 1\right),$$

then normalize rewards within each batch, for example using z-score or rank normalization. No table-level metrics enter the schema reward, which preserves the non-aggregating property.

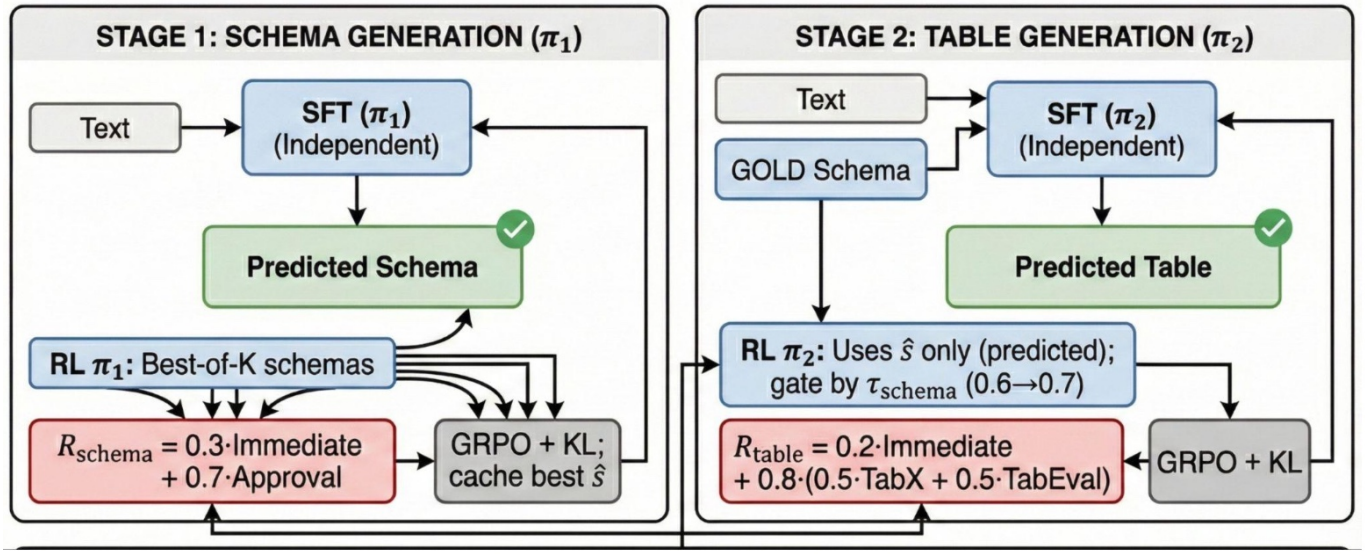

Figure 2: Planned MONA reinforcement learning architecture. The schema policy π_1 and table policy π_2 are optimized independently. Schema rewards use immediate structural checks and approval judges. Table rewards combine structural checks with TabXEval and TabEval scores. Both policies are regularized toward frozen supervised references via KL penalties. Only predicted schemas are used when training π_2 to avoid train–test mismatch.

4.3 Table rewards

Table rewards are defined in an analogous way.

Immediate table rewards. These measure consistency with the schema and basic formatting:

- Each row parses as valid JSON.
- Column sets and order exactly match the schema.
- Values satisfy simple type constraints and row counts lie within reasonable limits.

Approval table rewards. Approval rewards capture semantic quality and are based on TabEval [16] and TabXEval [15] when available. These evaluators assess correctness and completeness of the table content. When they are not available, we combine cell-level F1, row coverage, and penalties for extra or missing rows.

Total table reward. The combined table reward is

$$R_{\text{table}} = \text{clip}\left(0.2R_{\text{imm}} + 0.8R_{\text{appr}}, 0, 1\right),$$

again followed by batch-level normalization. Schema metrics are not included in R_{table}.

4.4 Optimization with group relative preference

Both policies are optimized using a group relative preference objective with KL control. For a document x, we first draw K_s candidate schemas $\{s_i\}$ from π_1 with stochastic decoding. We compute $R_{\text{schema}}(s_i)$ for each candidate and define an advantage A_i as the normalized reward within the group. The schema policy is updated using

$$\mathcal{L}_{\pi_1} = -\sum_{i=1}^{K_s} A_i \log \pi_1(s_i \mid x) + \beta_1 \,\text{KL}\big(\pi_1(\cdot \mid x) \,\|\, \pi_1^{\text{ref}}(\cdot \mid x)\big),$$

where β_1 controls the strength of the KL penalty.

For the table policy, we choose the highest-reward schema $\hat{s}$ as input. If $R_{\text{schema}}(\hat{s})$ is below a threshold τ_{schema} we skip updates for that document. Otherwise we generate M candidate tables $\{y_j\}$ from π_2 conditioned on $(x, \hat{s})$, compute $R_{\text{table}}(y_j)$, and update

$$\mathcal{L}_{\pi_2} = -\sum_{j=1}^{M} A_j \log \pi_2(y_j \mid x, \hat{s}) + \beta_2 \,\text{KL}\big(\pi_2(\cdot \mid x, \hat{s}) \,\|\, \pi_2^{\text{ref}}(\cdot \mid x, \hat{s})\big)$$

This approach focuses learning on the best candidates in each group, prevents the table policy from being trained on obviously bad schemas, and maintains stability by keeping policies close to supervised references.

Implementation and large-scale evaluation of this MONA reinforcement learning framework are left for future work, but the design is specified in sufficient detail to support independent reproduction.

5 Experiments

We now report results for the supervised system without reinforcement learning and analyze where reinforcement

learning is most likely to help.

5.1 Metrics

For LiveSum, let $\{y_i\}_{i=1}^N$ be gold numeric cell values and $\{\hat{y}_i\}_{i=1}^N$ the corresponding predictions. We compute:

- Mean absolute error

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|.$$

- Root mean squared error

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}.$$

- Exact match error

$$\mathrm{EM} = 100\left(1 - \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}[\hat{y}_i = y_i]\right),$$

  expressed as a percentage, where lower is better.

The benchmark reports these metrics separately for Easy, Medium, and Hard subsets and as macro averages across the three.

For Rotowire we use a string similarity metric on team tables that reports correctness, completeness, and an overall combination. We also use TabEval for both team and player tables, again with correctness, completeness, and overall scores.

5.2 LiveSum results

We evaluate four LiveSum configurations:

1. Zero-shot LLaMA 3.1 8B, prompted to output the full statistics table directly.
2. T3 on LLaMA, where we prompt the same model to follow Text-Tuple-Table stages without reinforcement learning.
3. Zero-shot TabRL, where we apply the schema and table prompts on the base model without supervised fine tuning.
4. T3 on TabRL, where we wrap TabRL prompts in the T3 procedure.

The numeric results are shown in Table 1, and Figure 3 visualizes the EM error.

Zero-shot LLaMA performs well on Easy cells (EM 3.51 percent) and moderately on Medium and Hard cells, with macro averages of RMSE 2.65, MAE 1.61, and EM 56.55. Adding the T3 pipeline purely through prompting degrades all metrics, raising macro EM to 67.24. The largest deterioration occurs on Easy cells, which suggests

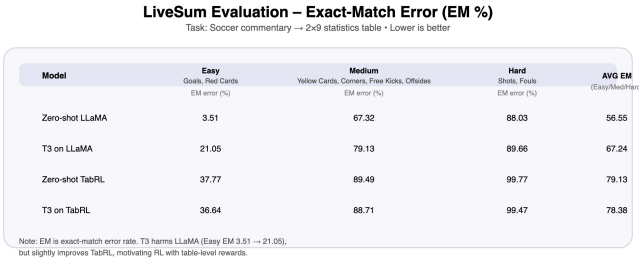

Figure 3: LiveSum exact match error (EM) by difficulty split. T3 structure harms zero-shot LLaMA, particularly on Easy cells, but slightly improves the TabRL backbone.

Figure 4: Rotowire evaluation. TabRL-Summarizer attains the highest team-table completeness but obtains zero TabEval scores on player tables, which motivates reinforcement learning with table-level rewards for fine-grained statistics.

that unnecessary intermediate structure introduces additional opportunities for tuple and aggregation errors.

The TabRL backbone behaves differently. Its zero-shot performance is weaker than that of LLaMA, reflecting more demanding prompts. However, wrapping TabRL in the same T3 procedure slightly improves its macro EM from 79.13 to 78.38, with small but consistent gains on Medium and Hard cells. Here, the intermediate tuple structure appears to help organize information, but the absence of reinforcement limits the benefit.

These patterns indicate that multi-step pipelines can be fragile without explicit table-level rewards, and they point directly to the need for reinforcement learning that optimizes the full text-to-table trajectory.

5.3 Rotowire results

For Rotowire we compare three systems: plain LLaMA 3.1 8B, the Map and Make baseline [1], and our two-stage supervised TabRL-Summarizer. Numerical scores for string similarity and TabEval are summarized in Table 2, and Figure 4 shows the corresponding visualization.

On team tables, TabRL-Summarizer achieves the strongest string similarity: overall score 33.11 compared with 3.79 for LLaMA and 20.79 for Map and Make. Its completeness score of 56.83 indicates that most team-level statistics are captured.

On player tables, however, TabRL-Summarizer fails.

Table 1: LiveSum metrics for four configurations. Lower is better for RMSE, MAE, and EM (error rate).

Setting	E-RMSE	E-MAE	E-EM	M-RMSE	M-MAE	M-EM	H-RMSE	H-MAE	H-EM	A-RMSE	A-MAE	A-EM
Zero-shot LLaMA	0.070	0.036	3.515	2.477	1.666	67.324	3.797	3.055	88.031	2.650	1.606	56.548
T3 on LLaMA	0.520	0.296	21.054	2.949	2.119	79.128	4.004	3.279	89.655	2.985	1.953	67.241
Zero-shot TabRL	1.233	0.738	37.765	6.134	4.640	89.489	11.970	11.217	99.768	7.467	5.309	79.128
T3 on TabRL	1.306	0.776	36.638	6.079	4.527	88.710	11.340	10.459	99.469	7.248	5.072	78.382

Table 2: Rotowire evaluation on team and player tables. String similarity is computed on team tables; TabEval is reported separately for team and player tables (higher is better).

	String similarity, team			TabEval, team			TabEval, player		
Model	C	Comp	Ov	C	Comp	Ov	C	Comp	Ov
LLaMA 3.1 8B	1.238	6.348	3.793	61.5	15.3	20.7	14.9	11.3	11.6
Map and Make	18.045	23.530	20.787	56.2	14.6	20.2	12.9	7.1	7.8
TabRL (two-stage SFT)	9.385	56.829	33.107	50.9	14.8	19.8	0.0	0.0	0.0

TabEval reports zero correctness, completeness, and overall scores. Manual inspection reveals frequent row misalignment, including missing players, duplicated players, and confusion between team and player rows. In contrast, LLaMA and Map and Make achieve modest but nonzero player TabEval scores.

These results suggest that the explicit schema and table stages help organize team-level information but do not by themselves enforce fine-grained row correspondence. Rewarded training that directly penalizes player-level errors is likely required to close this gap.

6 Discussion and Future Work

Our supervised experiments lead to two main conclusions.

First, multi-step structure such as T3 is not inherently beneficial. When introduced only through prompting and without any reward signal, it can significantly degrade performance, especially on cases that are already easy for the base model. This is clearly visible on LiveSum, where T3 hurts zero-shot LLaMA on Easy and Medium cells.

Second, the two-stage TabRL-Summarizer backbone provides strong structure and high completeness for team-level tables on Rotowire but fails on player-level tables. The failure is not due to formatting but to missing or misaligned rows. This type of error is difficult to correct through token-level supervised learning alone.

The MONA reinforcement learning design directly addresses both issues. By giving the schema and table policies separate rewards, by training with best-of-K sampling under strict structural checks, and by using KL penalties to keep policies close to supervised references, the method provides a principled way to improve intermediate representations and end tables simultaneously. We expect that applying this framework will reduce LiveSum error, particularly for Easy and Medium cells, and will substantially improve player-level TabEval scores on Rotowire.

Future work will implement this MONA reinforcement learning system, study its stability and data efficiency, and evaluate its impact across a wider range of domains, including financial tables and scientific tables.

7 Individual Contributions

Thanishka Bolisetty

- Led the literature review, dataset preparation, and baseline reproduction for the project.
- Designed and implemented **TabEval**, a semantic evaluator and reinforcement-learning reward for table generation that prompts from intent text plus Markdown layout to identify headers and rows, infer primary keys, and emit one atomic fact per row as JSON before cleaning and deduplicating the set of facts.
- Integrated RoBERTa-large-MNLI to score textual entailment over a matrix of predicted vs. gold facts, computing precision, recall, and F1, and used the resulting R_{TabEval} signal to drive GRPO-based fine-tuning.
- Implemented evaluation scripts for the LiveSum benchmark and the T3 baseline, including data pipelines, metric computation, and comparison against TabRL-Summarizer models.

Arya Ravindra Patil

- Worked on model implementation, ablations, and the training and evaluation harness for the two-stage text-to-table pipeline.
- Contributed to data research by identifying and shortlisting relevant text-to-table datasets for supervised training and evaluation.
- Prototyped and analysed TabXEval as a structural table metric, with the goal of using it later as a reward component in the reinforcement learning stage.
- Implemented a TabEval-based semantic evaluation pipeline that converts tables into factual statements and uses an NLI model to compute precision, recall, and F1 between golden and predicted tables for baseline comparisons.

Hriday Pradhan

- Set up, ran, and iteratively debugged the LiveSum and T3 evaluation pipeline, including zero-shot LLaMA, T3-on-LLaMA, zero-shot TabRL, and T3-on-TabRL configurations; collected RMSE, MAE, and EM metrics and performed comparative analysis that now drives our reinforcement-learning design choices.
- Developed the second supervised fine-tuning (SFT2) process for row population, including data pipelines, training routines, and evaluation steps that plug into both the LiveSum and Rotowire-style workflows.
- Implemented reinforcement-learning components with VERL and studied the Sol framework, laying the groundwork for future reinforcement-learning phases that optimize table quality directly from evaluation metrics.
- Reverse-engineered and documented the LiveSum and T3 evaluation code from scripts and guide PDFs, clarifying the multi-step schema, tuple, and table pipeline and ensuring that experiments are reproducible and configurable.
- Performed a focused literature review on schema generation and domain adaptivity, identifying gaps and synthesizing findings into collated written reports and project documentation.

Simran Sharma

- Contributed to model implementation, ablation studies, and development of the full training and evaluation harness.
- Led model selection across decoder-only LLMs by evaluating instruction-tuning quality, JSON and tool reliability, context-length scaling, inference latency and cost, and licensing constraints, resulting in a deployable shortlist.
- Built a GRPO-based reinforcement-learning pipeline using VERL on GSM8K, including reward and KL design, training-loop implementation, monitoring, and schema-violation penalties, producing a reusable reinforcement-learning workflow for the team.
- Developed a multi-stage Rotowire knowledge-extraction system comprising data preprocessing, atomic fact extraction, and schema inference.
- Implemented vLLM-based inference with HPC-safe multiprocessing, CUDA initialization handling, offline Hugging Face caching, and authenticated access to gated models.
- Authored robust Python pipelines (`preparedata`, `atomize_llama31`, `schema_llama31`) featuring JSONL streaming, error handling, and reproducible output artifacts.

Mohammed Usman Mehboob

- Led the overall system design and implementation of the TabRL-Summarizer backbone, from initial problem formulation through final integration of schema and table models.
- Prepared, collected, cleaned, and standardized all training datasets, including E2E, Rotowire, WikiBio, and LiveSum, to ensure consistent input quality and unified formatting across sources.
- Designed the two-stage supervised pipeline architecture, including the separation between schema and table policies, the choice of message formats, and the handoff of predicted schemas between stages.
- Implemented the full supervised training stack for Stage 1 (schema generation), including JSONL data preparation, LoRA configuration, optimization hyperparameters, and large-scale training with FSDP and FlashAttention.
- Implemented the full supervised training stack for Stage 2 (table generation), including incorporation of predicted schemas, structural sanity checking, and generation logging for downstream analysis.
- Ran and monitored all large-scale SFT experiments for both stages, tuned learning-rate schedules and batch sizes for stability, and curated the final checkpoints that were released on Hugging Face.
- Developed the inference pipeline that connects schema prediction, table generation, and post-processing into a single callable system for all evaluation benchmarks.

- Studied the MONA architecture and translated it into a concrete multi-policy reinforcement-learning design for this project, specifying how rewards are computed, how schemas gate table training, and how GRPO with KL control is applied to each policy.
- Authored internal documentation and experiment logs that capture training configurations, decision rationales, and failure analyses, thereby enabling reproducibility and future extension of the system.

References

[1] N. Ahuja, F. Bardoliya, C. Baral, and V. Gupta. Map and Make: Schema Guided Text-to-Table Generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025, pp. 30249–30262.

[2] Z. Wu, L. Li, Y. Zhang, and C. Tang. Table-R1: Region-based Reinforcement Learning for Table Understanding. *arXiv preprint* arXiv:2505.12415, 2025.

[3] Z. Yang, L. Chen, A. Cohan, and Y. Zhao. Table-R1: Inference-Time Scaling for Table Reasoning. *arXiv preprint* arXiv:2505.23621, 2025.

[4] Q. Dong, X. Hu, H. Peng, and Y. Cao. TableCoder: Table Extraction from Text via Reliable Code Generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025, pp. 1532–1545.

[5] A. Oestreich and T. Müller. Evaluating Structured Decoding for Text-to-Table Generation: Evidence from Three Datasets. *arXiv preprint* arXiv:2508.15910, 2025.

[6] B. Newman, S. Narayan, and I. Augenstein. Synthesizing Scientific Literature into Tables using Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024, pp. 1123–1138.

[7] X. Zheng, L. Wang, and H. Li. FinTabNet: Large-Scale Financial Table Dataset for NLP. *arXiv preprint* arXiv:2410.12345, 2024.

[8] E. Zhong, C. Xiong, and R. Socher. PubTabNet: Large-Scale Table Recognition in Scientific Articles. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 1905–1916.

[9] C. Zhang, X. Chen, and D. Li. WikiTableTQA: Open-Domain Table Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[10] S. Wiseman, A. Shieber, and A. Rush. Challenges in Data-to-Text Generation from Sports Tables: The Rotowire Dataset. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2017, pp. 1431–1441.

[11] H. Deng, M. Li, and J. Zhang. Text-Tuple-Table Summarization with LiveSum. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024, pp. 2210–2223.

[12] P. Roit, J. Ferret, L. Shani, R. Aharoni, G. Cideron, R. Dadashi, M. Geist, S. Girgin, L. Hussenot, O. Keller, N. Momchev, S. Ramos, P. Stanczyk, N. Vieillard, O. Bachem, G. Elidan, A. Hassidim, O. Pietquin, and I. Szpektor. Factually Consistent Summarization via Reinforcement Learning with Textual Entailment Feedback. *arXiv preprint* arXiv:2306.00186, 2023.

[13] A. Sundar, C. Richardson, and L. Heck. gTBLS: Generating Tables from Text by Conditional Question Answering. *arXiv preprint* arXiv:2403.14457, 2024.

[14] S. Liu, J. Cao, R. Yang, and Z. Wen. Long Text and Multi-Table Summarization: Dataset and Method. *arXiv preprint* arXiv:2302.03815, 2023.

[15] V. Pancholi, J. Bafna, T. Anvekar, M. Shrivastava, and V. Gupta. TabXEval: Why this is a Bad Table? An Exhaustive Rubric for Table Evaluation. *arXiv preprint* arXiv:2505.22176, 2025.

[16] P. Ramu, A. Garimella, and S. Bandyopadhyay. Is This a Bad Table? A Closer Look at the Evaluation of Table Generation from Text. *arXiv preprint* arXiv:2406.14829, 2024.