Pipeline Neural Data-to-text with Large Language Models

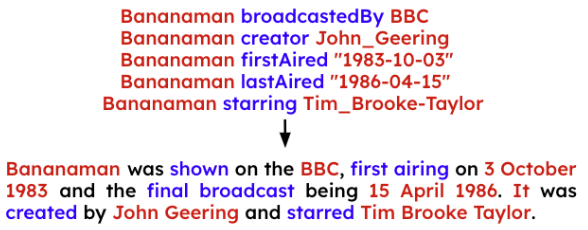
Chinonso Cynthia Osuji[♡♣], **Brian Timoney**[♣], **Thiago Castro Ferreira**[◇], **Brian Davis**[♡♣]
Adapt Research Centre, Ireland[♡]
Dublin City University, Ireland[♣]
aiXplain, USA[◇]
chinonso.osuji@adaptcentre.ie brian.timoney3@mail.dcu.ie
thiago@aixplain.com brian.davis@adaptcentre.ie

Abstract

Previous studies have highlighted the advantages of pipeline neural architectures over end-to-end models, particularly in reducing text hallucination. In this study, we extend prior research by integrating pretrained language models (PLMs) into a pipeline framework, using both fine-tuning and prompting methods. Our findings show that fine-tuned PLMs consistently generate high quality text, especially within end-to-end architectures and at intermediate stages of the pipeline across various domains. These models also outperform prompt-based ones on automatic evaluation metrics but lag in human evaluations. Compared to the standard five-stage pipeline architecture, a streamlined three-stage pipeline, which only include ordering, structuring, and surface realization, achieves superior performance in fluency and semantic adequacy according to the human evaluation.

1 Introduction

Advancements in data-to-text natural language generation (NLG) have evolved from seq2seq models (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) and vanilla encoder-decoder models (Vaswani et al., 2017) towards pretrained language models (PLMs) (Raffel et al., 2020; Lewis et al., 2019; Radford et al., 2019) . Initially, PLMs were fine-tuned on specific datasets to perform text generation tasks. Recently, these models are prompted with textual instructions, with or without examples, to guide text generation (zero-shot and few-shot learning). Although PLMs excel in several natural language processing tasks, they face challenges in generating text from complex structured data due to the intricate demands of accuracy and structure (Kasner and Dušek, 2024). Despite these challenges, PLMs demonstrate superior performance in generating high-quality text under fine-tuned, few-shot, or zero-shot learning scenarios, leveraging extensive pre-training on general knowledge.



Bananaman broadcastedBy BBC
Bananaman creator John_Geering
Bananaman firstAired "1983-10-03"
Bananaman lastAired "1986-04-15"
Bananaman starring Tim_Brooke-Taylor

Bananaman was shown on the BBC, first airing on 3 October 1983 and the final broadcast being 15 April 1986. It was created by John Geering and starred Tim Brooke Taylor.

Figure 1: A sample of the input triples and the expected output.

In a previous study, Ferreira et al. (2019) compared traditional 5-stage pipeline approaches to end-to-end neural methods, utilizing systems such as GRU (Cho et al., 2014) and the BERT transformer (Vaswani et al., 2017). The pipeline approach, despite lacking pretraining or fine-tuning, outperformed the end-to-end method in automatic and human evaluations, especially in domains not seen in the training phase.

Building on Ferreira et al. (2019), this study integrates PLMs and large language models (LLMs) into the pipeline architecture to compare their effectiveness against the baseline. We assess the generalization capabilities of pipeline neural architectures and end-to-end systems under fine-tuned and few-shot settings, also proposing a simplified 3-stage pipeline architecture. Automatic evaluations and human assessments of the results highlight a preference for end2end architecture and the potential for optimized pipeline designs. The code and results are publicly available[1].

2 Related Work

End-to-End (E2E) architectures, while simplifying generation processes, face limitations due to the lack of intermediate steps, which can hinder control over semantic fidelity (Kasner and Dušek, 2020; Ferreira et al., 2019). Researchers have increasingly adopted pipeline architectures for data-to-text tasks, leveraging diverse deep neural network mod-

[1] https://github.com/NonsoCynthia/PipeD2T

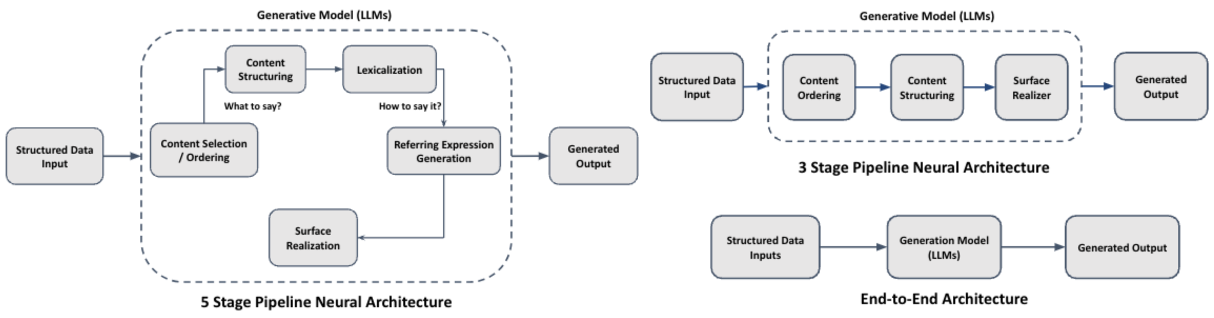


Figure 2: Experimental Setup.

els (Moryossef et al., 2019; Ferreira et al., 2019; Kasner and Dusek, 2022).

The data-to-text generation pipeline, originally delineated by (Reiter and Dale, 1997) and refined by (Ferreira et al., 2019) with deep neural models, involves several stages: content selection/ordering, content aggregation/structuring, lexicalization, Reference Expression Generation (REG), and surface realization (SR), details of which is explained in the Appendix A and broader in the study. This comprehensive approach integrates neural techniques to convert structured data into readable text, with linguistic rules for the surface realizer. Unlike this architecture, some studies use simplified pipeline neural architectures with fewer stages, focusing on content selection, structuring, and textual realization. For example, Moryossef et al. (2019); Zhao et al. (2020) divides text generation into planning and realization stages, using ordered trees or relational graph convolutional networks (R-GCN) (Zhao et al., 2020) to guide the neural generation system, providing explicit control over the output.

Recent research has utilized PLMs like T5 (Raffel et al., 2020) and BART (Lewis et al., 2019) for both pipeline and end-to-end data-to-text generation, achieving more fluent text than human references (Ribeiro et al., 2020). This is evident from the top competitor (Guo et al., 2020) in the WebNLG'20 (Castro Ferreira et al., 2020) competition. Studies have also shown that these PLMs when fine-tuned outperform generative LLMs like GPT-3.5 (Ye et al., 2023) in prompt-based scenarios, reducing hallucinations and over-generation issues (Yuan and Färber, 2023; Axelsson and Skantze, 2023), which are pivotal areas of investigation in our current study. By integrating PLMs into both traditional and simplified pipeline architectures, our research seeks to quantify their impact on the fidelity and fluency of generated text, particularly under fine-tuned and few-shot conditions.

3 Methodology

3.1 Data

We utilize the enhanced WebNLG'17 English dataset (Castro Ferreira et al., 2018), a derivative of the WebNLG corpus (Gardent et al., 2017), which includes 25,298 texts describing 9,674 sets of up to 7 RDF triples across 15 domains. Five of these domains are exclusive to the test set, making them *unseen* during training, while the remaining 10 domains are *seen*. These domain distinctions pose challenges for model generalization and domain adaptation. For the intermediate stages of our pipeline, we utilized a specially curated dataset that includes specific inputs and expected outputs for each stage. However, the outputs from the Surface Realization (SR) stage are evaluated against the gold standard provided by the WebNLG'17 test set.

3.2 Models

To evaluate the performance and suitability of end-to-end and pipeline architectures, we employed fine-tuned models such as GPT-2-*large* (Radford et al., 2019), BART-*large* (Lewis et al., 2019), Flan-T5-*large* (Chung et al., 2022), as well as instruction-based models like GPT-3.5 and GPT-4 Turbo (Ye et al., 2023; Achiam et al., 2023) OpenAI models, Cohere Command Text v14 (Üstün et al., 2024), and Mistral-7B-Instruct-v0.1 (Jiang et al., 2023). The Cohere and OpenAI models were accessed through the aiXplain platform (Sharma et al., 2024). We set learning rates to 3e-5 for BART, 5e-5 for GPT-2, and 1e-5 for the Flan-T5 model.

3.3 Pipeline Architecture

We implemented two experimental setups for the pipeline architecture. The first setup is a 5-stage neural pipeline architecture consisting of ordering, structuring, lexicalization, REG, and surface

	Ordering			**Structuring**			**REG**			**Lexicalization**								
Domains	**All**	**Seen**	**Unseen**	**All**	**Seen**	**Unseen**	**All**	**Seen**	**Unseen**	**All**			**Seen**			**Unseen**		
Metrics										**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**
Baseline	0.34	0.56	0.09	0.36	0.59	0.12	0.39	0.70	0.07	38.12	0.55	0.75	**48.14**	0.6	0.76	24.15	0.49	0.71
Flan-t5	**0.57**	**0.65**	**0.48**	0.53	**0.67**	0.39	**0.58**	**0.72**	0.45	**45.37**	**0.60**	**0.76**	45.72	**0.62**	**0.77**	**44.33**	**0.58**	**0.75**
bart	0.49	0.60	0.36	**0.58**	0.61	**0.54**	0.56	0.66	0.46	19.87	0.39	0.64	20.16	0.40	0.64	19.45	0.39	0.63
gpt2	0.37	0.57	0.15	0.40	0.63	0.16	0.43	0.69	0.17	40.37	0.57	0.75	43.87	0.59	0.76	36.04	0.54	0.73
gpt4	0.37	0.33	0.43	0.46	0.48	0.43	–	–	–	38.28	0.53	0.74	37.92	0.53	0.74	38.70	0.53	0.74
gpt-3.5	0.39	0.32	0.47	0.48	0.50	0.47	0.48	0.48	**0.47**	29.58	0.46	0.69	31.23	0.47	0.70	27.63	0.45	0.68
Mistral7b	0.28	0.24	0.33	0.28	0.29	0.28	0.00	0.00	0.00	18.43	0.36	0.55	14.16	0.33	0.51	23.21	0.39	0.59
Cohere	0.24	0.23	0.26	0.16	0.18	0.14	0.30	0.30	0.30	3.56	0.14	0.33	4.26	0.13	0.33	2.70	0.14	0.33

	End2end									**SR**								
Domains	**All**			**Seen**			**Unseen**			**All**			**Seen**			**Unseen**		
Metrics	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**
Baseline	31.88	**0.45**	0.61	50.79	**0.39**	0.76	5.88	0.09	0.45	**51.68**	0.32	0.67	**56.35**	**0.41**	**0.77**	**38.39**	0.21	0.56
Flan-t5	**51.55**	0.32	**0.81**	**53.05**	0.33	**0.81**	**49.71**	0.30	**0.80**	40.58	0.28	**0.69**	46.61	0.30	0.71	33.13	0.26	**0.67**
bart	41.41	0.31	0.79	49.85	0.32	**0.81**	31.25	0.30	0.76	18.69	0.26	0.51	23.43	0.27	0.54	12.61	0.24	0.49
gpt2	38.03	0.31	0.75	49.19	0.32	0.80	22.96	0.29	0.70	21.37	0.21	0.53	31.85	0.26	0.61	7.84	0.15	0.44
gpt4	41.43	0.32	0.80	40.50	0.32	0.80	42.55	**0.32**	**0.80**	10.73	0.23	0.50	11.85	0.23	0.50	9.30	0.22	0.49
gpt-3.5	39.95	0.32	0.80	39.16	0.32	0.80	40.90	0.31	**0.80**	21.69	0.30	0.60	21.68	0.31	0.59	21.69	0.29	0.62
Mistral7b	34.33	0.32	0.78	33.61	0.33	0.78	35.07	0.31	0.78	7.59	**0.39**	0.56	7.50	0.37	0.57	7.72	**0.40**	0.55
Cohere	40.40	0.30	0.79	39.00	0.31	0.79	42.08	0.30	0.79	21.63	0.28	0.64	21.29	0.28	0.64	22.04	0.27	0.65

Table 1: Results from the individual stages of the 5-stage pipeline and the end-to-end data-to-text systems. Bold and underlined results denote the best and the second best ones respectively.

Domains	**All**			**Seen**			**Unseen**		
Metrics	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**	**Bleu**	**Meteor**	**Comet**
gpt4	**40.17**	0.31	**0.79**	**39.17**	0.32	**0.80**	**41.39**	0.30	0.78
gpt-3.5	39.37	**0.32**	**0.79**	38.46	**0.33**	**0.80**	40.25	**0.31**	**0.79**
mistral7b	28.09	0.29	0.71	29.52	0.30	0.74	26.15	0.27	0.69

Table 2: Surface realization results of the 3-stage pipeline architecture (Struct2SR).

realization. We fine-tuned the PLMs on task-specific gold datasets and used five-shot examples to prompt the instruction-based LLMs for each task. In the ordering and structuring stages, predicates served as pointers and were mapped to their respective triples after generation. The output from the lexicalization stage was mapped to the corresponding entities from the structuring stage. The REG stage results were then passed to the surface realizer, which uses hand-crafted rules to produce the final output. The results for the intermediate stages are sourced from a gold standard test set, ensuring both input and expected output accuracy. In our pipeline approach, each stage methodically processes its input and passes the resulting output to the subsequent stage, culminating in the surface realization (SR) stage. However, comprehensive evaluations are concentrated at this final SR stage, providing a measure of the overall performance based on the integrated outputs from all preceding stages.

Due to the high performance of state-of-the-art neural models, some proposed pipeline approaches decrease the number of stages, simplifying the generation process (Guo et al., 2020). In this direction, our second setup is a streamlined 3-stage pipeline architecture consisting of ordering, structuring, and surface realization. Here, the outputs from the structuring stage in the 5-stage setup are directly fed into the surface realization models, such as GPT-3.5, GPT-4 Turbo, and Mistral7b. This configuration uses five-shot examples to facilitate the generation of the final text, focusing on optimizing the pipeline's efficiency and minimizing error accumulation through reduced complexity. Detailed representations of these setups and examples of the prompts used are available in Appendix A for further reference.

3.4 End2End Surface Realizer

In this approach, we fine-tuned Flan-T5, BART, and GPT-2 on our end-to-end dataset. For GPT-4 Turbo, GPT-3.5, Cohere, and Mistral7b, we used prompt engineering with tailored instructions and 5-shot examples of end-to-end data to achieve the desired data-to-text generation.

3.5 Metrics

The performance of the models across various pipeline stages, including discourse ordering, structuring, and referring expression generation, was assessed using accuracy. This evaluation method compared the models' predictions against a single gold-standard reference due to the multiple verbalizations of triples in the input stages. For the remaining pipeline stages—lexicalization and surface realization—as well as the outputs of the end-to-end experiment, evaluation was conducted using Meteor (Banerjee and Lavie, 2005) and Bleu

(Papineni et al., 2002). Additionally, we included the Comet neural metric (Rei et al., 2020), known for its strong correlation with human judgments.

4 Results

Table 1 presents the performance outcomes for each stage of the 5-stage pipeline, as well as for the end-to-end architecture. The baseline results are based on the transformer model from Ferreira et al. (2019), evaluated across both the individual pipeline stages and the end-to-end architecture. To ensure clarity, we initially focus on comparing the performance of the fine-tuned models Flan-T5, GPT2, and BART across these stages. Subsequently, we compare the performance of prompt-based models GPT-3.5, GPT4-turbo, Cohere and Mistral7b. Finally, we draw a general conclusion regarding the overall performance of the models across the pipeline stages.

Fine-tuned models Across *all* domains, Flan-T5 surpasses both BART and GPT-2, except for the structuring stage where BART excels. In the *seen* category, Flan-T5 maintains its superiority across all pipeline stages compared to GPT-2 and BART. Notably, GPT-2 closely competes with BART, particularly in the ordering stage where BART outperforms. In the *unseen* domain (referenced in Table 1), Flan-T5 and BART regularly outperform GPT-2 across various stages, including ordering, structuring, and referring expression generation (REG). However, in the lexicalization stage, GPT-2 outshines BART in this domain.

In the surface realization stage of the pipeline architecture, the baseline model seemed to perform best followed by the Flan-T5 model. All other model seemed to perform poorly. But in general the fine-tuned models performed best.

Prompt-based LLMs Due to the substantial costs linked to proprietary models like GPT-4 Turbo, we limited their application to specific stages of the pipeline and for end-to-end data-to-text generation. To control expenses, we refrained from generating referring expressions for evaluation from the gold standard inputs due to the extensive dataset involved. Nonetheless, we did produce results for the Referring Expression Generation (REG) stage within the pipeline, where the inputs were directly sourced from the mapped lexicalization outputs of the pipeline itself. The results of these models in Table 1 indicate that the performance of the Cohere model across several pipeline stages was notably inferior, followed by the results of the Mistral7b model. However, GPT-3.5 was seen to perform better than GPT4-turbo in the ordering and structuring stage but an exception is observed in the *seen* category of the ordering stage and in all categories of the lexicalization stage where it trailed behind GPT4-turbo.

Fine-tuned vs. Prompt-based models Overall, in comparing fine-tuned and instruction-based models in Table 1, we noticed better performance in the fine-tuned models compared to the prompt-based model. Furthermore, it's worth highlighting that GPT-3.5 exhibited exceptional performance in the REG *unseen* domain category, a noteworthy achievement for models of its kind.

End2End Architecture The Flan-T5 model outperformed other models, including the baseline in the end-to-end architecture, achieving the highest scores in both Bleu and Comet for the *all* and *unseen* domains. However, the baseline model delivered superior results in the Meteor category. Among the fine-tuned models, GPT-2 ranked the lowest, followed by the BART model, with Flan-T5 leading. While comparing prompt-based models in the collective domains, the GPT-4 model excelled in Bleu, Meteor, and Comet metrics, followed by the Cohere model, GPT-3.5, and finally Mistral7B.

Pipeline vs. End2End We evaluated the results of the surface realization stage in both the 5-stage and 3-stage pipeline architectures, as well as the End-to-End architecture as shown in Table 1 and 2. The End-to-End method uniformly outperformed the pipeline setups, except in the baseline, where it emerged as the overall best in both the *all* and *seen* domains across the models and architectures. However, the performance gap between the End-to-End and the 3-stage pipeline was smaller than the gap between the End-to-End and the 5-stage pipeline when using GPT-3.5 and GPT-4 as benchmarks. This suggests that while the End-to-End approach generally yields superior results, the more pronounced performance decline in the 5-stage pipeline may be due to error cascading, indicating that reducing the number of pipeline stages could lead to better text generation.

Human Evaluation Two of our authors served as human evaluators for four top models: Flan-T5 end-to-end, GPT-4 end-to-end, Flan-T5 surface re-

Domains	**Fluency**	**Semantic Adequacy**	**Omission**	**Addition**	**Incorrect Number**	**Incorrect Entity**	**Average**
flan-t5-sr	6.30^{C}	6.19^{C}	0.48	$\underline{0.73}$	0.91	0.62	0.68
flan-t5-end2end	6.68^{B}	$\underline{6.86}^{B}$	0.86	**0.98**	**1.00**	0.83	0.92
gpt4-struct2SR	$\mathbf{6.83}^{A}$	$6.85^{A\,B}$	$\underline{0.93}$	**0.98**	$\underline{0.99}$	$\underline{0.95}$	$\underline{0.96}$
gpt4-end2end	$\underline{6.82}^{A}$	$\mathbf{6.94}^{A\,B}$	**0.97**	**0.98**	**1.00**	**0.96**	**0.98**

Table 3: Results of the human evaluation and semantic Accuracy evaluation using GPT-4o. Ranking was determined by pair-wise Mann-Whitney statistical tests with $p < 0.05$.

alization (flan-t5-sr) stage result, and the GPT-4 Struct2SR result, using 100 balanced samples. The evaluators were not informed about which models generated the samples to ensure an impartial assessment. For proper comparison, they rated fluency and semantic adequacy on a 1-7 Likert scale just as in Ferreira et al. (2019). Semantic errors such as omissions, additions, and incorrect numbers and entities were identified using GPT-4o[2] on 120 samples each. Results are presented in Table 3.

GPT-4 Struct2SR achieved the highest fluency rating, while GPT-4 end-to-end scored highest in semantic adequacy. The Flan-T5-SR model had the most semantic errors and the lowest semantic accuracy, while GPT-4 end-to-end had the lowest errors.

The Mann-Whitney test (Mann and Whitney, 1947) showed significant differences in fluency and semantics between most model pairs, except between some GPT-4's and Flan-T5 end2end comparisons. Overall, GPT-4 models performed better or comparably to the Flan-T5 end-to-end model, with the Flan-T5-SR model the least performing.

5 Conclusion

This study demonstrates that PLMs tend to outperform the baseline, particularly in unseen domains. The baseline in this case is a vanilla transformer model that was trained from scratch on the dataset. It also corroborates existing research which shows that fine-tuned models generally outperform prompt-based models in zero-shot scenarios and exhibit comparable trends in few-shot learning (Yuan and Färber, 2023; Axelsson and Skantze, 2023). However, prompt-based models exhibited fewer errors in numbers and entities, as well as fewer additions and omissions compared to the fine-tuned models. This confirms previous research on fine-tuned models in pipeline architecture generating imaginary numbers (Cunha et al., 2024). Moreover, the performance of prompt-based models does not decrease in unseen domains, as shown in previous studies and for fine-tuned models.

In the comparison between pipeline and end-to-end approaches, our study shows that end-to-end architecture yielded the best results in both automatic and human evaluations. In the comparison between pipeline approaches, our analysis indicates that a pipeline architecture with fewer stages produces better outcomes than a full-stage pipeline.

In a combination between model designs, we speculate that fine-tuned models under a 3-stage architecture could outperform prompt-based models. Additionally, using fine-tuned models for ordering and structuring, and a prompt-based model for surface realization (i.e., model hybridization) could yield better results. This is intended to be explored as future work.

Limitations

Prompt engineering is inherently subjective, and the prompts used in this experiment may not be the optimal choices. Additionally, models like GPT-3.5 and GPT-4 are not open source and can produce varying responses to the same prompt, which affects the reproducibility of the evaluation scores.

Ethic Statement

Two members of our research group conducted the evaluations, so ethical approval for human subjects was not required. The publicly accessible data used in this research contains no sensitive information, ensuring compliance with the EU's GDPR. Additionally, since large language models (LLMs) can produce factually incorrect information and we lack access to their training data, we cannot control inherent biases or guarantee the accuracy and impartiality of the generated text.

Acknowledgements

This work was conducted with the financial support of the Science Foundation Ireland Centre for Research Training in Artificial Intelligence under Grant No. 18/CRT/6223. We would also like to extend our gratitude to aiXplain for providing free credits that supported part of this study.

[2] `https://platform.openai.com/docs/models/gpt-4o`

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Agnes Axelsson and Gabriel Skantze. 2023. Using large language models for zero-shot natural language generation from knowledge graphs. *arXiv preprint arXiv:2307.07312*.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Thiago Castro Ferreira, Claire Gardent, Nikolai Ilinykh, Chris van der Lee, Simon Mille, Diego Moussallem, and Anastasia Shimorina. 2020. The 2020 bilingual, bi-directional WebNLG+ shared task: Overview and evaluation results (WebNLG+ 2020). In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 55–76, Dublin, Ireland (Virtual). Association for Computational Linguistics.

Thiago Castro Ferreira, Diego Moussallem, Emiel Krahmer, and Sander Wubben. 2018. Enriching the WebNLG corpus. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 171–176, Tilburg University, The Netherlands. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Rossana Cunha, Osuji Chinonso, João Campos, Brian Timoney, Brian Davis, Fabio Cozman, Adriana Pagano, and Thiago Castro Ferreira. 2024. Imaginary numbers! evaluating numerical referring expressions by neural end-to-end surface realization systems. In *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pages 73–81.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, pages 552–562.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

Qipeng Guo, Zhijing Jin, Ning Dai, Xipeng Qiu, Xiangyang Xue, David Wipf, and Zheng Zhang. 2020. $\mathcal{P}^2$: A plan-and-pretrain approach for knowledge graph-to-text generation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 100–106, Dublin, Ireland (Virtual). Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Zdeněk Kasner and Ondřej Dušek. 2020. Data-to-text generation with iterative text editing. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 60–67, Dublin, Ireland. Association for Computational Linguistics.

Zdeněk Kasner and Ondrej Dusek. 2022. Neural pipeline for zero-shot data-to-text generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3914–3932, Dublin, Ireland. Association for Computational Linguistics.

Zdeněk Kasner and Ondřej Dušek. 2024. Beyond reference-based metrics: Analyzing behaviors of open llms on data-to-text generation. *arXiv preprint arXiv:2401.10186*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2267–2277, Minneapolis, Minnesota. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. Comet: A neural framework for mt evaluation. *arXiv preprint arXiv:2009.09025*.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems.

Leonardo FR Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. 2020. Investigating pretrained language models for graph-to-text generation. *arXiv preprint arXiv:2007.08426*.

Shreyas Sharma, Lucas Pavanelli, Thiago Castro Ferreira, Mohamed Al-Badrashiny, and Hassan Sawaf. 2024. aixplain sdk: A high-level and standardized toolkit for ai assets. In *Proceedings of the 17th International Natural Language Generation Conference (INLG)*, Tokyo, Japan. To appear.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, et al. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *arXiv preprint arXiv:2303.10420*.

Shuzhou Yuan and Michael Färber. 2023. Evaluating generative models for graph-to-text generation. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pages 1256–1264.

Chao Zhao, Marilyn Walker, and Snigdha Chaturvedi. 2020. Bridging the structural gap between encoding and decoding for data-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2481–2491, Online. Association for Computational Linguistics.

Ahmet Üstün, Viraat Aryabumi, Zheng-Xin Yong, Wei-Yin Ko, Daniel D'souza, Gbemileke Onilude, Neel Bhandari, Shivalika Singh, Hui-Lee Ooi, Amr Kayid, Freddie Vargus, Phil Blunsom, Shayne Longpre, Niklas Muennighoff, Marzieh Fadaee, Julia Kreutzer, and Sara Hooker. 2024. Aya model: An instruction finetuned open-access multilingual language model.

A Appendix

A.1 Pipeline Neural Architecture Modules

Ordering The ordering stage organizes information derived from randomly generated triples in the dataset. Drawing from methods described in previous study, the linearized triples are processed through the model to generate sequences using predicates. This ensures a logical sequence of information, with predicates crucially arranging the triples. The resulting ordered predicates are then re-associated with their corresponding objects and subjects, ensuring a seamless information flow. An example of this process is illustrated in Figure 3, where input triples (shown in Figure 1) are inputted into the neural model to determine the ordering based on predicates. These ordered triples are then used by the mapping modules to prepare inputs for the next pipeline stage.

Structuring In the structuring stage, the text is organized into paragraphs that may consist of single or multiple sentences, each carrying sequential information. This stage crafts sentence realization from the content of the ordered triples, with predicates guiding the structuring process. The outputs are mapped to their respective subjects and objects to enhance text coherence and readability, as illustrated in Figure 3.

Lexicalization The provided text, as shown in Figure 3, represents the output of this process, featuring structured information denoted by placeholders like `ENTITY-1, ENTITY-2`, etc., representing entities such as proper nouns, dates, places, and numbers. Each line describes an action or attribute associated with these entities, including details like the determiner (DT) and verb phrase (VP) such as the aspect, tense, voice, person, and number. The mapping process then reverts these entity representations to their original forms for further processing.

Referring Expression Generation REG ensures consistent and clear references to entities within the text by using appropriate nouns and pronouns like "country", "he", "she", "her", and "it" instead of repeatedly mentioning proper nouns. This technique enhances readability and coherence. The REG output in Figure 3 illustrates this process, contributing to a smoother narrative flow.

Surface Realization The surface realization stage is the culmination of the pipeline, where the ordered, structured, and lexically enhanced text, along with suitable referring expressions, is finalized. Displayed in Figure 3, this stage applies handcrafted rules to adjust verb phrases and refine the text, ensuring grammatical accuracy, coherence, and stylistic integrity. This final step effectively transforms structured data representations into polished, comprehensible natural language text, ready for presentation.

A.2 Data Processing

Preprocessing: To enhance clarity and prevent misinterpretations in the fine-tuned models, we substituted the '<' and '>' tags with '[' and ']', respectively. This change was made after observing that the original tags often led the models to generate hallucinated content.
Post processing: We implemented a thorough cleaning process using Python's regular expression package, applying specific patterns to filter out over-generations in our results.

Input Triples:
[TRIPLE] Bananaman broadcastedBy BBC [/TRIPLE] [TRIPLE] Bananaman creator John_Geering [/TRIPLE] [TRIPLE] Bananaman firstAired "1983-10-03" [/TRIPLE] [TRIPLE] Bananaman lastAired "1986-04-15" [/TRIPLE] [TRIPLE] Bananaman starring Tim_Brooke-Taylor [/TRIPLE]

Ordering Output:
broadcastedBy firstAired lastAired creator starring

Input Triples after mapping:
[TRIPLE] Bananaman broadcastedBy BBC [/TRIPLE] [TRIPLE] Bananaman firstAired "1983-10-03" [/TRIPLE] [TRIPLE] Bananaman lastAired "1986-04-15" [/TRIPLE] [TRIPLE] Bananaman creator John_Geering [/TRIPLE] [TRIPLE] Bananaman starring Tim_Brooke-Taylor [/TRIPLE]

Structuring Output:
[SNT] broadcastedBy firstAired lastAired [/SNT] [SNT] creator starring [/SNT]

Input Triples after mapping:
[SNT] [TRIPLE] Bananaman broadcastedBy BBC [/TRIPLE] [TRIPLE] Bananaman firstAired "1983-10-03" [/TRIPLE] [TRIPLE] Bananaman lastAired "1986-04-15" [/TRIPLE] [/SNT] [SNT] [TRIPLE] Bananaman creator John_Geering [/TRIPLE] [TRIPLE] Bananaman starring Tim_Brooke-Taylor [/TRIPLE] [/SNT]

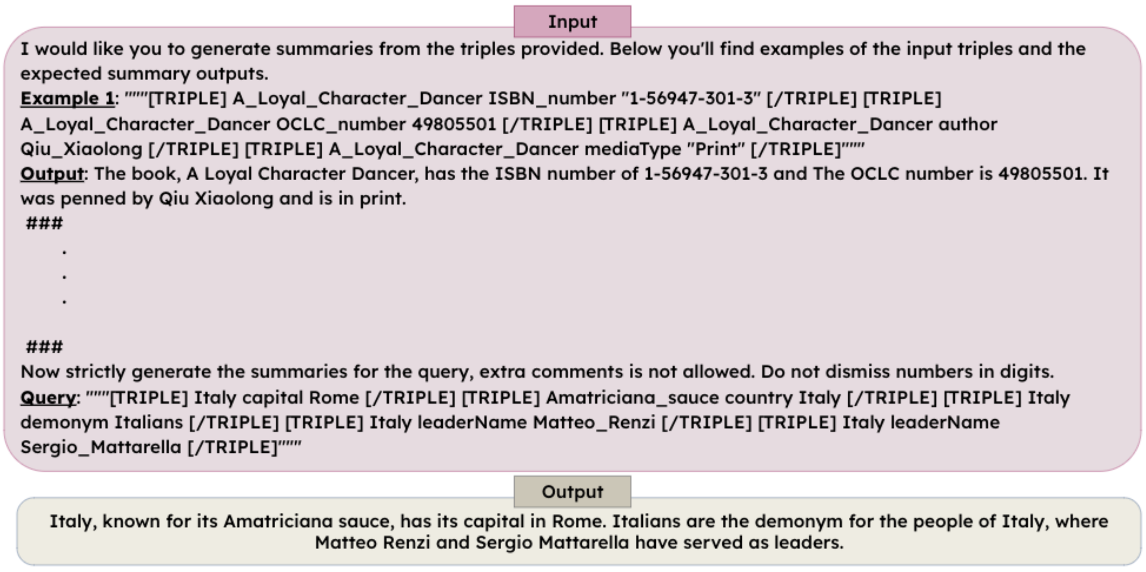
Lexicalization Output:
ENTITY-1 VP[aspect=simple, tense=past, voice=passive, person=null, number=singular] show on ENTITY-2, first airing on ENTITY-3 and DT[form=defined] the final broadcast VP[aspect=progressive, tense=present, voice=active, person=null, number=null] be ENTITY-4. ENTITY-1 VP[aspect=simple, tense=past, voice=passive, person=null, number=singular] create by ENTITY-5 and VP[aspect=simple, tense=past, voice=active, person=null, number=null] star ENTITY-6.

Output after mapping:
Bananaman VP[aspect=simple, tense=past, voice=passive, person=null, number=singular] show on BBC, first airing on "1983-10-03" and DT[form=defined] the final broadcast VP[aspect=progressive, tense=present, voice=active, person=null, number=null] be "1986-04-15". Bananaman VP[aspect=simple, tense=past, voice=passive, person=null, number=singular] create by John_Geering and VP[aspect=simple, tense=past, voice=active, person=null, number=null] star Tim_Brooke-Taylor.

REG Output:
Bananaman VP[aspect=simple, tense=past, voice=passive, person=null, number=singular] show on the BBC, first airing on October 3, 1983 and DT[form=defined] the final broadcast VP[aspect=progressive, tense=present, voice=active, person=null, number=null] be April 15, 1986. It VP[aspect=simple, tense=past, voice=passive, person=null, number=singular] create by John Geering and VP[aspect=simple, tense=past, voice=active, person=null, number=null] star Tim Brooke Taylor.

Surface Realizer Output:
Bananaman was shown on the BBC, first airing on October 3, 1983 and the final broadcast being April 15, 1986. It was created by John Geering and starred Tim Brooke Taylor.

Figure 3: 5 Stage Pipeline Neural Architecture Outputs


Input

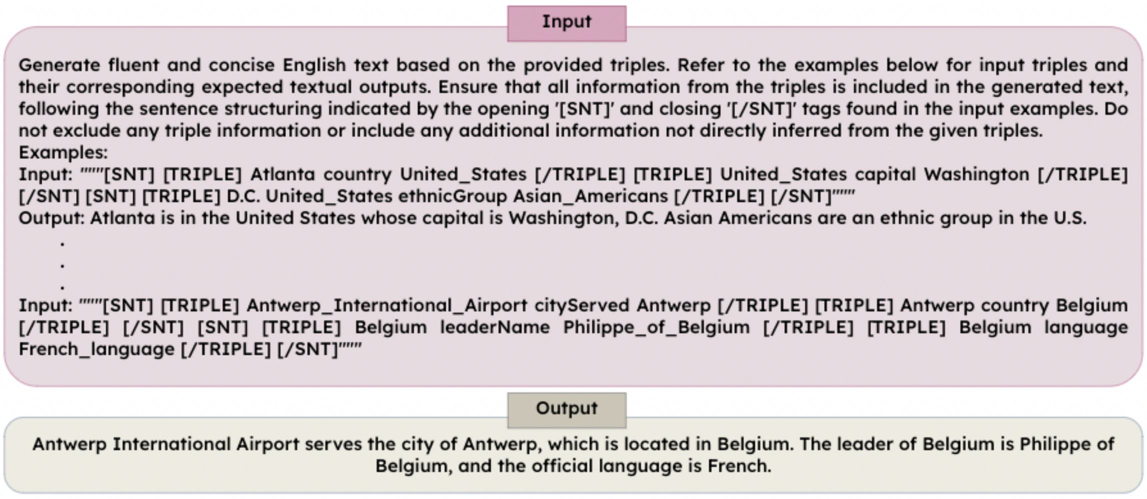
I would like you to generate summaries from the triples provided. Below you'll find examples of the input triples and the expected summary outputs.
Example 1: """[TRIPLE] A_Loyal_Character_Dancer ISBN_number "1-56947-301-3" [/TRIPLE] [TRIPLE] A_Loyal_Character_Dancer OCLC_number 49805501 [/TRIPLE] [TRIPLE] A_Loyal_Character_Dancer author Qiu_Xiaolong [/TRIPLE] [TRIPLE] A_Loyal_Character_Dancer mediaType "Print" [/TRIPLE]"""
Output: The book, A Loyal Character Dancer, has the ISBN number of 1-56947-301-3 and The OCLC number is 49805501. It was penned by Qiu Xiaolong and is in print.
###
.
.
.

###
Now strictly generate the summaries for the query, extra comments is not allowed. Do not dismiss numbers in digits.
Query: """[TRIPLE] Italy capital Rome [/TRIPLE] [TRIPLE] Amatriciana_sauce country Italy [/TRIPLE] [TRIPLE] Italy demonym Italians [/TRIPLE] [TRIPLE] Italy leaderName Matteo_Renzi [/TRIPLE] [TRIPLE] Italy leaderName Sergio_Mattarella [/TRIPLE]"""

Output

Italy, known for its Amatriciana sauce, has its capital in Rome. Italians are the demonym for the people of Italy, where Matteo Renzi and Sergio Mattarella have served as leaders.

Figure 4: A GPT-(3.5 & 4) prompt for end2end surface realization.


Input

Generate fluent and concise English text based on the provided triples. Refer to the examples below for input triples and their corresponding expected textual outputs. Ensure that all information from the triples is included in the generated text, following the sentence structuring indicated by the opening '[SNT]' and closing '[/SNT]' tags found in the input examples. Do not exclude any triple information or include any additional information not directly inferred from the given triples.
Examples:
Input: """[SNT] [TRIPLE] Atlanta country United_States [/TRIPLE] [TRIPLE] United_States capital Washington [/TRIPLE] [/SNT] [SNT] [TRIPLE] D.C. United_States ethnicGroup Asian_Americans [/TRIPLE] [/SNT]"""
Output: Atlanta is in the United States whose capital is Washington, D.C. Asian Americans are an ethnic group in the U.S.

.
.
.

Input: """[SNT] [TRIPLE] Antwerp_International_Airport cityServed Antwerp [/TRIPLE] [TRIPLE] Antwerp country Belgium [/TRIPLE] [/SNT] [SNT] [TRIPLE] Belgium leaderName Philippe_of_Belgium [/TRIPLE] [TRIPLE] Belgium language French_language [/TRIPLE] [/SNT]"""

Output

Antwerp International Airport serves the city of Antwerp, which is located in Belgium. The leader of Belgium is Philippe of Belgium, and the official language is French.

Figure 5: A GPT-(3.5 & 4) 3-stage pipeline prompt for the final surface realization stage.