Hybrid Graphs for Table-and-Text based Question Answering using LLMs

Ankush Agarwal, Ganesh S, Chaitanya Devaguptapu

Fujitsu Research of India

Contact: ankush.agarwal@fujitsu.com

NAACL 2025

ALBUQUERQUE, NEW MEXICO

APRIL 29-MAY 4, 2025

Problem Definition

Table-Text QA: The goal is to answer complex questions that require reasoning over both unstructured text (documents) and structured data (tables).

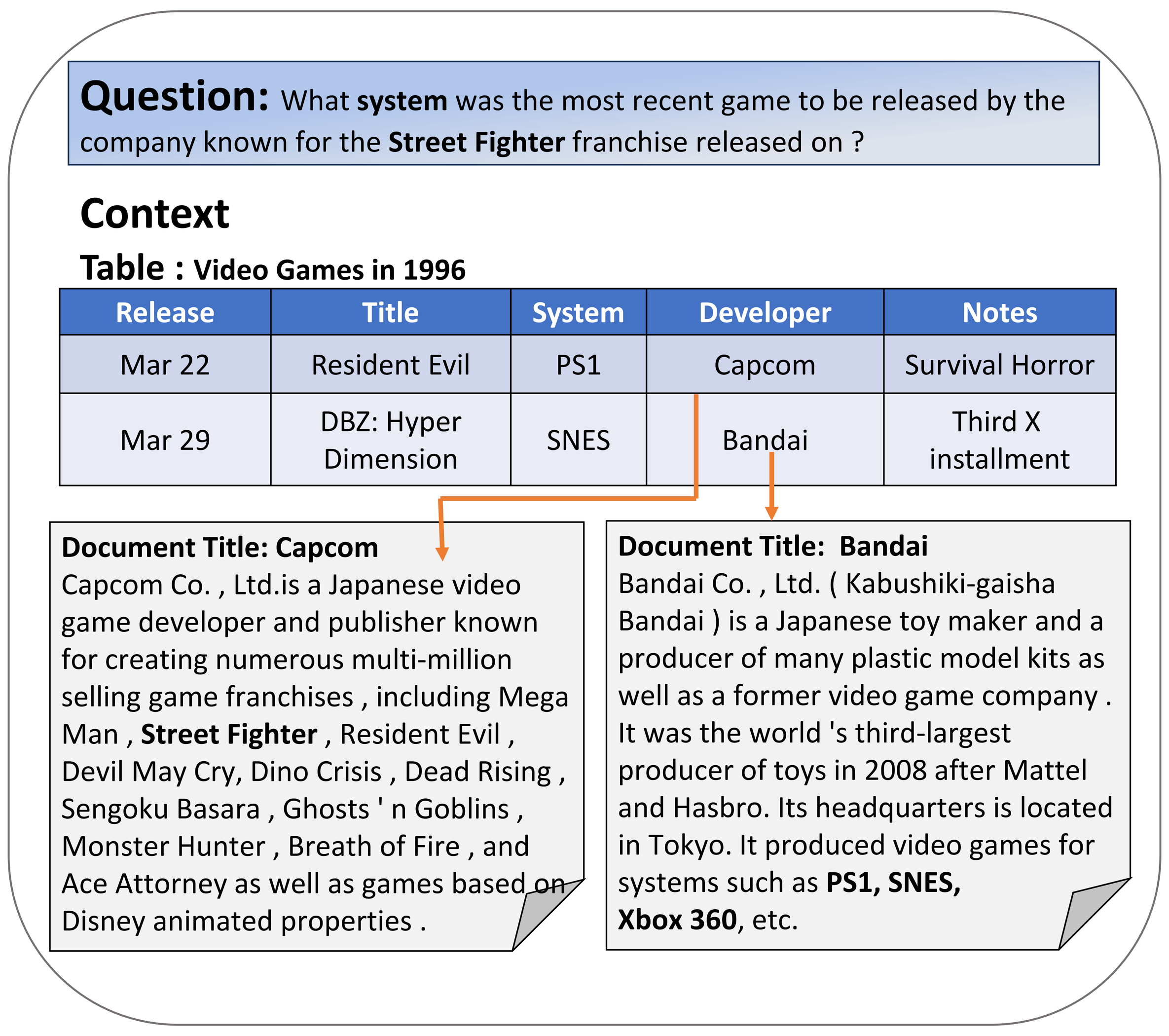

Figure 1: An illustrative example of the Table-Text QA

Contributions

- A novel approach, ***ODYSSEY,*** jointly distills information from structured and unstructured data sources to construct a Hybrid Graph.
- An increase in performance over the current SoTA fine-tuning-free approach, improving EM and F1 scores by 7.3% and 20.9% for Hybrid-QA using GPT-4.
- A significant reduction in the input token size. Our Hybrid Graph based approach uses up to 45% and 53% fewer tokens than the original table and text for the Hybrid-QA and OTT-QA datasets respectively.

ODYSSEY: A Hybrid Graph Approach

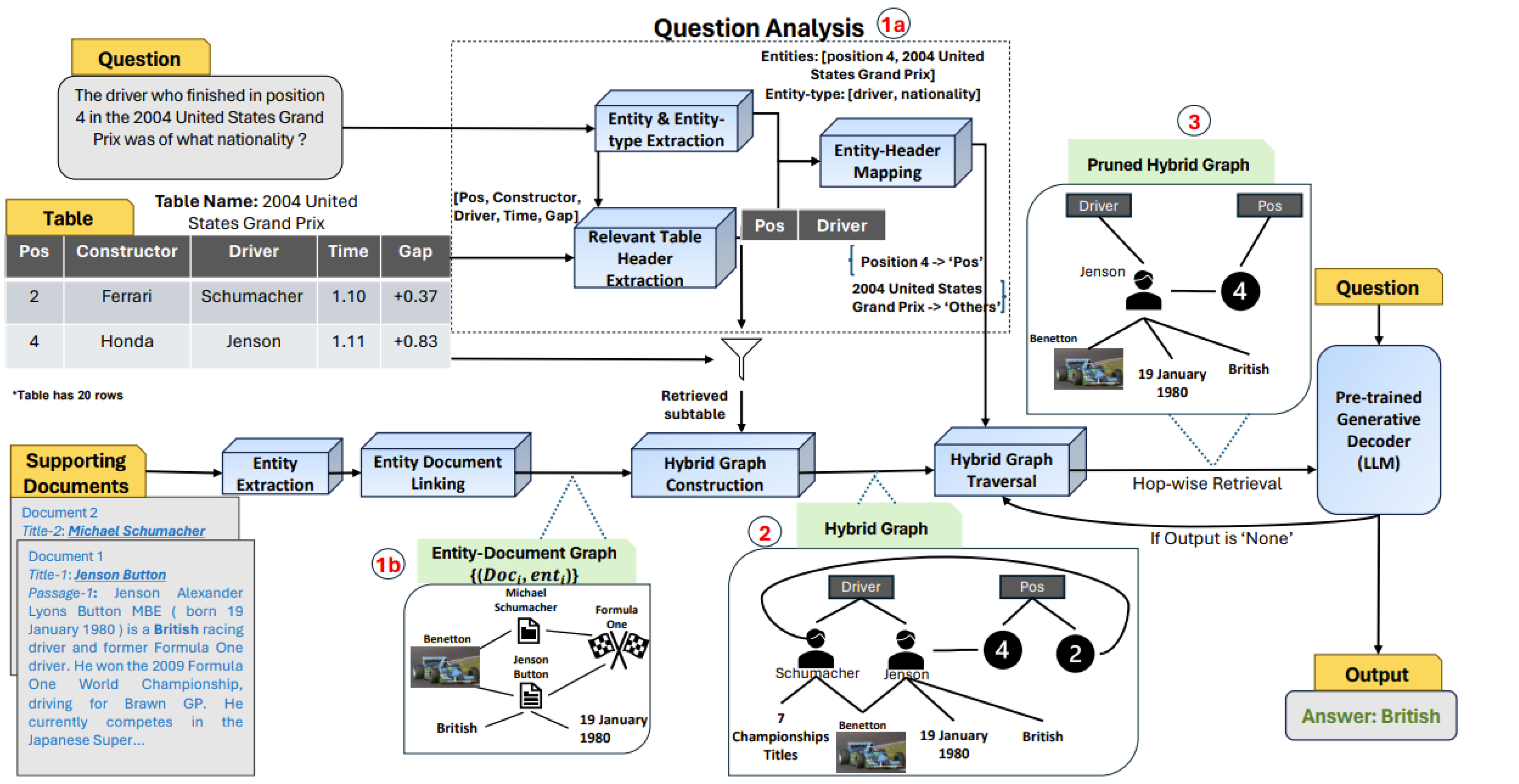

Figure 2: **Overview of the ODYSSEY framework**. Our method comprises of 3 steps: i) Question Analysis, ii) Hybrid Graph Construction, and iii) Hybrid Graph Traversal. First, we begin with Question Analysis (1a in the figure) from where we get question entities, retrieved sub-table, and entity-header mapping. Next, we construct the Entity-Document Graph (1b in the figure). Using entity-doc graph and retrieved sub-table, we construct the Hybrid Graph (2 in the figure). At last, we perform Hybrid Graph Traversal (3 in the figure) to get the pruned graph which serves as input for the LLM.

Results

Findings:

1. Our method achieves the best performance in a zero-shot setting across various LLMs.
2. For Hybrid-QA, our method performs comparably to fine-tuning-based approaches and outperforms them on the OTT-QA dataset.

Datasets	Hybrid-QA					OTT-QA				
Methods	EM (↑)	F1 (↑)	P (↑)	R (↑)	B (↑)	EM (↑)	F1 (↑)	P (↑)	R (↑)	B (↑)
Reader: gpt-4-1106-preview (zero-shot)										
Base	4.60	12.44	12.08	12.25	62.10	4.85	12.44	12.25	14.24	64.30
Base w/ Table & Text (Zhang et al., 2023)	55.40	68.84	68.92	71.79	85.54	58.86	72.28	72.16	74.28	87.51
Base w/ Table & Summarized Text[2]	45.29	58.72	58.78	61.39	81.14	48.31	60.90	61.47	63.12	82.02
Our Method w/o hopwise	58.20	71.54	71.75	74.35	86.30	61.00	72.64	73.60	74.27	88.06
Our Method w/ hopwise	**58.40**	**71.80**	**71.62**	**74.22**	**86.53**	**62.02**	**73.02**	**73.40**	**75.13**	**88.18**
Reader: gpt-3.5-turbo-1106 (zero-shot)										
Base	4.20	11.54	11.62	12.45	65.05	5.27	12.20	12.35	13.44	66.17
Base w/ Table & Text (Zhang et al., 2023)	40.22	53.47	54.18	55.57	81.18	42.41	54.06	54.04	**55.8**	81.63
Base w/ Table & Summarized Text[2]	41.19	51.64	52.03	53.12	81.05	37.34	49.58	49.80	51.73	79.87
Our Method w/o hopwise	41.8	52.37	52.82	53.72	81.31	42.19	53.61	54.18	55.30	81.58
Our Method w/ hopwise	**44.2**	**55.82**	**55.28**	**56.90**	**83.98**	**44.30**	**54.08**	**55.04**	54.67	**81.73**
Reader: Llama3-8B (zero-shot)										
Base	2.0	7.07	6.91	7.07	59.05	0.64	7.00	6.77	8.72	59.71
Base w/ Table & Text (Zhang et al., 2023)	28.6	37.05	37.22	48.07	74.01	33.12	43.43	43.53	45.12	76.75
Base w/ Table & Summarized Text[2]	30.33	39.42	39.60	41.30	75.06	31.22	41.72	42.42	42.88	75.57
Our Method w/o hopwise	33.2	41.37	41.77	42.95	75.15	36.08	45.75	46.60	45.75	77.04
Our Method w/ hopwise	**37.0**	**46.43**	**46.56**	**48.78**	**77.55**	**37.13**	**47.38**	**48.24**	**48.31**	**77.62**

Table 1: **Table-Text QA Evaluation:** We analyze Exact Match (EM), F1-Score, Precision (P), Recall (R), and BERTScore-F1 (B) in (%) to compare our method against baselines in a zero-shot setting using Llama3-8B, GPT-3.5, and GPT-4. The results consistently demonstrate significant improvements across datasets, metrics, and various language models. Base (only reader LLM); w/ Table & Text (table and passages relevant to the question); w/ Table & Summarized Text (table with summarized supporting passages); w/o hopwise (pruned information without considering hop-wise extraction).

Method	EM (↑)	F1 (↑)
Hybrid-QA Fine-Tuning		
HYBRIDER (Chen et al., 2020b)	43.5	50.6
HYBRIDER-LARGE (Chen et al., 2020b)	44.0	50.7
DocHopper (Sun et al., 2021)	47.7	55.0
MuGER[2] (Wang et al., 2022)	57.1	67.3
S^3HQA (Lei et al., 2023) [SoTA]	**68.4**	**75.3**
w/o Fine-Tuning		
Unsupervised-QG (Pan et al., 2021)	25.7	30.5
GPT-4[†] *w.* Retriever (Shi et al., 2024)	24.5	30.0
GPT-4[†] + CoT (Wei et al., 2022)	48.5	63.0
HProPro[†] (Shi et al. (2024), ACL 2024)	48.0	54.6
ODYSSEY[†] (Our Method)	**51.5**	**66.0**

Method	EM (↑)	F1 (↑)
OTT-QA Fine-Tuning		
BM25-HYBRIDER (Chen et al., 2020a)	10.3	13.0
Fusion+Cross-Reader (Chen et al., 2020a)	28.1	32.5
CARP (Zhong et al., 2022)	33.2	38.6
CORE (Ma et al., 2022)	49.0	55.7
COS (Ma et al., 2023) [SoTA]	**56.9**	**63.2**
w/o Fine-Tuning		
GPT-4[†] + CoT (Wei et al., 2022)	61.0	72.3
ODYSSEY[†] (Our Method)	**62.02**	**73.02**

Table 2: **Performance comparison of ODYSSEY with fine-tuning-based and fine-tuning-free approaches.** We evaluate our method against state-of-the-art fine-tuning-based methods as well as approaches without fine-tuning using GPT-4.

Analysis

Efficient Query Context handling

Method	Input Token Size (↓)	Input Token Cost (↓)
Dataset: Hybrid-QA		
Original Context	7195	$71.95
Summarized	3923	$39.23
Our Method	3857	$38.57
Dataset: OTT-QA		
Original Context	5866	$58.66
Summarized	3778	$37.78
Our Method	2745	$27.45

Table 3: **Reader Input Token Count and Cost:** We compare our method with baselines on average reader input token size and its pricing in dollars w.r.t. GPT-4 Turbo OpenAI pricing for 1000 samples.

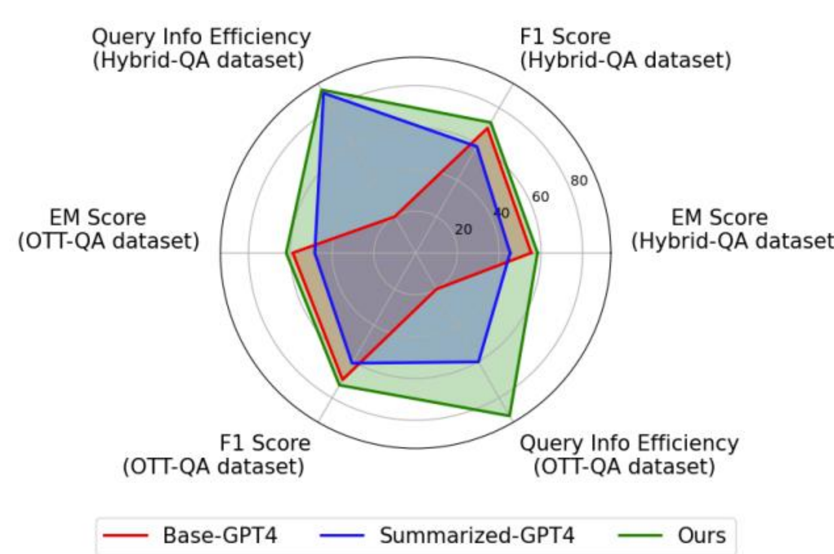

Figure 3: **Multi-Dimensional Improvements:** Our method (with GPT-4 as reader LLM) demonstrates superior results on Hybrid-QA and OTT-QA.

ODYSSEY Hopwise Analysis

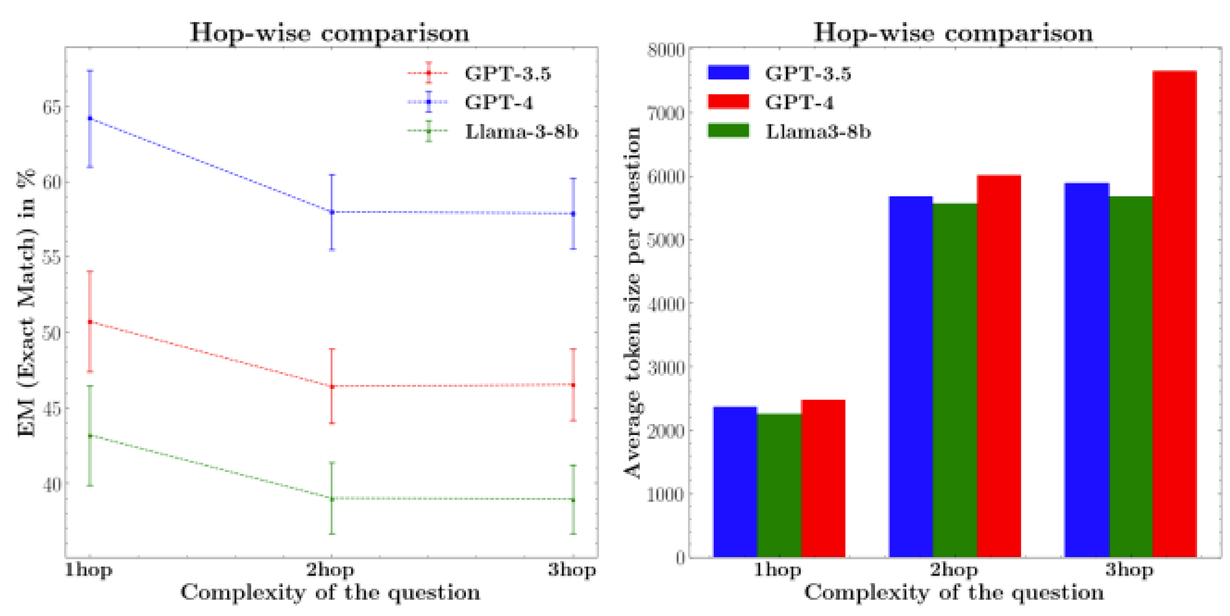

Figure 4: **Hopwise analysis:** For ODYSSEY (our method w/ hopwise), we calculate the cumulative EM score (left-side in figure) and average token size (right-side in figure) utilized after each hop for Llama3-8B, GPT3.5, and GPT-4 on Hybrid-QA.

Findings:

1. Our method effectively reduces and prunes the input token size for LLMs, enhancing efficiency.
2. The reduction in token size directly correlates with a decrease in computational cost.