LongWeave: A Long-Form Generation Benchmark Bridging Real-World Relevance and Verifiability

Anonymous ACL submission

Abstract

Generating long, informative, and factual outputs remains a major challenge for Large Language Models (LLMs). Existing benchmarks for long-form generation typically assess real-world queries with hard-to-verify metrics or use synthetic setups that ease evaluation but overlook real-world intricacies. In this paper, we introduce **LongWeave**, which balance real-world and verifiable assessment with Target-Anchored Evaluation (TAE). TAE constructs tasks by first defining verifiable targets within real-world scenarios, then systematically generating corresponding queries, textual materials, and anchors based on these targets. This ensures that tasks are both realistic and objectively assessable, enabling rigorous assessment of model capabilities in meeting complex real-world constraints. LongWeave supports customizable input/output lengths (up to 64K/8K tokens) across seven distinct tasks. Evaluation on 23 LLMs show that even state-of-the-art models encounter significant challenges in long-form generation as real-world complexity and output length increase. Dataset will be publicly available.

1 Introduction

Large Language Models (LLMs) have significantly enhanced their capabilities to process long inputs (Yang et al., 2024a, 2025; Grattafiori et al., 2024; Team et al., 2023) through architectural design (Dao, 2024) and data engineering (Fu et al., 2024; Gao et al., 2024). However, achieving robust long-sequence generation remains highly challenging (Que et al., 2024; Bai et al., 2024b). Several research efforts have attempted to optimize LLMs for long-form output generation (Pham et al., 2024; Bai et al., 2024b; Yang et al., 2024b; Xiong et al., 2025), which enables the model to generate outputs up to 8,192 tokens in length. However, the generated content often lacks adequate informativeness, comprehensiveness, and factuality (Qi et al., 2024; Pradeep et al., 2024; Song et al., 2024). The inherent complexity of long-form sequences further complicates accurate assessment of these qualities, highlighting the necessity for more reliable evaluation benchmarks.

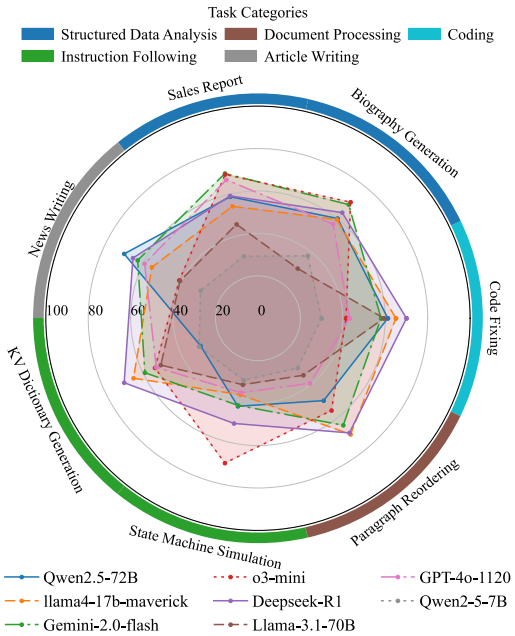

Figure 1: Radar chart of different models across 7 tasks.

Long-form generation with real-world queries is typically evaluated using similarity metrics (e.g., α-nDCG, Self-BLEU) or LLM-as-judge methods (Bai et al., 2024b). While straightforward to implement, direct evaluation struggles with the inherent long-sequence complexity. To address this, another line of work breaks long-text evaluation into a set of verifiable sub-tasks, which can include factual claims (e.g., a statement like "the Earth orbits the Sun") or aspects (e.g., completeness, logical consistency). Checklists are constructed through expert-curated guidelines (Tan et al., 2024; Que et al., 2024) or automated methods leveraging LLMs to extract claims from outputs for factual verification via search engines (Song et al.,

Table 1: Comparison between long-context benchmarks. **'Open-ended'** indicates whether the task allows for diverse, creative responses. **'Deterministic'** means the task produces step-by-step, logically structured outputs. Our Target-Anchored Evaluation synthetically constructs tasks for real-world relevance. Color highlights indicate strengths (green) or challenges (orange). The length refers to the number of tokens under the cl100k tokenizer.

Benchmark	Input Len	Output Len	Open-ended	Deterministic	Evaluator
Benchmarks for Long Input					
LongBench (Bai et al., 2024a)	∼16k	∼100	✓	✓	Similarity
RULER (Hsieh et al., 2024)	∼128k	∼100	×	✓	Rules
HELMET (Yen et al., 2025)	∼128k	∼100	✓	✓	LLM-as-judge
InfiniteBench (Zhang et al., 2024)	Infinite	∼100	×	✓	Rules
Benchmarks for Long Generation					
LongWriter-Bench (Bai et al., 2024b)	∼100	∼5k	✓	×	LLM-as-judge
LongGenBench[1] (Liu et al., 2024c)	∼1k	∼4k	×	✓	Similarity
LongGenBench[2] (Wu et al., 2025)	∼100	∼8k	✓	✓	LLM-as-judge
Hello Bench (Que et al., 2024)	∼300	∼8k	✓	×	LLM-as-judge
LongProc (Ye et al., 2025)	∼32k	∼8k	×	✓	Rules
LongWeave	**64k**	**8k**	✓	✓	**Anchor-Target Pairs**

2024; Wei et al., 2024; Samarinas et al., 2025) or fixed databases (Samarinas et al., 2025). A critical challenge lies in optimizing the degree of specificity scope: overly broad checklists produce vague claims that hinder verification, while overly detailed ones tend to over-complicate verification processes by attempting to cover all corner cases.

To enhance verifiability, some approaches use synthetic data rather than real-world data-for instance, combining short questions from datasets like MMLU (Liu et al., 2024c) into longer ones, and then checking each segment individually. Other benchmarks conduct procedural simulation or utilize objective question-answering (QA) tasks where fixed answers are associated with precise constraints to limit the response scope (Wu et al., 2025; Ye et al., 2025). Though these methods simplify verification, they generally sacrifice realism in real-world scenarios.

To bridge real-world relevance with verifiability, we implement decomposition at the verification stage through a new Target-Anchored Evaluation (TAE) mechanism. Rather than extracting checklists from raw materials which is error-prone and hard to control, TAE reverses the test construction process: it begins with predefined verifiable checklist objectives (Targets) grounded in real-world tasks, then synthesizes corresponding inference samples (including Anchors and materials). The Anchor acts as a constrained input that causally guides models toward generating the predefined Target, enabling measurable verification and evaluation. Each Anchor-Target (AT) pair in TAE maintains a deterministic one-to-one relationship under structurally defined rules, systematically linked to source materials. TAE contains a series of AT pairs, where each pair is linked to the corresponding material. These pairs can take various forms, such as a question (A) and answer (T) in QA tasks, or a triplet (A) and corresponding sentence (T) in knowledge-to-text generation, as discussed in subsection 2.3.

Based on TAE, we introduce **LongWeave**, a new benchmark evaluating five challenge scenarios of long-form generation through seven real-world relevant tasks (Figure 1). LongWeave supports customizable input lengths (up to 64K tokens) and output lengths of 1K, 2K, 4K, and 8K tokens, with adjustable difficulty settings for each task as detailed in Table 1.

Our evaluation of 23 LLMs on LongWeave reveals critical limitations in long-form generation: even top models (DeepSeek-R1) reach a performance ceiling of 54.56%, with performance declining for 8K-token outputs (Figure 1). Furthermore, models exhibit input-output disconnect, while supporting inputs up to 64K tokens, they fail to effectively synthesize inputs into coherent long-form responses. Expanding input context windows, e.g., to 1M tokens, does not resolve long-generation issues but may degrade performance. Lastly, reasoning-oriented LLMs consistently outperform general counterparts, producing more concise and accurate outputs in complex, constrained long-form generation scenarios. Our main contributions are:

- We introduce the long-form generation benchmark **LongWeave**, with TAE that bridges real-world relevance with verifiability.
- We design seven tasks, with long input sizes (up to 64K tokens), long output requirements (1-8K), and varying difficulty levels.

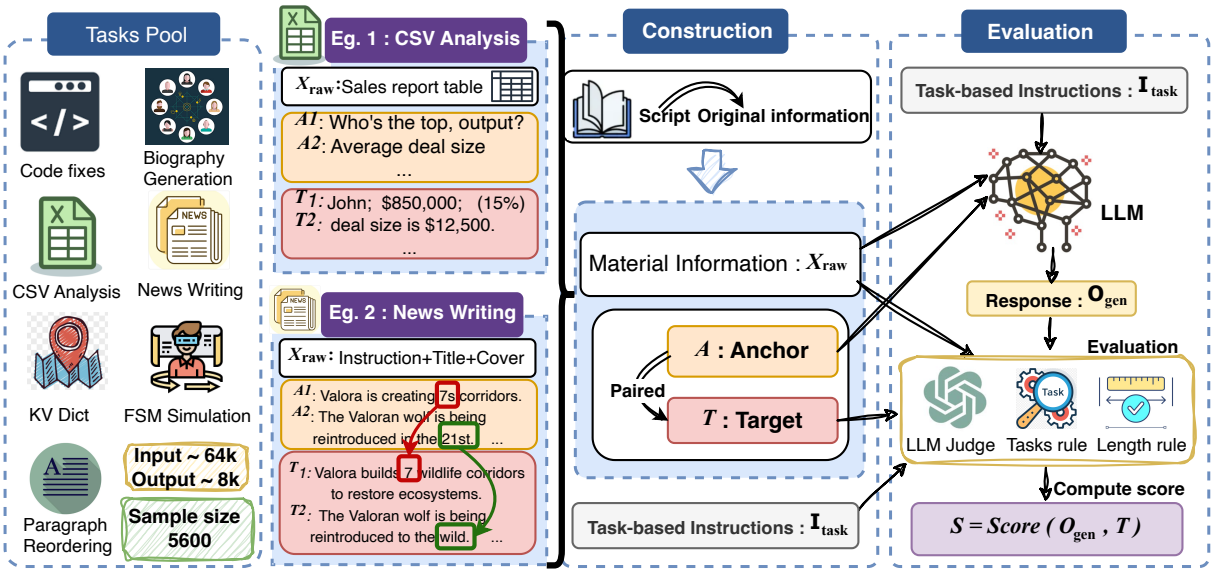

Figure 2: An illustration of evaluation pipeline

- Evaluation of 23 LLMs reveals critical limitations and highlights future directions in long-form generation and evaluation.

2 The LongWeave Benchmark

In this section, we first introduce the overall pipeline of LongWeave, followed by a detailed formulation of our Target-Anchored Evaluation and a description of the individual tasks.

2.1 Pipeline of LongWeave

As shown in Figure 2, the LongWeave pipeline consists of three steps: Construction, Evaluation, and Scoring. **In Construction**, task-specific attributes are systematically sampled through deterministic rule-based algorithms to generate perfectly aligned triples: (1) *raw material*, (2) *anchor*, and (3) *target*. The LLM processes the material and anchor during Evaluation to produce a response that meets task constraints. Finally, in Scoring, the output is compared to the target using a scoring function that aggregates metrics like accuracy, coherence, and style to assess the model's performance.

2.2 Target-Anchored Evaluation

We formulate long-form constrained generation as the task where a LLM, denoted as $\mathcal{L}$, must produce an output sequence O_{gen}. The input consists of a potentially lengthy raw material X_{raw}, and task-specific instruction I_{task}, which specifies requirements for the target output length $\mid O_{\text{gen}} \mid$, content accuracy, structural formatting, and logical coherence. The generation process is modeled as:

$$O_{\text{gen}} = \mathcal{L}(X_{\text{raw}}, I_{\text{task}}) \tag{1}$$

The primary challenge lies in ensuring O_{gen} adheres to all facets of I_{task}, especially as the input and output lengths increase, and as I_{task} becomes more complex.

Algorithm 1: LongWeave Construction

Input: Generator f_{gen}, Attribute space Θ
Output: Material X_{raw}, Anchor A, Target T

Preparation Phase:
1. Sample attributes $\boldsymbol{\theta} \sim \Theta$
2. Generate aligned components via the generator:
   $(X_{\text{raw}}, A, T) \leftarrow f_{\text{gen}}(\boldsymbol{\theta})$
   `//Material, Anchor, and Target are jointly derived and semantically linked`

To ensure a realistic yet verifiable setup, we propose Target-Anchored Evaluation (TAE). TAE strategically constructs raw material X_{raw} alongside an Anchor–Target pair (A, T) for assessment, see Algorithm 1. The input instruction incorporates both the material and the anchor, while the output is evaluated based on whether it correctly reflects the target T associated with the anchor A. Specifically, the generation process is modeled as:

$$O_{\text{gen}} = \mathcal{L}(X_{\text{raw}}, I_{\text{task}}, \mathcal{A}), \tag{2}$$

then the quality S of O_{gen} is quantified by a task-specific scoring function $Score$:

$$S = \text{Score}(O_{gen}, T) \tag{3}$$

Table 2: LongWeave Tasks: A summary of the tasks, outlining their names, abbreviations, core challenges, important configuration settings, and evaluation metrics. Metric types are color-coded as described in the table's legend. Purple represents rule-based metrics. Red refers to anchor-target metrics. Blue indicates length scores.

Task Name	Abbrev.	Challenge	Configuration	Metrics
Code Fixing with Flake8 Compliance	CF	Coding	violation_prob = 0.85 **error_lines** $\propto$ **gen_len**	Runnability Style score length score
KG to Text Biography Generation	BioG	Structured Data Analysis	**triple_count** $\propto$ **gen_len**	Coverage Rate.
CSV Sales Report Analysis	SR	Structured Data Analysis	**record_count** $\propto$ **gen_len** **target_count** $\propto$ **gen_len**	Coverage Rate Correctness Rate
AP Style News Writing	NW	Article Writing	**fact_counts** $\propto$ **gen_len** ap_stylebook_rules	Coverage Rate Style Score
KV Dictionary Generation	KVG	Instruction Following	**entry_count** $\propto$ **gen_len** key_length = 32 value_length = 32	Existence Score Length score Position score
State Machine Simulation	SMS	Instruction Following	num_states = 3 input_size = 3 output_size = 3 **step_length** $\propto$ **gen_len**	Step Match Ratio
Paragraph Reordering	PR	Document Processing	**para_length** $\propto$ **gen_len**	Kendall's Tau.

LongWeave evaluates LLMs by measuring S across diverse tasks that vary in input/output lengths and task complexity.

2.3 Tasks

We now introduce each of the tasks, where the Anchor–Target pair takes different forms depending on the task—for example, a negative–positive pair in code fixing and AP style news writing, or a question–answer pair in sales report analysis.

Code Fixing. This task requires LLMs to *fix Python code with Flake8 style* violations (line length, indentation) while ensuring the code remains runnable. We design the *code polluter* to inject Flake8 violations into randomly generated runnable Python scripts (M, T), forming a polluted code (A). The LLM is prompted to fix the code.

KG to Text Biography Generation. This task evaluates LLMs' ability to generate coherent and factual biographies based on given knowledge graph triples. The designed *knowledge graph generator* creates a large set of task relationships around a central character, then extracts triples (subject-predicate-object) (A) and corresponding sentences (T) starting from the nearest nodes. The evaluated model needs to incorporate all triples into a fluent narrative within the specified word count. The target is a rule-based natural language statement derived from these triples. The model is evaluated on its ability to accurately integrate all triples into the generated text, with penalties for missing or fabricated information.

CSV Sales Report Analysis. This task evaluates LLMs' ability to generate a sales report based on a CSV of sales data, including answering specific questions. We designed an Excel sales report generator that creates the file (M), while generating biased natural language questions (A) and answers (T). The model analyzes the data, presents insights, and answers predefined questions. Evaluation is based on the answer and correctness rates.

AP Style News Writing. This task evaluates LLMs' ability to write a news article following the Associated Press Stylebook (AP Style) (Goldstein, 1998), based on a central idea (M) and a series of fact statements generated by GPT-4o. The central idea is generated for the random scenario, and the LLM generates *fact statements that violate AP Style rules (A)* based on the central idea. The corresponding corrected statements (T) are the targets. The model is to incorporate the central idea with all required facts while adhering to AP Style rules on punctuation, capitalization, and other conventions. The final score is based on both content recall and style score of each target.

KV Dictionary Generation. This task, the inverse of KV Retrieval in (Hsieh et al., 2024), evaluates LLMs' ability to generate a dictionary string with

a target key-value pair placed *at the correct index*, following strict formatting rules (e.g., keys in uppercase with underscores, values in lowercase with numbers). The model is evaluated w.r.t. accuracy in placement and valid dictionary formatting.

State Machine Simulation. This task involves simulating state transitions based on a finite state machine (FSM) (Lee and Yannakakis, 1996) and processing an input string step-by-step. The model generates output sequences based on FSM rules. We use an FSM validation script to verify the output against the correct state transitions (M) and signals (A). The anchors are the input string and the states generated by the model, while the targets are the correct FSM sequences (T) generated by the FSM script. Models are evaluated based on match ratio and accuracy in simulating all steps without errors.

Paragraph Ordering. This task requires LLMs to reorder shuffled paragraphs (A) into the collected coherent sequence(T). The material consists of randomly sampled paragraphs, with the anchors being the shuffled order and the target being the correct sequence. Evaluation uses Kendall's Tau to measure the consistency of the predicted order (Liu et al., 2020; Shen and Baldwin, 2021).

2.4 Data Construction

The majority of the data within LongWeave is synthetically generated to ensure precise control over task parameters and facilitate the Target-Anchored Evaluation (TAE). The algorithmic approach allows for the systematic creation of raw materials across most tasks, such as code snippets for Code Fixing, triples for KG-to-Text, CSV data for Sales Reports, KV pairs for Dictionary Generation, state machine definitions for Simulation, and paragraph sets for Reordering. In contrast, the AP Style News Writing task employs a hybrid data generation: an LLM creates a news theme, ten AP Stylebook rules are chosen, and factual statements (material) are subsequently produced. To enable objective verification, some statements intentionally violate AP rules (forming the Anchors), while their correct integration serves as the Targets.

2.5 Input Length Statistic

Generative tasks with long input contexts are critical yet underexplored. To reduce hallucinations, users often provide extensive context for generating complex outputs. Unlike prior benchmarks capped at 1k tokens, LongWeave supports up to 64k-token inputs, enabling evaluation in real-world scenarios like structured file analysis and document processing. We provide the input length distribution of LongWeave in fig. 3.

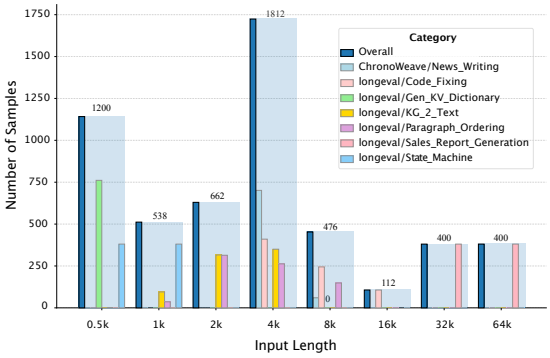

Figure 3: Input length distribution of LongWeave

2.6 Evaluation Metrics

LongWeave evaluates model performance across three main categories: *anchor-task metrics, length score, and rule-based metrics*. Each task's composite score is derived from the harmonic mean of its sub-metrics to prevent a model from having a particularly poor performance in one area while still achieving a high overall score.

Anchor-Task metrics use LLMs as judges to verify whether targets, corresponding to defined anchors, are accurately reflected in the model's output. These include style, factual coverage, and question answering. The *Style Score* measures adherence to Flake8 standards, penalizing unresolved violations. The *Factual Coverage Rate* tracks the proportion of knowledge graph triples in the text, while the *Answer Coverage Rate* measures the proportion of answered analytical questions. The *Correctness Rate* calculates answer accuracy, and the *Factual Statement Coverage Rate* tracks recall of required factual statements. The *AP Style Score* quantifies adherence to AP Stylebook guidelines.

Length Score is used to test whether the model outputs according to the required length. The implicit length score is applied when truncation occurs after exceeding the length, while the explicit length score is used in CF and KVG, where the length score is treated as a sub-score.

Rule-Based Metrics use tools to evaluate whether the model's response aligns with task instructions and accurately reflects targets corresponding to defined anchors. These include the Runnability score, indicating whether the fixed code compiles correctly; the Target Key Existence and Position Score, measuring if the target key exists and is placed at

the specified index in generated dictionaries; and Kendall's Tau Coefficient, assessing the model's ability to preserve correct order in outputs.

3 Experiments

3.1 Models and Inference Setup

We evaluated a range of LLMs using LongWeave, comprising proprietary and commercial API-accessed models, open-source models, and reasoning models. The *long-generation models* assessed include LongWriter-glm4-9b (GLM et al., 2024; Bai et al., 2024b). The *open-source models* include the Llama-3-series, Llama-4-series (Grattafiori et al., 2024), Phi-4-mini-instruct, Qwen2-5-series (3B, 7B, 14B, 72B) (Yang et al., 2024a), and the newer Qwen3 series (4B, 8B, 14B, 32B). Additionally, we evaluated Deepseek-V3 (Liu et al., 2024a). The *commercial models* include GPT-4o-1120 (Achiam et al., 2023), Gemini-2.0-flash (Team et al., 2023), and Qwen-long. Specialized *reasoning models*, such as o3-mini-2025-01-31 and Deepseek-R1 (Guo et al., 2025), were also included in the evaluation. The open-source model uses VLLM for inference deployment on an A100.

3.2 Task Configurations

LongWeave evaluates LLMs across seven distinct tasks, each with four variants targeting output lengths of 1k, 2k, 4k, and 8k tokens. For each variant, 200 test samples are used, resulting in a total of 5,600 samples per model. We primarily used `Qwen2-5-72B-Instruct` for LLM-as-judge evaluations. To control output length, we adjust the configuration as illustrated by the "gen_len" configurations in 2. Furthermore, LongWeave supports fine-grained control over task difficulty through adjustments to input complexity (e.g., key_length in KVG,), the strictness of constraints (e.g., AP stylebook rules in NW), and structural requirements of the target output (e.g., step_length, para_length).

3.3 Main Results

The results are summarized in Table 3. In the table, we have divided all the models into long-generation models, open-source models, commercial models, and reasoning models. We have listed the average performance across seven tasks at four different input lengths, as well as the overall average performance across all tasks at four lengths.

Existing models struggle in long form generation. Frontier proprietary models demonstrate the best performance. DeepSeek-R1 and Gemini-2.0-flash o3-mini achieve nearly 60% performance at 1k length, but when generating 8k, the performance drops to around 40%. GPT-4o only achieves 42.99% while it tends to generate short responses.

Increasing model scale can improve long generation quality. Llama4-17b-128e achieves the best performance due to having the largest number of parameters. The three smallest models, Phi-4-mini, Qwen-2.5-3B, and Qwen3-4b, all perform below 30%. We visualize the relationship between model size and corresponding performance in Figure 4, where the regression curve shows a positive correlation between the two.

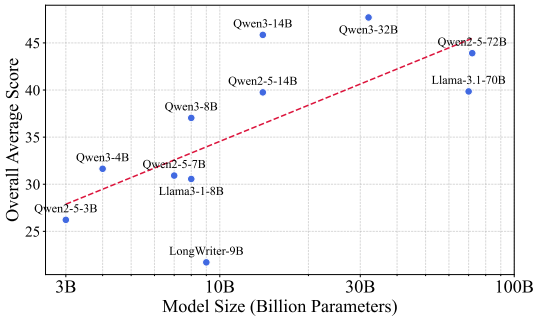

Figure 4: Performance of different model size

3.4 Reasoning Models Perform Better on Long-Sequence Generation Tasks.

We found that large-scale reasoning models perform significantly better on long-sequence generation tasks, as shown in Table 3. However, smaller inference models face challenges with long-sequence tasks. Specifically, models such as DeepSeek Distill-7/32B struggle to follow instructions in tasks involving both long inputs and outputs. During the reasoning process, they often produce repetitive, meaningless English letters, leading to a decline in quality.

Reasoning Models Have Higher Information Density in Long Outputs Reasoning models achieve higher scores despite producing shorter outputs on average. This is likely due to their ability to refine and organize information more effectively in the thinking stage, hence meeting task requirements with less redundancy and off-topic content.

4 Analysis

4.1 Stability of the Benchmark

To assess the stability of the benchmark, we conducted multiple experiments using the Llama-3.1-

Table 3: Model performance summary (task-average and length-average scores). The highest model performance for each task and score is bolded, and for the overall performance, the rank 5 model is bolded.

	Task scores							Length scores				Overall
Model	CF	BioG	SR	NW	KVG	SMS	PR	1k	2k	4k	8k	Avg
LongWriter-glm4-9B	29.67	67.27	18.23	14.62	4.48	3.68	13.99	24.55	23.11	20.69	18.48	21.71
Phi-4-mini-Inst	0.02	69.86	10.50	18.30	3.62	3.25	39.51	23.64	20.27	20.58	18.40	20.72
Llama3-1-8B-Inst	46.76	60.66	13.93	20.29	15.97	3.82	52.46	40.11	34.75	27.13	20.25	30.56
Llama3-1-70B-Inst	58.45	69.36	20.04	24.08	40.35	6.43	60.26	53.58	46.92	33.85	25.07	39.85
Llama4-scout-17B-16e-Inst	33.24	76.38	24.47	28.25	33.32	6.20	67.88	48.65	37.55	37.22	30.72	38.53
Llama4-maverick-17b-128e-Inst	64.63	84.09	22.94	27.96	55.11	9.84	**91.51**	55.81	54.93	50.61	42.12	**50.87**
Qwen2-5-3B-Inst	16.33	66.42	9.82	20.27	20.82	2.85	47.05	30.64	28.16	24.76	21.33	26.22
Qwen2-5-7B-Inst	26.09	73.16	14.91	21.27	19.64	4.94	56.45	38.13	33.77	27.19	24.60	30.92
Qwen2-5-14B-Inst	49.48	80.94	19.33	23.80	22.80	5.72	76.18	47.60	42.97	36.07	32.35	39.75
Qwen2-5-72B-Inst	60.43	84.41	24.48	**31.76**	18.84	13.77	73.67	51.67	48.69	40.99	34.29	43.91
Qwen3-4B	28.36	73.29	17.89	18.19	24.87	11.87	47.02	44.28	35.92	27.18	19.18	31.64
Qwen3-8B	45.92	76.88	18.98	18.90	17.70	13.40	67.50	46.86	40.08	34.06	27.17	37.04
Qwen3-14B	59.10	79.00	21.96	22.12	33.88	18.45	86.43	56.87	49.34	42.28	34.91	45.85
Qwen3-32B	63.44	79.77	24.95	21.46	44.36	16.18	83.71	59.71	52.68	44.57	33.82	47.70
DeepSeek-v3	59.43	80.62	23.25	27.11	33.25	11.47	91.12	56.30	51.61	43.19	35.34	46.61
Qwen-long	35.78	77.65	24.87	26.67	27.78	12.68	78.88	49.50	44.03	39.15	29.78	40.62
GPT-4o-2024-11-20	40.60	82.72	27.20	28.96	42.82	9.16	64.58	56.18	50.30	37.65	25.03	42.29
Gemini-2.0-flash	56.93	88.58	**28.22**	29.91	48.94	13.46	86.68	60.44	56.17	49.20	35.75	**50.39**
DeepSeek-R1-Distill-Qwen-7B	0.00	47.19	4.77	11.76	5.62	2.39	30.50	18.63	13.43	14.21	12.14	14.60
DeepSeek-R1-Distill-Qwen-32B	54.14	66.65	22.25	21.31	14.54	8.86	73.70	45.06	39.59	35.74	29.01	37.35
DeepSeek-R1	**70.10**	86.16	24.62	30.56	**60.14**	19.60	90.73	**63.86**	**59.25**	**52.85**	**42.28**	**54.56**
Qwq-plus-2025-03-05	57.22	80.71	26.66	25.66	40.96	26.10	85.04	62.40	51.82	44.20	37.21	**48.91**
o3-mini-2025-01-31	38.76	**89.30**	28.06	24.21	43.51	**33.06**	78.88	62.06	56.12	43.04	30.66	**47.97**

Table 4: Performance comparison across different tasks under varying sample sizes. Values represent mean performance metrics with standard deviations (format: $mean_{\pm std}$).

Task	**Number of Samples**									
	20	40	60	80	100	120	140	160	180	200
CF	$36.23_{\pm 2.72}$	$35.31_{\pm 2.80}$	$35.88_{\pm 2.50}$	$36.12_{\pm 2.30}$	$37.41_{\pm 0.65}$	$36.65_{\pm 0.55}$	$36.45_{\pm 0.45}$	$36.37_{\pm 0.50}$	$36.82_{\pm 0.32}$	$36.97_{\pm 0.28}$
BioG	$60.42_{\pm 0.16}$	$61.21_{\pm 0.18}$	$61.10_{\pm 0.20}$	$61.15_{\pm 0.18}$	$61.03_{\pm 0.28}$	$61.08_{\pm 0.25}$	$61.20_{\pm 0.23}$	$61.48_{\pm 0.30}$	$61.35_{\pm 0.28}$	$61.14_{\pm 0.41}$
SR	$13.16_{\pm 0.25}$	$12.26_{\pm 0.30}$	$12.60_{\pm 0.28}$	$12.80_{\pm 0.24}$	$13.81_{\pm 0.29}$	$13.50_{\pm 0.22}$	$13.60_{\pm 0.18}$	$13.86_{\pm 0.20}$	$14.05_{\pm 0.16}$	$14.13_{\pm 0.14}$
NW	$20.03_{\pm 0.01}$	$19.06_{\pm 0.10}$	$19.40_{\pm 0.15}$	$19.50_{\pm 0.18}$	$19.75_{\pm 0.20}$	$19.85_{\pm 0.18}$	$19.90_{\pm 0.16}$	$19.37_{\pm 0.30}$	$19.55_{\pm 0.35}$	$19.62_{\pm 0.48}$
KVG	$14.55_{\pm 1.47}$	$13.80_{\pm 1.50}$	$14.00_{\pm 1.40}$	$14.30_{\pm 1.20}$	$15.29_{\pm 0.17}$	$14.60_{\pm 0.30}$	$14.80_{\pm 0.25}$	$14.20_{\pm 0.30}$	$14.95_{\pm 0.45}$	$15.20_{\pm 0.68}$
SMS	$3.19_{\pm 0.07}$	$3.66_{\pm 0.10}$	$3.60_{\pm 0.09}$	$3.63_{\pm 0.08}$	$3.69_{\pm 0.03}$	$3.70_{\pm 0.02}$	$3.74_{\pm 0.02}$	$3.74_{\pm 0.02}$	$3.78_{\pm 0.02}$	$3.81_{\pm 0.01}$
PR	$54.67_{\pm 0.13}$	$58.02_{\pm 0.20}$	$58.10_{\pm 0.15}$	$58.15_{\pm 0.12}$	$57.23_{\pm 0.86}$	$57.90_{\pm 0.80}$	$58.10_{\pm 0.75}$	$58.02_{\pm 0.10}$	$59.20_{\pm 0.15}$	$60.05_{\pm 0.12}$
Overall	$28.92_{\pm 0.30}$	$29.04_{\pm 0.35}$	$29.10_{\pm 0.40}$	$29.30_{\pm 0.35}$	$29.80_{\pm 0.01}$	$29.60_{\pm 0.10}$	$29.80_{\pm 0.15}$	$29.69_{\pm 0.10}$	$30.05_{\pm 0.12}$	$30.13_{\pm 0.11}$

8B model with varying sample sizes (20-200), as shown in Table 4. We found that as the sample size increased, the total score gradually stabilized, and the variance decreased from 0.3 to 0.11. Once the sample size exceeded 100, the results converged within a margin of 0.15. For the official evaluation, we used 200 samples to ensure the stability of the benchmark's total score.

4.2 Efficiency of LLM-as-a-Judge

For the CF, BioG, SR, and NW tasks, we used the Qwen-2.5-72B model as an LLM judge. To ensure accuracy, we evaluated whether a short, clearly defined target sentence appeared in the output, which is simpler than judging whole ultra-long outputs. We further validated this method by testing the Llama-3.1-8B model on 100 samples with different evaluation models, computing scores for four tasks and the total score across seven tasks, as shown in Table 5. he results showed that models such as DeepSeek-v3, GPT-4o, and GPT-03-mini had an overall performance fluctuation variance of 0.45.

Table 5: Evaluation of LLM-as-Judge Stability in Target-Anchor Evaluation using Different Scoring Models

	Scoring Models					
Tasks	DeepSeek V3	o3 Mini	4o 1120	Qwen 2.5 72B	Qwen 2.5 32B	Qwen 2.5 14B
CF	51.64	45.2	40.96	46.76	3.53	0.76
BioG	59.86	59.87	60.13	60.66	58.01	57.88
SR	13.52	13.6	11.7	13.93	13.7	21.71
NW	19.95	10.4	28.88	20.29	10.39	10.67
Total Score	31.75	30.14	31.27	30.56	23.27	24.04

4.3 Output Length Distribution

During inference, we provided the models with required word counts and analyzed the output word lengths, categorizing them into four ranges: below 1k, 1k-2k, 2k-4k, and 4k-8k, as shown in Figure 5. It was observed that, with the exception of the 03-mini, other *reasoning models tend to generate shorter outputs after processing*. In contrast, smaller open-source models tend to generate

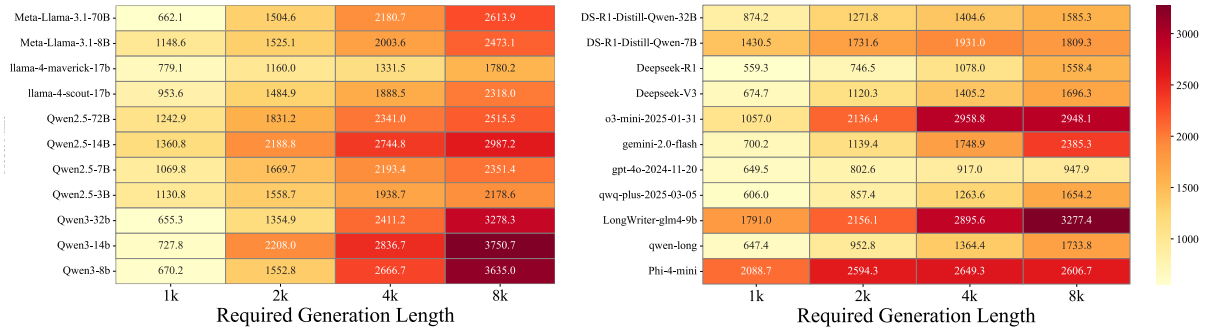

Figure 5: Output length distribution

longer outputs, despite their overall performance scores not being as high, indicating that output quality is not directly correlated with length. Notably, the Qwen-3 series demonstrates better length-following ability compared to the Qwen-2.5.

4.4 Potential Optimization Direction

Performance Degradation when Input Context is Long. As shown in Table 3, output quality deteriorates significantly with long input sequences. This is particularly evident in tasks like generating sales reports and writing AP-style news articles, which involve large datasets and detailed guidelines. However, handling both long inputs and outputs is essential for practical applications. Incorporating more relevant information into the input window can significantly reduce hallucinations. It highlights a critical direction for optimizing long-sequence generation models.

Optimizing long-form output data is possible. Although the structural differences between models are not significant, the Qwen3-32b model outperforms the previous generation's larger Qwen2.5-72b. Smaller models in the Qwen3 series can also be compared with models of the same scale in Qwen2.5. However, while Longwriter can generate outputs up to 10,000 tokens, the generation quality declines.

Increasing the Context Window Does Not Improve Long-Form Generation. We compared the performance of the Qwen2.5-14B and 7B models with a 1M context window version. As shown in Figure 6, *there was little difference in overall scores*. Specifically, long-input models performed better than standard models at 1K, 2K, and 4K lengths but showed decreased performance at 8K when generating ultra-long sequences. This suggests that while long-context understanding and long-form generation share similar mechanisms, factors such as error accumulation can hinder effective long-output generation. Therefore, *optimizing data for long-output tasks specifically is more important than merely increasing the context window*.

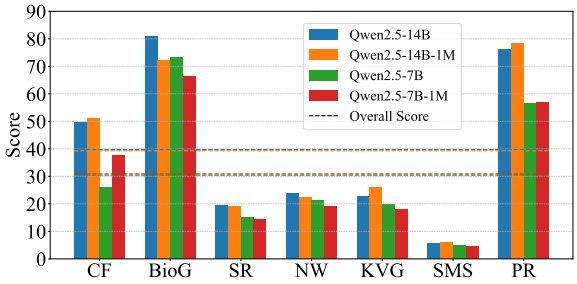

Figure 6: Input length distribution of LongWeave

5 Conclusion

Evaluating long, constrained LLM outputs is challenging. We introduce LongWeave, featuring Target-Anchored Evaluation (TAE) to bridge real-world relevance with objective verifiability. This suite spans seven tasks across five domains with customizable input/output lengths. Our evaluation of 23 LLMs using LongWeave demonstrates that even top models falter for long generations, with performance degrading significantly as length rises; reasoning models, however, navigate these challenges more effectively. LongWeave thereby provides a precise instrument to diagnose these systemic issues and guide the development of truly capable long-form generation.

Limitations

While LongWeave and its Target-Anchored Evaluation (TAE) offer diverse tasks for long-form generation evaluation, certain limitations, particularly concerning computational resources and evaluation efficiency, should be acknowledged:

- **High Computational Cost for Evaluation:** The nature of LongWeave, involving long input materials (up to 64K tokens) and the generation of long outputs (up to 8K tokens), inherently makes evaluating a wide range of models computationally expensive. Each test run consumes significant GPU time and resources, potentially limiting the scale and frequency of benchmarking.

- **Resource-Intensive LLM-as-Judge:** Several tasks within LongWeave rely on a large LLM (e.g., Qwen2-5-72B-Instruct) as the judge for verifying target achievement. Running such a large evaluation model further adds to the computational burden and cost. Developing methods to achieve comparable accuracy with smaller, more efficient evaluator models is a potential.

- **Limited Coverage of Creative Tasks and Stylistic Qualities:** LongWeave can be expanded to better address tasks requiring creativity, imaginative storytelling, or novel content generation. This could involve incorporating metrics for fluency, stylistic diversity, and literary merit, which would allow the benchmark to assess models on more subjective aspects of long-form generation, complementing its current focus on factual accuracy, instruction adherence, and structural correctness.

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*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024a. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Yushi Bai, Jiajie Zhang, Xin Lv, Linzhi Zheng, Siqi Zhu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024b. Longwriter: Unleashing 10,000+ word generation from long context llms. *arXiv preprint arXiv:2408.07055*.

Aydar Bulatov, Yuri Kuratov, Yermek Kapushev, and Mikhail S Burtsev. 2023. Scaling transformer to 1m tokens and beyond with rmt. *arXiv preprint arXiv:2304.11062*.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. Longlora: Efficient fine-tuning of long-context large language models. In *The International Conference on Learning Representations (ICLR)*.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, and Furu Wei. 2023. Longnet: Scaling transformers to 1,000,000,000 tokens. In *Proceedings of the 10th International Conference on Learning Representations*.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128K context. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 14125–14134. PMLR.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. 2024. How to train long-context language models (effectively). *arXiv preprint arXiv:2410.02660*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

Norm Goldstein. 1998. *The Associated Press Stylebook and Libel Manual. Fully Updated and Revised.* ERIC.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

David Lee and Mihalis Yannakakis. 1996. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2024b. Ringattention with blockwise transformers for near-infinite context. In *The Twelfth International Conference on Learning Representations*.

Sennan Liu, Shuang Zeng, and Sujian Li. 2020. Evaluating text coherence at sentence and paragraph levels. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1695–1703, Marseille, France. European Language Resources Association.

Xiang Liu, Peijie Dong, Xuming Hu, and Xiaowen Chu. 2024c. LongGenBench: Long-context generation benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 865–883, Miami, Florida, USA. Association for Computational Linguistics.

Chau Minh Pham, Simeng Sun, and Mohit Iyyer. 2024. Suri: Multi-constraint instruction following for long-form text generation. *Preprint*, arXiv:2406.19371.

Ronak Pradeep, Nandan Thakur, Shivani Upadhyay, Daniel Campos, Nick Craswell, and Jimmy Lin. 2024. Initial nugget evaluation results for the trec 2024 rag track with the autonuggetizer framework. *arXiv preprint arXiv:2411.09607*.

Zehan Qi, Rongwu Xu, Zhijiang Guo, Cunxiang Wang, Hao Zhang, and Wei Xu. 2024. $long^2rag$: Evaluating long-context & long-form retrieval-augmented generation with key point recall. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4852–4872, Miami, Florida, USA. Association for Computational Linguistics.

Shanghaoran Quan, Tianyi Tang, Bowen Yu, An Yang, Dayiheng Liu, Bofei Gao, Jianhong Tu, Yichang Zhang, Jingren Zhou, and Junyang Lin. 2024. Language models can self-lengthen to generate long texts. *arXiv preprint arXiv:2410.23933*.

Haoran Que, Feiyu Duan, Liqun He, Yutao Mou, Wangchunshu Zhou, Jiaheng Liu, Wenge Rong, Zekun Moore Wang, Jian Yang, Ge Zhang, et al. 2024. Hellobench: Evaluating long text generation capabilities of large language models. *arXiv preprint arXiv:2409.16191*.

Chris Samarinas, Alexander Krubner, Alireza Salemi, Youngwoo Kim, and Hamed Zamani. 2025. Beyond factual accuracy: Evaluating coverage of diverse factual information in long-form text generation. *arXiv preprint arXiv:2501.03545*.

Aili Shen and Timothy Baldwin. 2021. A simple yet effective method for sentence ordering. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 154–160, Singapore and Online. Association for Computational Linguistics.

Yixiao Song, Yekyung Kim, and Mohit Iyyer. 2024. VeriScore: Evaluating the factuality of verifiable claims in long-form text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9447–9474, Miami, Florida, USA. Association for Computational Linguistics.

Haochen Tan, Zhijiang Guo, Zhan Shi, Lu Xu, Zhili Liu, Yunlong Feng, Xiaoguang Li, Yasheng Wang, Lifeng Shang, Qun Liu, and Linqi Song. 2024. ProxyQA: An alternative framework for evaluating long-form text generation with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6806–6827, Bangkok, Thailand. Association for Computational Linguistics.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Zixia Hu, Jie Huang, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, and Quoc V Le. 2024. Long-form factuality in large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yuhao Wu, Ming Shan Hee, Zhiqiang Hu, and Roy Ka-Wei Lee. 2025. Longgenbench: Benchmarking long-form generation in long context LLMs. In *The Thirteenth International Conference on Learning Representations*.

Ruibin Xiong, Yimeng Chen, Dmitrii Khizbullin, Mingchen Zhuge, and Jürgen Schmidhuber. 2025. Beyond outlining: Heterogeneous recursive planning for adaptive long-form writing with language models. *arXiv preprint arXiv:2503.08275*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang.

2025. Qwen2.5-1m technical report. *arXiv preprint arXiv:2501.15383*.

Ruihan Yang, Caiqi Zhang, Zhisong Zhang, Xinting Huang, Sen Yang, Nigel Collier, Dong Yu, and Deqing Yang. 2024b. Logu: Long-form generation with uncertainty expressions. *arXiv preprint arXiv:2410.14309*.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. 2025. Longproc: Benchmarking long-context language models on long procedural generation. *arXiv preprint arXiv:2501.05414*.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. 2025. Helmet: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations (ICLR)*.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2025. Long context compression with activation beacon. In *The Thirteenth International Conference on Learning Representations*.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, Bangkok, Thailand. Association for Computational Linguistics.

A Appendix

B Related Work

B.1 Long Input and Output Models

Recent advancements in large language models (LLMs) have significantly improved long-context input processing through techniques such as efficient attention (e.g., Flash Attention (Dao, 2024), Ring Attention (Liu et al., 2024b)), sparse attention methods (e.g., shifted sparse attention in LongLoRA (Chen et al., 2024), dilated attention (Ding et al., 2023)), and memory mechanisms like recurrent caching (Zhang et al., 2025; Bulatov et al., 2023). For long output generation, methods like Suri (Pham et al., 2024) have explored multi-constraint instruction following, while LongWriter (Bai et al., 2024b) introduced AgentWrite to enable ultra-long outputs by decomposing tasks into subtasks. Additionally, the Self-Lengthen framework (Quan et al., 2024) iteratively expands initial outputs, training models to generate longer responses without requiring auxiliary data. These innovations enable LLMs to handle both long inputs and generate extended outputs.

B.2 Long Generation Benchmarks

Long generation benchmarks typically rely on similarity-based metrics like α-nDCG and Self-BLEU, or LLM-as-judge approaches (Que et al., 2024; Bai et al., 2024b), which struggle with longer texts due to their complexity. An alternative is to decompose evaluation into atomic statements, either extracted automatically using search engines or fixed databases for factual accuracy (Song et al., 2024; Wei et al., 2024; Samarinas et al., 2025), or manually designed through expert discussions (Tan et al., 2024) or checklists (Que et al., 2024). However, these methods face verification challenges due to broad or trivial claims from automated extraction and incompleteness from manual design. To address this, objective tasks, such as MMLU (Liu et al., 2024c) and procedural verification, provide more controlled evaluations but often misalign with real-world scenarios. While they support up to 4k tokens, they remain limited for longer texts.

B.3 Code Fixing Evaluation

We evaluate LLM-generated code ($\mathcal{C}_{\text{fixed}}$) against the original ($\mathcal{C}_{\text{orig}}$) using several metrics, applied only after ensuring the submission is a relevant and complete code fix via an auxiliary LLM judge. Irrelevant or incomplete responses receive zero scores.

Runnability Indicates syntactic validity: $r = \mathbb{I}(\texttt{compile}(\mathcal{C}_{\text{fixed}}) \text{ succeeds})$, where $\mathbb{I}(\cdot)$ is the indicator function.

Style Quality Measures Flake8 adherence based on violation count $N_v \geq 0$. With scaling factor $k = 50.0$:

$$q = \begin{cases} (1 + N_v/k)^{-1} & \text{if } r = 1 \text{ \& Flake8 ok} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Length Control Assesses structural fidelity via top-level function counts N_o, N_f (original, fixed). Penalizes difference $\Delta_N = |N_f - N_o|$ relative to scale $S_N = \max(1, 0.25N_o)$.

$$f = \begin{cases} (1 + (\Delta_N/S_N)^2)^{-1} & \text{if AST ok} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Overall Score The harmonic mean aggregates the metrics, penalizing any weak component:

$$\text{Total} = \begin{cases} 3(r^{-1} + q^{-1} + f^{-1})^{-1} & \text{if } r, q, f > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

for small $\epsilon > 0$ (e.g., 10^{-9}). This suite balances correctness, style, and structure.

B.4 Answer Length

Model Name	1k	2k	4k	8k
DeepSeek-R1-Distill-Qwen-32B	874.2	1271.8	1404.6	1585.3
DeepSeek-R1-Distill-Qwen-7B	1430.5	1731.6	1931.0	1809.3
LongWriter-glm4-9b	1791.0	2156.1	2895.6	3277.4
Meta-Llama-3 70B Instruct	662.1	1504.6	2180.7	2613.9
Meta-Llama-3 8B Instruct	1155.8	1592.0	2119.5	2520.0
Meta-Llama-3 8B Instruct (100 samples)	1206.3	1590.4	2092.3	2556.6
Meta-Llama-3 8B Instruct (160)	1189.9	1561.0	1978.3	2540.4
Meta-Llama-3 8B Instruct (200 samples)	1148.6	1525.1	2003.6	2473.1
Meta-Llama-3 8B Instruct (200 samples 2)	1184.8	1646.3	2060.9	2514.3
Meta-Llama-3 8B Instruct (200 samples 3)	1187.8	1608.1	1994.1	2584.1
Meta-Llama-3 8B Instruct (20 sample 3)	1221.6	1551.3	2051.6	2613.2
Meta-Llama-3 8B Instruct (20 samples)	1164.2	1620.5	2105.2	2694.7
Meta-Llama-3 8B Instruct (20 samples 2)	1272.4	1675.2	2085.7	2560.1
Meta-Llama-3 8B Instruct (40 samples)	1156.2	1557.6	2087.3	2568.7
Phi-4-mini-instruct	2088.7	2594.3	2649.3	2606.7
Qwen2 14B Instruct	1360.8	2188.8	2744.8	2987.2
Qwen2 14B Instruct (1M)	913.2	1477.3	1678.4	2092.7
Qwen2 3B Instruct	1130.8	1558.7	1938.7	2178.6
Qwen2 72B Instruct	1242.9	1831.2	2341.0	2515.5
Qwen2 7B Instruct	1069.8	1669.7	2193.4	2351.4
Qwen2 7B Instruct (1M)	933.1	1370.8	1666.8	2087.3
deepseek-r1	559.3	746.5	1078.0	1558.4
deepseek-v3	674.7	1120.3	1405.2	1696.3
gemini-2.0-flash	700.2	1139.4	1748.9	2385.3
gpt-4o-2024-08-06	442.8	541.3	636.4	739.4
gpt-4o-2024-11-20	649.5	802.6	917.0	947.9
llama-4-maverick-17b-128e-instruct	779.1	1160.0	1331.5	1780.2
llama-4-scout-17b-16e-instruct	953.6	1484.9	1888.5	2318.0
o3-mini-2025-01-31	1057.0	2136.4	2958.8	2948.1
qwen-long	647.4	952.8	1364.4	1733.8
qwen3-14b	727.8	2208.0	2836.7	3750.7
qwen3-32b	655.3	1354.9	2411.2	3278.3
qwen3-4b	658.5	1436.0	2455.8	3227.5
qwen3-8b	670.2	1552.8	2666.7	3635.0
qwq-plus-2025-03-05	606.0	857.4	1263.6	1654.2

Table 6: Average Answer Lengths

B.5 AP stype critiaion

B.6 Details of tasks

B.7 Details of sample construction

Excel2text: The methodology facilitates the generation of synthetic transactional sales data intrinsically correlated with corresponding analytical conclusions. The process commences with the definition of foundational sales scenario parameters, including sales region, target fiscal period, currency, overarching sales targets, and antecedent period sales figures, alongside a predefined corpus of sales representatives, product lines, and operational cities. A pivotal aspect involves the stochastic injection of predefined systemic biases during each operational instance; these biases may pertain to overall target achievement (e.g., exceeding, meeting, or missing targets), growth trajectory (positive, neutral, or negative), the anomalous performance of specific sales representatives or products, and variations in new customer acquisition rates. These stochastically determined biases subsequently modulate the synthesis of individual transactional records. Attributes of each transaction, such as sales representative assignment, product selection, customer provenance (new versus existing), and critically, the final transaction value, are probabilistically influenced by the afore-mentioned biases. This ensures that the generated dataset not only achieves a specified volume but also exhibits inherent, bias-driven characteristics across multiple dimensions, thereby providing a feature-rich foundation for subsequent analytical procedures. Upon completion of data synthesis, the resultant structured dataset is subjected to a multi-dimensional analytical engine. This engine emulates real-world business intelligence reporting by performing comprehensive quantitative aggregations and inferential processing across diverse facets, including overall performance metrics (e.g., total sales versus target, period-over-period growth, average transaction value), sales representative efficacy (e.g., top and bottom performers, target attainment distributions), product performance (e.g., leading revenue generators, category contributions), geographical sales distribution, and customer segment analysis (e.g., new versus existing customer value, key account contributions). Key metrics and identified trends derived from this analysis are then articulated as concise, natural language analytical conclusions. To enhance utility and stimulate further inquiry, each conclusion is systematically paired with a relevant analytical query, designed to prompt deeper investigation into the causal factors underpinning the observed phenomena. The system culminates in the delivery of two principal outputs: the raw, granular transactional dataset (typically in CSV format), which serves as the evidentiary basis for analysis, and a structured compendium (typically in JSON format) containing metadata, key performance indicators, and a curated, prioritized set of "conclusion-query" pairings, offering directly consumable insights for simulated business reporting. This integrated pipeline underscores a design philosophy centered on the coherent synthesis of data with its analytical interpretation.

Fix Code: The system employs a generative methodology to synthesize Python source code exhibiting a high density of nuanced linting violations, intended to serve as challenging test in-

CODE FIX

Role: Python Developer

Task: You are given a Python code file that may contain syntax errors or violate style guidelines. Your goal is to fix the code so that it is **runnable** and complies with the following coding standards:

FLAKE8 CATEGORIES TO CHECK:
- E / W – pycodestyle
  Basic PEP 8 formatting errors (E) and warnings (W), such as inconsistent indentation (E111), extra spaces (E221), or line length violations (E501).
- F – Pyflakes
  Potential runtime issues, e.g., undefined names (F821) or unused imports/variables (F401).
- B – flake8-bugbear
  Code patterns prone to bugs or pitfalls, like modifying a list while iterating (B007) or using mutable default arguments (B008).
- N – pep8-naming
  Naming convention violations, such as function names not in snake_case (N802) or class names not in CamelCase (N801).
- SIM – flake8-simplify
  Suggestions to simplify and streamline code, for instance redundant `if x == True` checks (SIM102) or favoring `dict.get` over manual key checks (SIM108).
- C4 – flake8-comprehensions
  Best practices around comprehensions: avoid unnecessary list() wrappers (C400) or use dict comprehensions instead of `dict()` calls with generator expressions (C401).

Input Python Code: Code

```
import decimal,string # E401
......
if __name__=="__main__":
    MainEntryPoint()
# --- END OF CODE ---
```

Instructions:
- Fix Syntax Errors: Ensure the code is valid Python.
- Correct Style Violations: Fix all style issues under the categories above.
- Preserve Functionality: Keep the original behavior, keep the number of functions unchanged, prioritize runnability.
- Output Only Code: Return only the complete, corrected Python code within a single ```python block, without any explanations before or after.

Complete, Corrected Python Code:

Figure 7: An illustrative prompt example for the Code Fix task. The prompt presents a segment of 'dirty' Python code and instructs the LCLM to rectify it according to comprehensive Flake8 standards (E/W/F, B, N, SIM, C4) while maintaining runnability and core functionality.

stances for static analysis tools and code quality assessment. The generation process is initiated by establishing global configuration parameters, including stylistic targets (e.g., line length) and complexity constraints (e.g., maximum nesting depth, function length), alongside lexical resources such as curated lists of nouns, verbs, and adjectives for constructing semantically plausible, albeit potentially misleading, identifiers. A core component is a dynamic scope management system, which tracks variable definitions and usage across nested lexical contexts. This enables the generation of syntactically valid code where identifier-related violations, such as improper naming conventions (e.g., N-series violations from flake8-naming) or unused variables (F841), are contextually embedded. Identifier generation itself is a probabilistic process, designed to stochastically introduce deviations from Python Enhancement Proposal 8 (PEP 8) style guidelines, while also attempting to create names that might subtly obscure their true purpose or shadow existing identifiers in parent scopes. The synthesis of executable code blocks and function bodies is orchestrated through a weighted, probabilistic selection of diverse code constructs. These constructs range from simple assignments and print statements to complex control flow structures like conditional statements and loops. Each construct generator is imbued with the capability to introduce specific categories of violations. For instance, conditional statement generators might create explicit boolean comparisons (SIM21x) or if-else patterns amenable to ternary expressions (SIM108). Loop generators may produce unconventional iter-

GEN KV DICTIONARY

Generate a Python dictionary with the following requirements:
- **Total entries:** 20
- **MUST include the entry:** 'DKUNULZASGUKBXPBVUNROMKQMOKMTFBC': 'iwidr01birzxfv6s8hkdxysaoyw8ce4i'
- **The special entry should be placed at index** 12
- **Other keys and values must follow these rules:**
  * Keys must be random strings of length 32, consisting ONLY of uppercase letters (A-Z) and underscores (_)
  * Values must be random strings of length 32, consisting ONLY of lowercase letters (a-z) and digits (0-9)
  * Keys and values MUST NOT contain any special characters (e.g., /, =, $, @, :, etc.) or spaces
- **Output ONLY the dictionary in the following format** (as a single-line string):
{'...': '...', ...,
'DKUNULZASGUKBXPBVUNROMKQMOKMTFBC':
'iwidr01birzxfv6s8hkdxysaoyw8ce4i', ..., '...': '...'
}
- **Ensure the dictionary string is valid JSON** and can be parsed by `json.loads()` without errors.
- **DO NOT include any code or explanations.** Only return the dictionary string.

Figure 8: An illustrative prompt example for the Gen KV Dictionary task. The prompt details the requirements for generating a Python dictionary, including the total number of entries, a specific key-value pair to be inserted, its target index, and formatting rules for other entries.

ator variable names or inefficient comprehensions (C4xx series from flake8-comprehensions). Furthermore, generators for function definitions are specifically designed to introduce more complex issues, such as mutable default arguments (B006 from flake8-bugbear) or function calls within default argument expressions (B008), often obfuscated by the presence of other parameters and non-trivial function bodies. Whitespace and formatting violations (E-series and W-series) are pervasively introduced at various granularities, from inconsistent spacing around operators and after commas to improper blank line usage and trailing whitespace. The system also synthesizes a sequence of interdependent functions, simulating a rudimentary program flow (e.g., data loading, validation, analysis, reporting), which are ultimately orchestrated within a main execution block. This structural coherence provides a more realistic backdrop for the embedded violations, moving beyond isolated infractions to scenarios requiring more holistic refactoring. The overall probability of introducing a violation is a configurable parameter, allowing for control over the density of infractions, with the system actively aiming to make these violations less trivial to automatically or manually remediate by intertwining them with functional, albeit flawed, program logic. The final output is a runnable Python script, replete with these intentionally challenging, multi-category linting issues.

KG2Text: system synthesizes rich, protagonist-centric knowledge graphs (KGs) and subsequently translates salient subgraphs into natural language narratives. The generative process for each KG commences with the instantiation of a unique protagonist, whose attributes, including socio-economic background and a randomly assigned character archetype (e.g., Scientist, Artist, En-

KG2TEXT

Role: Biographer / Content Writer

Task: Write a coherent and readable biography about the entity associated with the slug '00006_ambassador_cathy_allen'. Your biography must be based **exclusively** on the factual statements provided below in Subject-Predicate-Object (Triple) format. Combine the facts naturally into a narrative.

Input Facts (Triples): Knowledge Graph

- A Manifesto for Incubate Impactful E-Markets - authored by - Ambassador Cathy Allen
- A Manifesto for Incubate Impactful E-Markets - publication year - 1874
- A Manifesto for Incubate Impactful E-Markets - work type - Manifesto
- Ambassador Cathy Allen - authored - A Manifesto for Incubate Impactful E-Markets
- Ambassador Cathy Allen - authored year - 2005
- Ambassador Cathy Allen - birth year - 1977
-- Justin Bowman - death year - 1998
- Justin Bowman - job - Senator
- Justin Bowman - nationality - Sri Lanka
- Justin Bowman - socioeconomic background - Upper Class
- Justin Bowman - stated motivation - Sought to create lasting change.
- Major Promotion (1821) - context description - This event occurred during a period of industrialization and rising global tensions.
- Major Promotion (1821) - event type - Personal Milestone
- Major Promotion (1821) - historical era - Pre-WWI Era
- Major Promotion (1821) - participant - Ambassador Cathy Allen
- Major Promotion (1821) - significance - Medium
- Major Promotion (1821) - year - 1821
- Nevada - place type - Region

Writing Style:
Produce a well-structured paragraph or paragraphs. Ensure smooth transitions between facts where possible. The tone should be informative and neutral.

Required Content:
Ensure that the core information from *each* of the input triples is included in your generated biography.

Length Specifications (TARGET WORD COUNT):
- The biography should be around 1024 words. Strive for this length, but prioritize covering all facts accurately.

You may now begin writing the biography based on the provided triples around 1024 words:

Figure 9: An illustrative prompt example for the KG 2 Text task. The prompt provides a set of knowledge graph triples and asks the LCLM to synthesize them into a coherent biographical text of a specified target word count.

trepreneur), are stochastically determined. These initial conditions significantly influence the subsequent probabilistic expansion of the KG. The protagonist's lifespan and historical era are also established to ensure temporal coherence for related entities and events. The KG is then incrementally constructed through an iterative expansion process originating from the protagonist. At each step, existing nodes are selected for expansion based on their proximity to the protagonist and predefined archetypal relationship propensities. New nodes, representing persons, organizations, places, creative works, or events, are generated with contextually relevant attributes, or existing nodes are connected, adhering to a set of permissible relationship types defined within a structured map. This map also dictates the likelihood of specific relationships based on the source node's type and, for persons, their current life phase (e.g., Childhood, Education, MidCareer). Attribute generation for new entities, such as names, job titles, or event descriptions, leverages procedural generation techniques and controlled randomness, often influenced by the protagonist's established background and archetype to foster narrative consistency. Temporal plausibility is rigorously maintained by ensuring that dates associated with relationships and events align with the lifespans of involved entities. Once a KG reaches a target size or expansion limits, a focused subgraph is extracted. This subgraph typically comprises nodes within a specified graph distance from the protagonist, representing the most narratively relevant portion of the larger KG. This subgraph then serves as the direct input for the text generation phase. Each node attribute (excluding the primary name) and each relationship within

NEWS WRITING

Instructions:
Write a news report titled 'Rewilding the Earth: A Revolutionary Approach to Ecosystem Restoration.'
Cover: Introduce the concept of rewilding in the fictional country of 'Valora,' where massive wildlife corridors are being created to restore ecosystems and reintroduce extinct species.
.....

Style Guidelines (AP Style):

Content Requirements:
You MUST include ALL of the following **STATEMENTS** as listed below:
1. The pilot results show that two-thirds of the rewilded areas exhibit increased wildlife activity.
2. In Valora's project, the government has decided to reintroduce extinct species like the saber-toothed tiger, woolly mammoth; to restore ecosystem balance.
……

Length Specifications:
- **TARGET WORD COUNT:** Generate an article that is **around to 1024 words**. Significant deviation from this length will be penalized.

You may now begin your {self.test_length} words writing:

Figure 10: An illustrative prompt example for the News Writing task. The prompt includes a news query, a list of factual statements to incorporate (and potentially correct for AP style), the AP style rubric, and a target word count for the generated article.

this subgraph, along with significant attributes of these relationships (e.g., roles, dates, specific details like degree or investment amount), are systematically converted into individual descriptive sentences using predefined, templated linguistic patterns. These patterns map structural KG elements (subject-predicate-object triples, or subject-attribute-value) to natural language constructs. The system's output for each generated KG is multi-faceted, including the full KG data, the extracted subgraph data, and the derived natural language sentences, typically stored in structured JSON files. Optionally, visualizations of both the full KG and the subgraph can be produced using graph layout algorithms. Finally, as an aggregative step, the natural language sentences generated from all individual KGs within a single execution run are compiled into a consolidated dataset, facilitating larger-scale analysis or downstream natural language processing tasks. This methodology emphasizes the creation of datasets where structured knowledge and its textual manifestation are coherently and traceably linked, grounded in simulated sociological and temporal contexts.

News Writing: The system under discussion is designed to rigorously evaluate a large language model's (LLM) proficiency in generating news reports that conform to the Associated Press (AP) style guidelines. This evaluation is predicated on the model's ability to synthesize a coherent narrative based on a given news topic query, integrate a series of predefined factual statements, and adhere to a specified target word count, all while meticulously applying AP style conventions. The process of generating verifiable test data, specifically the factual statements, is a critical precursor to the evaluation. These statements are meticulously crafted to serve as direct inputs that the LLM must incorporate into its generated news article. Crucially, each statement is designed to test a specific facet of the AP style guide; thus, many are intentionally formulated to violate these rules. For instance, a statement might employ incorrect number usage (e.g., writing out "eleven" instead of using the numeral "11"), misuse punctuation (e.g., including an Oxford comma), or improperly format dates, times, or

PARAGRAPH ORDERING

Please rearrange the following paragraphs into a logically coherent article:

[[Segment 0]]
The result is completely random, except for the first 3 cards from the Whimsical machine. In particular, ⋯

[[Segment 2]]
xx

Requirements:
1. Keep the original content of paragraphs unchanged, only adjust their order
2. Use [[Segment X]] to identify original paragraph numbers, starting from 0 up to 2.
3. Output the complete content in final order (include paragraph identifiers)
4. The final output must contain exactly 3 segments

Example:
[[Segment 0]]
Paragraph content
[[Segment 1]]
Another paragraph content
...

Figure 11: An illustrative prompt example for the Paragraph Ordering task. The prompt presents a set of shuffled paragraphs, each tagged with an identifier (e.g., "[[Segment i]]"), and requires the LCLM to output the correct sequence of these identifiers to form a coherent text.

titles. Accompanying each such potentially flawed statement in the test dataset is its corresponding correct AP style expression and a clear rationale explaining the nature of the original stylistic error. This structured approach ensures that each statement serves as a verifiable unit for assessing the LLM's capacity for rule-based stylistic correction. The construction of the prompt provided to the generative LLM is a multi-component process. It begins with the query, which defines the overarching news topic, often suggesting a narrative structure or specific angles to be explored. To this, the complete AP style rubric—a comprehensive guide detailing rules across numerous categories with illustrative examples—is appended. A key element of the prompt is a curated list of the aforementioned factual statements. These statements, presented in their original, potentially non-compliant form, are explicitly designated as mandatory inclusions for the generated article. The prompt also specifies the target word count, imposing a length constraint on the LLM's output. This careful assembly of the prompt creates a challenging scenario where the LLM must not only generate fluent and relevant content based on the query but also actively engage with the AP style guide to identify and rectify the stylistic infelicities within the provided statements as they are woven into the narrative. The verifiability of the task lies in the direct comparison of the model's treatment of these embedded statements against their known correct AP style forms, all within the context of the broader news writing assignment.

SALES REPORT

Role: Senior Business Analyst

Task: You are provided with raw sales transaction data in CSV format. Your goal is to perform a detailed analysis based *only* on this data and generate a comprehensive sales performance report.

Input Sales Data (CSV Format): Excel

```csv
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | OrderID | OrderDate | Region | City | Salesperson | Salesperson | Salesperson | CustomerID | CustomerNa | IsNewCusto | ProductID | ProductNan | ProductCate | Quantity | UnitPrice | TotalSaleAm | WeekOfYea | DayOfWeek | DayOfMonth |
| 2 | ORD-2025-( | 2025/4/7 | East Region | Philadelphia | EMP008 | Meghan Ro | 130000 | CUST-9151- | Tran-Griffit | FALSE | PROD-O01 | Extended W | Other | 1 | 1200 | 1073.07 | 15 | Monday | 7 |
| 3 | ORD-2025-( | 2025/4/29 | East Region | Pittsburgh | EMP009 | Danielle Har | 93000 | CUST-3153- | Mullen, John | FALSE | PROD-S03 | Predictive M | Software | 2 | 7500 | 17188.87 | 18 | Tuesday | 29 |
| 4 | ORD-2025-( | 2025/4/20 | East Region | New York | EMP002 | Matthew Jor | 87000 | CUST-8239- | Cruz-Mc | FALSE | PROD-C01 | Sensor Pack | Consumable | 20 | 500 | 8739.04 | 16 | Sunday | 20 |
| 5 | ORD-2025-( | 2025/4/9 | East Region | Newark | EMP001 | Carlos Malo | 105000 | CUST-3292- | Rodgers | FALSE | PROD-O01 | Extended W | Other | 1 | 1200 | 1166.24 | 15 | Wednesday | 9 |
| 6 | ORD-2025-( | 2025/4/27 | East Region | Miami | EMP016 | Kimberly Oc | 98000 | CUST-4750- | Mcneil Gro | FALSE | PROD-V01 | Platinum Su | Service | 1 | 3000 | 5752.21 | 17 | Sunday | 27 |
| 7 | ORD-2025-( | 2025/4/1 | East Region | Philadelphia | EMP008 | Meghan Ro | 130000 | CUST-1750- | Skinner Bro | FALSE | PROD-S04 | Reporting D | Software | 1 | 2000 | 2803.01 | 14 | Tuesday | 1 |
| 8 | ORD-2025-( | 2025/4/18 | East Region | Newark | EMP001 | Carlos Malo | 105000 | CUST-NEW- | Stewart | FALSE | PROD-S02 | Data Integra | Software | 2 | 4500 | 10441.19 | 16 | Friday | 18 |
| 9 | ORD-2025-( | 2025/4/16 | East Region | Philadelphia | EMP003 | Duane Garci | 112000 | CUST-9875- | Weaver | FALSE | PROD-S02 | Data Integra | Software | 1 | 4500 | 4995.17 | 16 | Wednesday | 16 |
| 10 | ORD-2025-( | 2025/4/13 | East Region | Washington | EMP014 | Michael Jon | 94000 | CUST-NEW- | Gutierrez-Ba | TRUE | PROD-H03 | Network Sw | Hardware | 8 | 3500 | 29921.17 | 15 | Sunday | 13 |
| 11 | ORD-2025-( | 2025/4/24 | East Region | Newark | EMP017 | Patrick Mart | 103000 | CUST-NEW- | Anderson-T | TRUE | PROD-S04 | Reporting D | Software | 4 | 2000 | 9919.8 | 17 | Thursday | 24 |
```

Analysis Structure Guidance:
Please structure your sales performance report logically. Start with an overall performance summary, then delve into analyses of sales representative performance, product performance, and any other relevant insights identified from the data. Use a narrative style suitable for a management report, ensuring all insights are directly derived from the provided CSV data.

Required Content - Address These Specific Questions:
Within your structured analysis, ensure you specifically attempt to answer the following questions based *only* on the provided data:

- **Question 1:** Who was the top sales representative by revenue and what was their contribution?
- **Question 2:** What was the average deal size across all transactions?
- **Question 3:** Which product generated the most revenue and what was its contribution?
- **Question 4:** How many sales representatives met or exceeded their sales targets?

……

- **Question 15:** Who was the second-ranked sales representative and how far behind the leader were they?
- **Question 16:** How did the top performer's average deal size compare to the team average?
- **Question 17:** What were the top 3 products by revenue?
- **Question 18:** How did the top product's average sale value compare to the overall average deal size?
- **Question 19:** What percentage of total revenue did the top 3 products contribute collectively?
- **Question 20:** What was the contribution of new customers to total revenue (count and percentage)?

- Length Specifications (TARGET WORD COUNT):
- The report should be **around 1024 words**. The deviation from this length may affect your evaluation.

You may now begin your analysis and write the approximately 1024 words report:

Figure 12: An illustrative prompt example for the Sales Report task. The prompt provides sales data (as a Markdown table derived from CSV) and a list of analytical questions, instructing the LCLM to generate a structured report that includes answers to these questions.

STATE MACHINE

Your task is to simulate a state transition process based on the following rules.
The input string for this simulation is:
'2020112011201010121112012102100022202000222211212010110'.

The state machine operates with the following configuration:

1. Initial State: S0
2. State Transition Rules:

```
Current State | Input | Next State | Output Signal
-----------------------------------------------
S0            | 0     | S0         | 0
S0            | 1     | S1         | 1
S0            | 2     | S2         | 2
S1            | 0     | S1         | 1
S1            | 1     | S2         | 2
S1            | 2     | S0         | 0
S2            | 0     | S2         | 2
S2            | 1     | S0         | 0
S2            | 2     | S1         | 1
```

Here is an example of a valid state transition process:
Assume the input string is '202'. The state transition process would be as follows:

```
Current State | Input | Next State | Output Signal
-----------------------------------------------
S0            | 2     | S2         | 2
S2            | 0     | S2         | 2
S2            | 2     | S1         | 1
```

Note: The above example is dynamically generated based on the state transition rules and the input string. The actual output may vary depending on the specific input string.

Based on the above rules, please generate a simulated state transition process for the input string '2020112011201010121112012102100022202000222211212010110'.
Display the current state, input, next state, and output signal for each step.
Ensure that the generated process strictly adheres to the state machine rules.
Important:
1. Do NOT generate any code or explanatory text. 2. Do **NOT** use any form of truncation. You **must** list all steps.
Only provide the state transition process in the following format:

```
Current State | Input | Next State | Output Signal
-----------------------------------------------
<State>       | <Char>| <NextState>| <Output>
...
```

Figure 13: An illustrative prompt example for the State Machine task. The prompt defines a complete finite state machine (initial state, transition table) and provides an input string, requiring the LCLM to simulate the state transitions step-by-step and output the resulting states and signals.

CODE FIX

Role: Python Developer

Task: You are given a Python code file that may contain syntax errors or violate style guidelines. Your goal is to fix the code so that it is **runnable** and complies with the following coding standards:

FLAKE8 CATEGORIES TO CHECK:
- E / W – pycodestyle
  Basic PEP 8 formatting errors (E) and warnings (W), such as inconsistent indentation (E111), extra spaces (E221), or line length violations (E501).
- F – Pyflakes
  Potential runtime issues, e.g., undefined names (F821) or unused imports/variables (F401).
- B – flake8-bugbear
  Code patterns prone to bugs or pitfalls, like modifying a list while iterating (B007) or using mutable default arguments (B008).
- N – pep8-naming
  Naming convention violations, such as function names not in snake_case (N802) or class names not in CamelCase (N801).
- SIM – flake8-simplify
  Suggestions to simplify and streamline code, for instance redundant `if x == True` checks (SIM102) or favoring `dict.get` over manual key checks (SIM108).
- C4 – flake8-comprehensions
  Best practices around comprehensions: avoid unnecessary list() wrappers (C400) or use dict comprehensions instead of `dict()` calls with generator expressions (C401).

Input Python Code: Code

```
import decimal,string # E401
......
if __name__=="__main__":
    MainEntryPoint()
# --- END OF CODE ---
```

Instructions:
- Fix Syntax Errors: Ensure the code is valid Python.
- Correct Style Violations: Fix all style issues under the categories above.
- Preserve Functionality: Keep the original behavior, keep the number of functions unchanged, prioritize runnability.
- Output Only Code: Return only the complete, corrected Python code within a single ```python block, without any explanations before or after.

Complete, Corrected Python Code:

Figure 14: Another illustrative prompt example for the Code Fix task, showcasing the input format where 'dirty' Python code is provided alongside instructions for Flake8-compliant correction and functionality preservation.

Figure 15: An example visualization of a synthetic Knowledge Graph (KG) segment used in the KG 2 Text task. Such graphs provide the structured triple data (subject-predicate-object) that LCLMs are tasked with converting into narrative text.

"Category 1": "Number Usage",

"Scoring_Criteria": "- Number writing rules:\n 1-9 should be written in words; 10 and above should use Arabic numerals.\n Always use Arabic numerals for ages, amounts, percentages, dates, and times.\n- Ordinal numbers:\n Do not use 'st', 'nd', 'rd', or 'th'.\n- Plural forms:\n Do not add an apostrophe to plural numbers (e.g., '7s').",

"Incorrect_Examples": "- 'I have 7 apples.'\n- 'The event is on the 3rd day.'\n- 'She is twenty-five years old.'\n- 'All 7's rolled.'",

"Correct_Examples": "- 'I have seven apples.'\n- 'The event is on the third day.'\n- 'She is 25 years old.'\n- 'All 7s rolled.'"

"Category 2": "Punctuation",

"Scoring_Criteria": "- Quotation marks:\n Periods and commas always go inside quotation marks.\n- Oxford comma:\n Avoid using the Oxford comma in lists.\n- Space rules:\n Use only one space after a period.\n- Colons:\n Capitalize the first letter after a colon only if it starts a complete sentence or is a proper noun.",

"Incorrect_Examples": "- He said, 'Let's go'.\n- Red, white, and blue.\n- This is a sentence. Another one follows.\n- 'The following: rules.'",

"Correct_Examples": "- He said, 'Let's go.'\n- Red, white and blue.\n- This is a sentence. Another one follows.\n- 'The following: Rules.'"

"Category 3": "Dates and Times",

"Scoring_Criteria": "- Date rules:\n Do not use ordinal indicators like '1st'.\n Abbreviate months as Jan., Feb., Aug., Sept., Oct., Nov., Dec.\n- Time rules:\n Use a.m./p.m. format and omit ':00' for whole hours.\n Write 'midnight' and 'noon' instead of '12 a.m.' or '12 p.m.'",

"Incorrect_Examples": "- 'The event is on July 3rd.'\n- 'It starts at 8:00 p.m.'\n- 'Meet me at 12 p.m.'",

"Correct_Examples": "- 'The event is on July 3.'\n- 'It starts at 8 p.m.'\n- 'Meet me at noon.'"

"Category 4": "Addresses and Locations",

"Scoring_Criteria": "- Street address rules:\n Abbreviate 'Ave.', 'Blvd.', 'St.' but spell out 'Road'.\n- State name rules:\n Abbreviate state names after city names (except Alaska, Hawaii, Idaho, etc.).\n- Direction rules:\n Use lowercase for directions like 'north' and 'south'.",

"Incorrect_Examples": "- '1600 Pennsylvania Avenue'\n- 'Nashville, Tennessee'\n- 'We went to the East last year.'",

"Correct_Examples": "- '1600 Pennsylvania Ave.'\n- 'Nashville, Tenn.'\n- 'We went east last year.'"

"Category 5": "Titles and Positions",

"Scoring_Criteria": "- Title rules:\n Capitalize formal titles before a person's name; use lowercase after the name or when used alone.\n Avoid courtesy titles like Mr., Mrs., Ms.\n- Gender-neutral language:\n Use gender-neutral terms (e.g., 'police officer' instead of 'policeman').",

"Incorrect_Examples": "- 'President Joe Biden visited.'\n- 'Joe Biden, President, spoke.'\n- 'The policeman arrived.'",

"Correct_Examples": "- 'President Joe Biden visited.'\n- 'Joe Biden, the president, spoke.'\n- 'The police officer arrived.'"

"Category 6": "Media References",

"Scoring_Criteria": "- Reference rules:\n Add quotation marks around titles of articles, books, movies, songs, etc.\n Do not use quotation marks for newspaper or magazine names; capitalize them.\n Use website titles instead of URLs.\n- Reference format:\n Provide full information on first mention; simplify subsequent mentions.",

"Incorrect_Examples": "- 'I read the New York Times today.'\n- 'Check out www.google.com.'\n- 'According to the study.'",

"Correct_Examples": "- 'I read The New York Times today.'\n- 'Check out Google.'\n- 'According to the study by Smith et al.'"

"Category 7": "Capitalization Rules",

"Scoring_Criteria": "- Proper nouns:\n Capitalize specific names (e.g., places, institutions).\n- Seasons and directions:\n Only capitalize seasons/directions in specific cases (e.g., 'Winter Olympics').\n- Abbreviations:\n Follow capitalization rules for abbreviations.",

"Incorrect_Examples": "- 'The River flows north.'\n- 'We went to the East last year.'\n- 'The study was conducted in the Summer.'",

"Correct_Examples": "- 'The river flows north.'\n- 'We went east last year.'\n- 'The study was conducted in the summer.'"

"Category 8": "Technical Terms",

"Scoring_Criteria": "- Technical terms:\n Spell technical terms correctly (e.g., 'email', 'smartphone').\n Capitalize brand names like 'iPhone'.\n Write 'website' as one word and 'web page' as two words.\n- Spelling:\n Ensure accurate spelling of technical terms.",

"Incorrect_Examples": "- 'I sent an E-mail.'\n- 'Check out my new Iphone.'\n- 'Visit our Webpage.'",

"Correct_Examples": "- 'I sent an email.'\n- 'Check out my new iPhone.'\n- 'Visit our web page.'"

"Category 9": "Clarity and Brevity",

"Scoring_Criteria": "- Brevity:\n Avoid long or complex sentences.\n- Readability:\n Use simple language suitable for general audiences.\n- Consistency:\n Maintain consistent style throughout the text.",

"Incorrect_Examples": "- 'The aforementioned individual arrived at the location.'\n- 'This is a highly technical subject matter.'",

"Correct_Examples": "- 'The person arrived at the site.'\n- 'This is a technical topic.'"

"Category 10": "Overall Consistency",

"Scoring_Criteria": "- Consistency:\n Maintain consistent style for numbers, punctuation, capitalization, etc.\n- Style:\n Avoid mixing styles (e.g., APA, Chicago).\n- Structure:\n Ensure logical flow and clear paragraph transitions.",

"Incorrect_Examples": "- Mixed number formats (e.g., 'seven' and '8')\n- Inconsistent punctuation (e.g., 'quote.' vs. 'quote.')\n- Large jumps between paragraphs.",

"Correct_Examples": "- Unified number formats.\n- Consistent punctuation.\n- Smooth paragraph transitions."

Figure 16: Generation and Evaluation Rules for Anchors in the News Writing Task. Ten dimensions, each containing rules, positive examples, and error cases.