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

Zikai Xiao[1*†], **Fei Huang**[2*], **Jianhong Tu**[2], **Jianhui Wei**[1], **Wen Ma**[1], **Yuxuan Zhou**[2],
Jian Wu[1], **Bowen Yu**[2], **Zuozhu Liu**[1‡], **Junyang Lin**[2‡]

[1]Zhejiang University, [2]Qwen Team, Alibaba Inc.
zuozhuliu@intl.zju.edu.cn, junyang.ljy@alibaba-inc.com

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 balances real-world and verifiable assessment with Constraint-Verifier Evaluation (CoV-Eval). CoV-Eval constructs tasks by first defining verifiable targets within real-world scenarios, then systematically generating corresponding queries, textual materials, and constraints 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 shows that even state-of-the-art models encounter significant challenges in long-form generation as real-world complexity and output length increase. Our codes are available at https://github.com/ZackZikaiXiao/LongWeave.

1 Introduction

Large Language Models (LLMs) have significantly enhanced their capabilities to process long inputs (Yang et al., 2024a, 2025b; 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). 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.

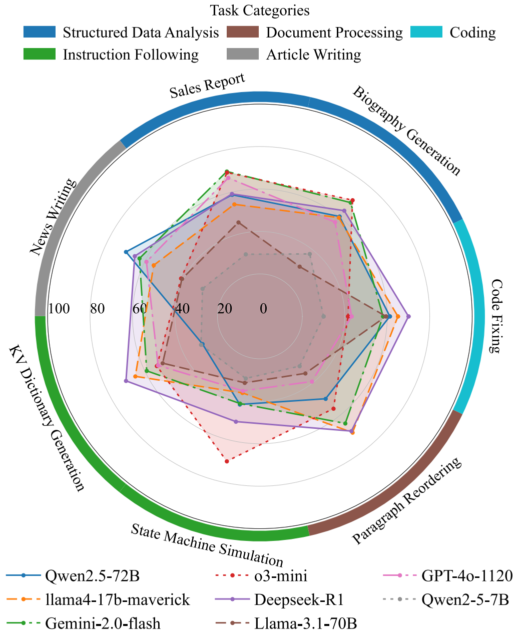


Figure 1: The performance across the seven tasks in LongWeave. For better visualization, performance scores have been normalized to a range of 0.3 to 0.7.

Long-form generation with real-world queries is typically evaluated using similarity metrics (e.g., α-nDCG, Self-BLEU) or LLM-as-a-Judge (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

---

[*] Co-first authors contribute equally to this work.

[†] Work done during an internship at Qwen Team.

[‡] Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 10386–10417
November 4-9, 2025 ©2025 Association for Computational Linguistics

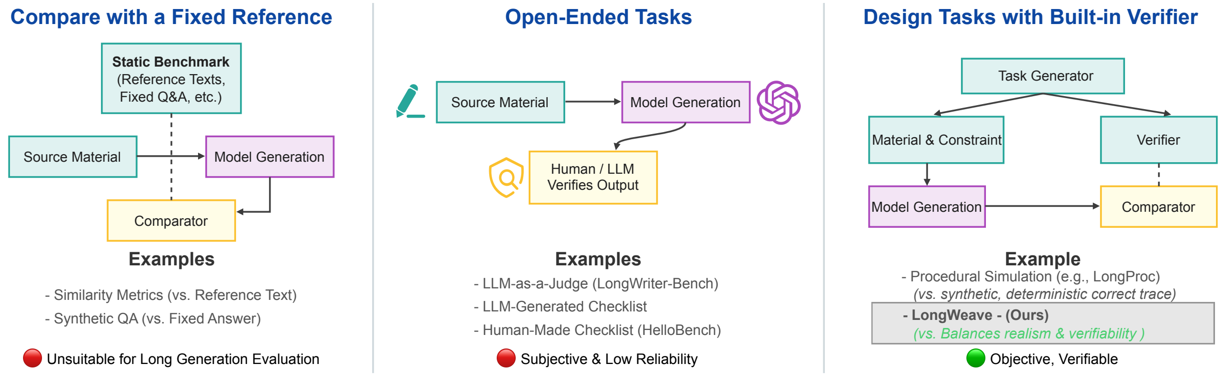

Compare with a Fixed Reference
Static Benchmark
(Reference Texts, Fixed Q&A, etc.)
Source Material
Model Generation
Comparator
Examples
- Similarity Metrics (vs. Reference Text)
- Synthetic QA (vs. Fixed Answer)
🔴 Unsuitable for Long Generation Evaluation
Open-Ended Tasks
Source Material
Model Generation
Human / LLM Verifies Output
Examples
- LLM-as-a-Judge (LongWriter-Bench)
- LLM-Generated Checklist
- Human-Made Checklist (HelloBench)
🔴 Subjective & Low Reliability
Design Tasks with Built-in Verifier
Task Generator
Material & Constraint
Verifier
Model Generation
Comparator
Example
- Procedural Simulation (e.g., LongProc)
(vs. synthetic, deterministic correct trace)
- LongWeave - (Ours)
(vs. Balances realism & verifiability)
🟢 Objective, Verifiable

Figure 2: Three evaluation paradigms for long-form generation. LongWeave is grounded in real-world scenarios and based on objective, verifiable scoring with built-in ground truth, reducing subjectivity and inconsistencies.

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., 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 overcomplicate 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 Constraint-Verifier Evaluation (CoV-Eval) mechanism, as illustrated in Figure 2. Rather than extracting checklists from raw materials, which is error-prone and hard to control, CoV-Eval reverses the test construction process: it begins with predefined verifiable checklist objectives (Verifiers) grounded in real-world tasks, then synthesizes corresponding inference queries (including Constraints and materials). The Constraint acts as a constrained input that causally guides models toward generating the predefined Verifier, enabling measurable verification and evaluation. Each Constraint-Verifier (CV) pair in CoV-Eval maintains a deterministic one-to-one relationship under structurally defined rules, systematically linked to source materials. CoV-Eval contains a series of CV pairs, where each pair is linked to the corresponding material. These pairs can take various forms, such as a question (C) and answer (V) in QA tasks, or a triplet (C) and corresponding sentence (V) in knowledge-to-text generation, as discussed in Section 2.3.

Based on CoV-Eval, 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 fundamentally solve long-generation challenges and may even degrade performance. Moreover, large-scale reasoning-oriented LLMs consistently outperform general-purpose counterparts, but often suffer from failure to terminate the reasoning phase, leading to truncated outputs. Our main contributions are:

- We introduce the long-form generation benchmark **LongWeave**, with CoV-Eval that bridges real-world relevance with verifiability.

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 Constraint-Verifier Evaluation (CoV-Eval) is a constrained open-ended evaluation that synthetically constructs tasks to ensure real-world relevance. Color highlights indicate strengths (green) or challenges (orange). The length refers to the number of tokens under the cl100k tokenizer. The ✓∗ symbol denotes that the characteristic is present in a subset of the benchmark's tasks.

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-a-Judge
InfiniteBench (Zhang et al., 2024)	Infinite	∼100	×	✓	Rules
Benchmarks for Long Generation					
LongWriter-Bench (Bai et al., 2024b)	∼100	∼5k	✓	×	LLM-as-a-Judge
LongGenBench[1] (Liu et al., 2024c)	∼1k	∼4k	×	✓	Similarity
LongGenBench[2] (Wu et al., 2025)	∼100	∼8k	✓	✓	LLM-as-a-Judge
Hello Bench (Que et al., 2024)	∼300	∼8k	✓	×	LLM-as-a-Judge
LongProc (Ye et al., 2025)	∼32k	∼8k	×	✓	Rules
LongWeave	**64k**	**8k**	✓*	✓*	**Constraint-Verifier Pairs**

- We design seven tasks, with long input sizes (up to 64K tokens), long output requirements (1-8K), and varying difficulty levels.
- 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 Constraint-Verifier Evaluation and a description of the individual tasks.

2.1 Pipeline of LongWeave

As shown in Figure 3, 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) *constraint*, and (3) *verifier*. The LLM processes the material and constraints during Evaluation to produce a response that meets constraints. Finally, in the scoring phase, the output is compared to the target and then aggregated to calculate the total score. Finally, in Scoring, the output is compared to the verifier using a scoring function and aggregated to calculate the total score.

2.2 Constraint-Verifier-Based Evaluation (CoV-Eval)

Formulation of Basic Evaluation. We formulate long-form constrained generation as the task where an 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 criteria for the target output's 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.

Data Construction Stage of CoV-Eval. To ensure the benchmark is both realistic and verifiable, we introduce a construction process that jointly generates the raw material X_{raw}, the constraint C, and the corresponding Verifier V. The entire construction process is formalized as:

$$(X_{\text{raw}}, C, V) \leftarrow f_{\text{gen}}(\boldsymbol{\theta}), \quad \text{where} \quad \boldsymbol{\theta} \sim \Theta \tag{2}$$

The process is driven by a **Generator** (f_{gen})—a set of task-specific, deterministic, rule-based scripts, which can be seen **in the bottom part of Figure 3**. The generator's behavior is controlled by structured **Attribute Seeds** ($\boldsymbol{\theta}$), which are sampled from a predefined attribute space Θ and specify properties like material scale, reasoning complexity, and constraint strictness. CoV-Eval combines deterministic generation with explicit attribute control, guarantees that every Constraint–Verifier pair is grounded in its material, and can be automatically verified.

Evaluation Stage. The input instruction incorporates both the material and the constraint, while

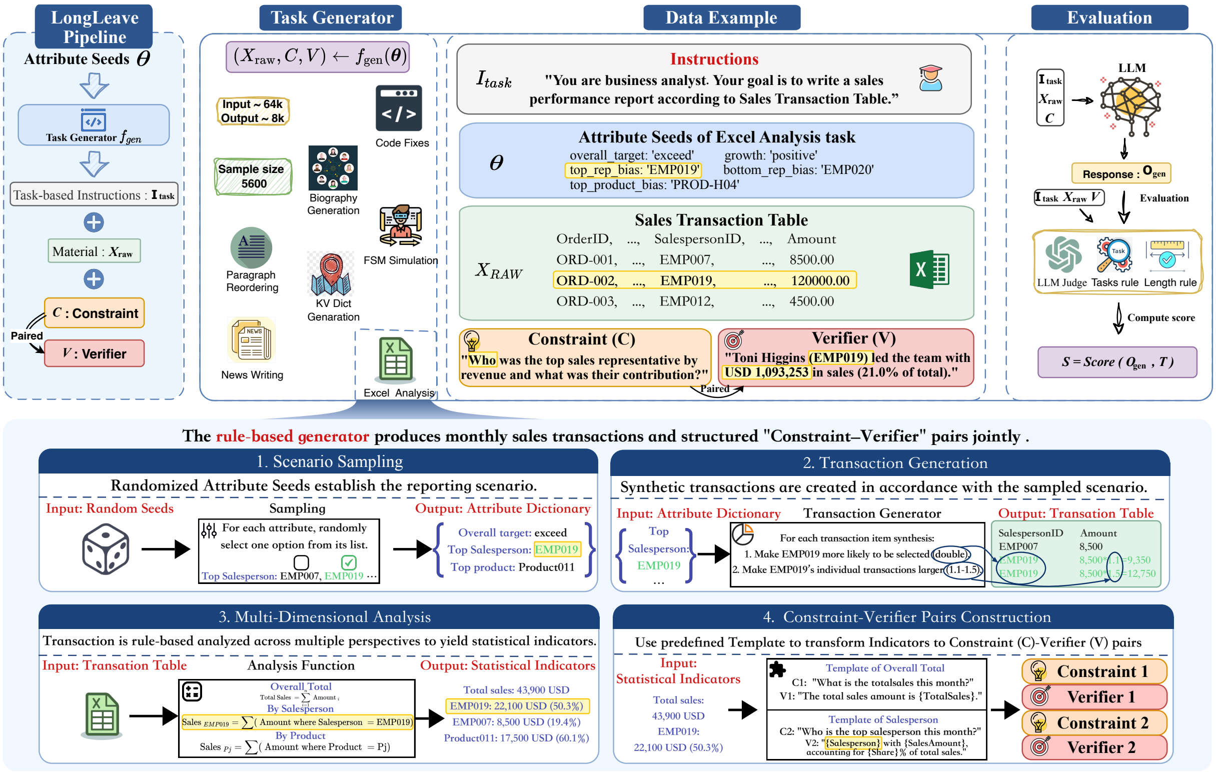

LongLeave Pipeline
Attribute Seeds θ
Task Generator f_{gen}
Task-based Instructions : I_{task}
Material : X_{raw}
C : Constraint
Paired
V : Verifier
Task Generator
$(X_{raw}, C, V) \leftarrow f_{gen}(\theta)$
Input ~ 64k
Output ~ 8k
Code Fixes
Sample size 5600
Biography Generation
FSM Simulation
Paragraph Reordering
KV Dict Generation
News Writing
Excel Analysis
Data Example
Instructions
I_{task}
"You are business analyst. Your goal is to write a sales performance report according to Sales Transaction Table."
Attribute Seeds of Excel Analysis task
θ
overall_target: 'exceed'
growth: 'positive'
top_rep_bias: 'EMP019'
bottom_rep_bias: 'EMP020'
top_product_bias: 'PROD-H04'
Sales Transaction Table
X_{RAW}
OrderID, ..., SalespersonID, ..., Amount
ORD-001, ..., EMP007, ..., 8500.00
ORD-002, ..., EMP019, ..., 120000.00
ORD-003, ..., EMP012, ..., 4500.00
Constraint (C)
"Who was the top sales representative by revenue and what was their contribution?"
Verifier (V)
"Toni Higgins (EMP019) led the team with USD 1,093,253 in sales (21.0% of total)."
Paired
Evaluation
I_{task}
X_{raw}
C
LLM
Response : O_{gen}
I_{task} X_{raw} V
Evaluation
LLM Judge
Tasks rule
Length rule
Compute score
$S = Score\,(O_{gen}, T)$
The rule-based generator produces monthly sales transactions and structured "Constraint–Verifier" pairs jointly .
1. Scenario Sampling
Randomized Attribute Seeds establish the reporting scenario.
Input: Random Seeds
Sampling
Output: Attribute Dictionary
For each attribute, randomly select one option from its list.
Top Salesperson: EMP007, EMP019 ...
Overall target: exceed
Top Salesperson: EMP019
Top product: Product011
2. Transaction Generation
Synthetic transactions are created in accordance with the sampled scenario.
Input: Attribute Dictionary
Transaction Generator
Output: Transation Table
Top Salesperson: EMP019 ...
For each transaction item synthesis:
1. Make EMP019 more likely to be selected (double)
2. Make EMP019's individual transactions larger (1.1-1.5)
SalespersonID Amount
EMP007 8,500
EMP019 8,500*1.1=9,350
EMP019 8,500*1.5=12,750
3. Multi-Dimensional Analysis
Transaction is rule-based analyzed across multiple perspectives to yield statistical indicators.
Input: Transation Table
Analysis Function
Output: Statistical Indicators
Overall Total
Total Sales = ∑ Amount ,
By Salesperson
$Sales_{EMP019} = \sum$(Amount where Salesperson = EMP019)
By Product
$Sales_{Pj} = \sum$(Amount where Product = Pj)
Total sales: 43,900 USD
EMP019: 22,100 USD (50.3%)
EMP007: 8,500 USD (19.4%)
Product011: 17,500 USD (60.1%)
4. Constraint-Verifier Pairs Construction
Use predefined Template to transform Indicators to Constraint (C)-Verifier (V) pairs
Input: Statistical Indicators
Total sales: 43,900 USD
EMP019: 22,100 USD (50.3%)
Template of Overall Total
C1: "What is the totalsales this month?"
V1: "The total sales amount is {TotalSales}."
Template of Salesperson
C2: "Who is the top salesperson this month?"
V2: "{Salesperson} with {SalesAmount}, accounting for {Share}% of total sales."
Constraint 1
Verifier 1
Constraint 2
Verifier 2

Figure 3: Illustration of the LONGLEAVE evaluation pipeline. Attribute seeds define task scenarios, and the task generator creates long-form generation tasks paired with constraint–verifier sets. Model outputs are then evaluated by matching against verifiers with length and instruction-following checks.

the output is evaluated based on whether it correctly reflects the verifier V associated with the constraint C. Specifically, the generation process is modeled as shown in Eq. (1): Specifically, under CoV-Eval, the generation process is updated as shown in Eq. (3):

$$O_{\text{gen}} = \mathcal{L}(X_{\text{raw}}, I_{\text{task}}, \mathcal{C}), \tag{3}$$

then the quality S of O_{gen} is quantified by a task-specific scoring function $Score$, as shown in Eq. (4):

$$S = \text{Score}(O_{gen}, V). \tag{4}$$

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 task, where the Constraint–Verifier pair varies by task. We indicate Material as X_{raw}, Constraint as C, and Verifier as V. We use rule-based scripts to generate X_{raw}, C and V. For AP Style News Writing, we use LLM to generate news topics and statements. For Paragraph Reordering, original texts are from QreCC documents (Anantha et al., 2021).

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 a randomly generated runnable Python code, forming a polluted code (X_{raw}). The LLM is prompted to fix the code. The part code required repair is C. The repaired code can be automatically checked by Flake8 toolkit(V).

KG to Text Biography Generation (BioG). This task evaluates LLMs' ability to generate coherent and factual biographies based on given knowledge graph (KG) triples. The designed *knowledge graph generator* creates a large set of task relationships around a central person, then extracts triples (subject-predicate-object) and corresponding sentences starting from the nearest nodes. The evaluated model needs to incorporate all triples (C) into a fluent narrative within the specified word count. The verifier (V) 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 (SR). This task eval-

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. Red refers to LLM-as-a-Judge metrics. Blue indicates length scores. Purple represents other rule-based metrics.

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.

uates LLMs' ability to generate a sales report and answer predefined, specific questions based on a transaction table. We designed a *sales report generator* that synthesises the transaction table (X_{raw}), while generating natural language questions (C) and corresponding answers (V). Evaluation focuses on both coverage and accuracy of the answers.

AP Style News Writing (NW). This task evaluates LLMs' ability to write a news article following the Associated Press Stylebook (AP Style) (Goldstein, 1998). Given a news topic (X_{raw}), GPT-4o-2024-11-20 generates correct fact statements (V) together with corresponding flawed statements (C) that violate AP Style rules. The evaluated LLM is then required to write an article on the topic, integrating all statements in their correct form.

KV Dictionary Generation (KVG). 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 formatting rules (e.g., keys in uppercase with underscores); the query specifying the key–value pair and index is the *Constraint* (C), and a rule-based script verifies placement and formatting as the *Verifier* (V).

State Machine Simulation (SMS). This task requires simulating state transitions of a finite state machine (FSM) (Lee and Yannakakis, 1996) step by step. Here, the transition table serves as the raw material (X_{raw}), the initial state and input string constitute the *Constraint* (C), and an FSM validation script acts as the *Verifier* (V) by checking the generated sequence against the correct state transitions and signals. Models are evaluated by their match ratio and overall accuracy in reproducing all steps without errors.

Paragraph Reordering (PR). This task requires LLMs to reorder shuffled paragraphs (C) into the coherent sequence (V). The material consists of randomly sampled paragraphs, with the constraints being the shuffled order and the verifier 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 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(Bai et al., 2024b; Wu et al., 2025; Liu et al., 2024c; Que et al., 2024), 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 Figure 4.

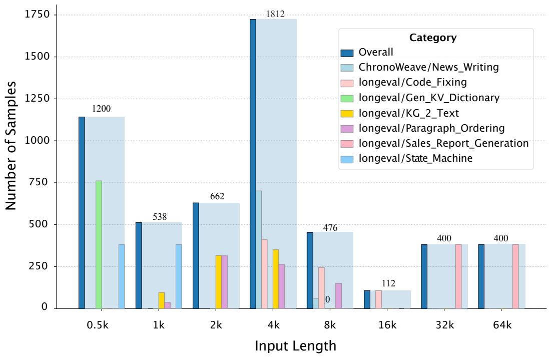


Figure 4: Input length distribution of LongWeave

2.5 Output Length Control

LongWeave controls target output length through a multi-faceted strategy. First, the scale and complexity of input materials—such as the number of data records or code lines—are procedurally generated to be proportional to each target length tier (1K, 2K, 4K, 8K tokens). Second, prompts provide models with explicit instructions specifying the desired output length. Finally, the evaluation protocol enforces these constraints by directly penalizing length deviations.

2.6 Evaluation Metrics

The evaluation metrics include LLM-as-a-Judge for CoV-Eval, length-related metrics, and others. Each task's final score is the harmonic mean of sub-metrics, ensuring that poor performance in one area cannot be offset by better performance in another.

LLM-as-a-Judge use LLM-as-a-Judge to check whether verifiers, corresponding to defined constraints, 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 rely on deterministic code to verify correctness, such as code runnability, existence and placement of target key–value pairs, and Kendall's Tau for paragraph reordering.

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, QWQ-Plus) (Yang et al., 2024a), and the newer Qwen3 series (4B, 8B, 14B-Think/Non-Think, 32B-Think/Non-Think) (Yang et al., 2025a). Additionally, we evaluated DeepSeek-V3 (Liu et al., 2024a). The *commercial models* include GPT-4o-2024-11-20 (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 deployment on 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-a-Judge evaluations. To control output length, we adjust the configuration as illustrated by the "gen_len" configurations in Table 2. Furthermore, LongWeave supports task difficulty control 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 standard 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.

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, models in the top 5 are 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-Instruct	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-Instruct	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-Instruct	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-Instruct	33.24	76.38	24.47	28.25	33.32	6.20	67.88	48.65	37.55	37.22	30.72	38.53
Llama4-17b-128e-Instruct	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-Instruct	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-Instruct	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-Instruct	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-Instruct	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
Qwen3-14B-Think	45.41	78.71	28.59	23.86	42.92	10.64	89.30	52.33	49.96	43.96	36.30	45.64
Qwen3-32B-Think	59.69	83.64	**35.67**	20.86	52.40	13.89	88.38	57.71	56.89	49.54	38.45	50.65
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**

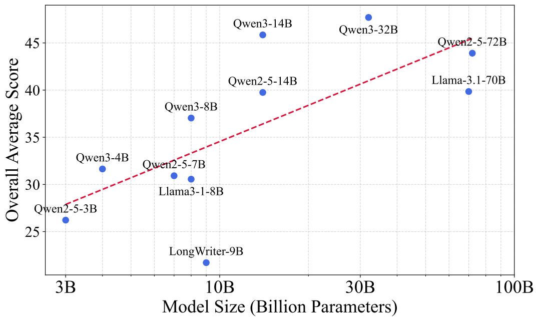


Figure 5: Performance of different model sizes

Table 4: Distribution of failure patterns across 1,400 analyzed samples, grouped by category.

Failure Pattern	**Count**	**Share**
Instruction-following errors		
Selective instruction execution	375	30.4%
Stepwise deviation	172	14.0%
Incomplete factual coverage	156	12.7%
Length control issues	153	12.4%
Numerical errors		
Calculation failures	123	10.0%
Content problems		
Fabricated facts	27	2.2%
Redundancy / looping	17	1.4%
Reasoning-specific failures		
Failure to terminate reasoning	210	17.0%

Existing models struggle in long form generation. Frontier proprietary models demonstrate the best performance. DeepSeek-R1, Gemini-2.0-flash, and o3-mini-2025-01-31 achieve nearly 60% performance at 1k length, but when generating 8k, the performance drops to around 40%. GPT-4o-2024-11-20 only achieves 42.99% while it tends to generate short responses.

Increasing model size 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 5, where the regression curve shows a positive correlation between the two.

Performance of Reasoning Models on Long-Sequence Generation. The very large reasoning models (e.g., DEEPSEEK-R1) perform strongly on long-sequence generation tasks (Table 3). In contrast, smaller reasoning models often fail to terminate the reasoning phase, repeatedly generating large chunks of the input, which leads to truncation.

Performance Degradation when Input Context is Long. As shown in Table 3, the quality of long outputs deteriorates significantly with longer inputs. This is especially evident in tasks like Sales Report Analysis and writing AP-style News Articles, which require handling long materials and detailed guidelines. Despite this challenge, managing both long inputs and outputs is crucial for practical applications. By incorporating more relevant information into the input window, hallucinations can be

Table 5: 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	$28.92_{\pm 0.30}$	$29.04_{\pm 0.35}$	$29.10_{\pm 0.40}$	$29.30_{\pm 0.35}$	$29.80_{\pm 0.01}$	$29.60_{\pm 0.10}$	$29.80_{\pm 0.15}$	$29.69_{\pm 0.10}$	$30.05_{\pm 0.12}$	$30.13_{\pm 0.11}$

minimized, offering a key direction for optimizing long-sequence generation models.

3.4 Failure Pattern

To better understand where models fail, we analyzed 1,400 samples across seven tasks and four target lengths (1k, 2k, 4k, and 8k; 200 samples per task and 50 per length). Outputs were generated with `Qwen3-32B` in think mode (without thinking budgets). Failure types were first labeled with `GPT-5-2025-08-07` and then manually checked. We observed eight common failure patterns, which we group into four categories (Table 4).

Instruction-following errors are the most common. Selective instruction execution (30.4%) means models handle the easy parts of a prompt but ignore the harder constraints. Stepwise deviations (14.0%) show that the ability to follow instructions gets worse as the output becomes longer. Incomplete factual coverage (12.7%) often happens in structured tasks like BIOG, where some required facts are dropped. Length control issues (12.4%) also appear, with outputs not matching the requested length.

Numerical errors (10.0%) mostly occur in quantitative tasks like SR, where models miscalculate percentages, averages, or growth rates. These errors suggest that reliable number handling may require tool support.

Content problems include fabricated facts (2.2%), where models add unsupported information, and redundancy or looping (1.4%), where outputs repeat content or drift into filler text.

Reasoning-specific failures are unique to reasoning models. In 17.0% of cases, the reasoning phase did not stop: models produced very long "thinking" traces, often repeating large chunks of the input, leaving too little budget for the final answer and causing truncation.

Table 6: Evaluation of LLM-as-Judge Stability in CoV-Eval 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 Analysis

4.1 Stability of the Benchmark

To assess the stability of the benchmark, we conducted multiple experiments using the Llama-3.1-8B model with varying sample sizes (20-200), as shown in Table 5. 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 Stability 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 test the reliability, we used other LLMs as evaluators. The Llama-3.1-8B model was tested as the evaluated model on 100 samples with different evaluation models. As shown in Table 6, the results revealed a performance fluctuation variance of 0.45 for models like DeepSeek-V3, GPT-4o-2024-11-20, and o3-mini-2025-01-31.

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

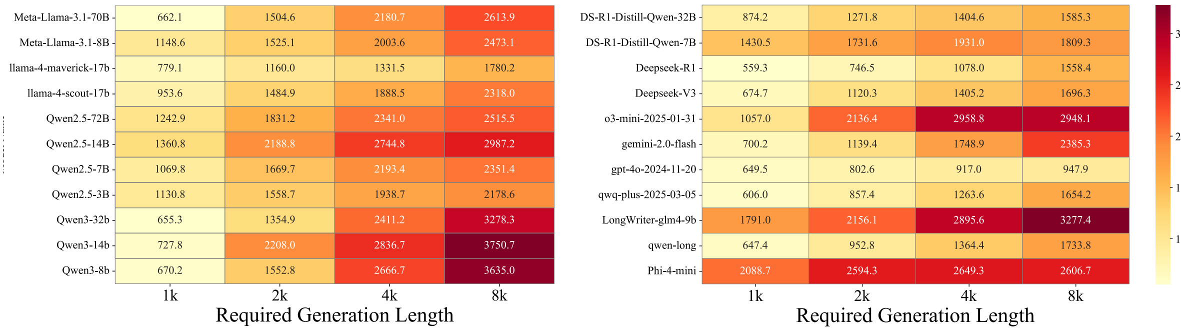


Figure 6: The heatmap visualizes the actual output lengths across four target length settings (1k, 2k, 4k, and 8k tokens). Each row represents the model, while the column corresponds to the length.

Figure 6. 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 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 Increasing the Context Window Does Not Necessarily Improve Long Generation

We compared the performance of the Qwen2.5-14B and 7B models with a 1M context window version, as shown in Figure 7, *there was little difference in overall scores*: 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. It indicates that although long input models and long generation models share the same model structure, the performance is inconsistent due to the training data.

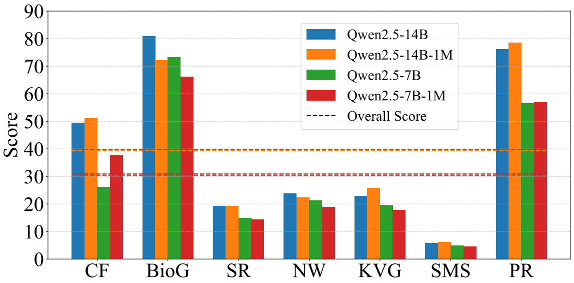


Figure 7: Input length distribution of LongWeave

5 Conclusion

Evaluating long, constrained LLM outputs is challenging. We introduce LongWeave, featuring CoV-Eval 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 CoV-Eval contain several limitations that should be acknowledged: **High Cost for Inference.** 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. **High Cost LLM-as-a-Judge.** Several tasks within LongWeave rely on large LLMs (e.g., Qwen2.5-72B-Instruct) as judges, which adds significant computational overhead and cost. **Limited Coverage of Creative Tasks.** LongWeave currently focuses on factual accuracy and structural correctness, but it could be expanded to better assess creative tasks.

Acknowledgments

This work is supported by the National Key R&D Program of China (Grant No. 2024YFC3308304), the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (Grant No. 2025C01128), the National Natural Science Foundation of China (Grant No. 62476241), the Natural Science Foundation of Zhejiang Province, China (Grant No. LZ23F020008), the Zhejiang Zhejiang Key Laboratory of Medical Imaging Artificial Intelligence.

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*.

Raviteja Anantha, Svitlana Vakulenko, Zhucheng Tu, Shayne Longpre, Stephen Pulman, and Srinivas Chappidi. 2021. Open-domain question answering goes conversational via question rewriting. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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*.

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. 2025. Copyspec: Accelerating llms with speculative copy-and-paste without compromising quality. *arXiv preprint arXiv:2502.08923*.

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*.

Xiaonan Wang, Bo Shao, and Hansaem Kim. 2025. Toward reliable vlm: A fine-grained benchmark and framework for exposure, bias, and inference in korean street views. *arXiv preprint arXiv:2506.03371*.

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, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Wang, and Liangming Pan. 2025c.

How is llm reasoning distracted by irrelevant context? an analysis using a controlled benchmark. *arXiv preprint arXiv:2505.18761*.

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

A.1 Related Work

Long Input and Output Models Recent advancements in 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, with parallel efforts focused on improving inference efficiency (Dumitru et al., 2025).

Long Generation Benchmarks Long generation benchmarks typically rely on similarity-based metrics like α-nDCG and SelfBLEU, or LLM-as-a-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, highlighting the need for more fine-grained and specialized benchmarks across different domains (Wang et al., 2025; Yang et al., 2025c).

A.2 Answer Length

Our analysis of generated output lengths, detailed in Table 7, reveals discrepancies between instructed and actual token counts across all evaluated models. Key findings are as follows: First, most models exhibit poor adherence to explicit length constraints, with the deviation increasing for longer targets (4k and 8k tokens). Second, distinct patterns emerge based on model type. Reasoning-oriented models (`DeepSeek-R1`) and certain proprietary models (`GPT-4o`) consistently produce outputs substantially shorter than requested. In contrast, the `Qwen3` series demonstrates more effective length control than its predecessor, `Qwen2.5`. Crucially, we find no direct correlation between output length and overall task performance. Verbosity does not equate to higher quality, as many models that generate longer text achieve lower scores on our benchmark's core metrics.

A.3 AP Style Criteria

In our AP Style News Writing task, we assess a model's proficiency in adhering to complex, real-world stylistic guidelines from the Associated Press

Table 7: Average output length for each model across four target length settings, highlighting models' adherence to length constraints.

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
Phi-4-mini-instruct	2088.7	2594.3	2649.3	2606.7
Qwen2.5-14B-Instruct	1360.8	2188.8	2744.8	2987.2
Qwen2.5-14B-Instruct (1M)	913.2	1477.3	1678.4	2092.7
Qwen2.5-3B Instruct	1130.8	1558.7	1938.7	2178.6
Qwen2.5-72B Instruct	1242.9	1831.2	2341.0	2515.5
Qwen2.5-7B Instruct	1069.8	1669.7	2193.4	2351.4
Qwen2.5-7B Instruct (1M)	933.1	1370.8	1666.8	2087.3
DeepSeek-R1	559.3	746.5	1078.0	1558.4
SeepSeek-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

(AP) Stylebook. To ensure an objective and verifiable evaluation, we moved beyond holistic review and focused on verifiable rules. The creation of our test cases was guided by ten distinct categories of AP Style rules, as detailed in Figure 8. For each category, we generated Constraint-Verifier pairs where the Constraint is a factual statement deliberately crafted to violate a specific rule (e.g., writing "7 apples" instead of "seven apples"). The corresponding Verifier is the same statement, corrected to be fully compliant with AP style. The model is then tasked with incorporating the factual information from the incorrect Constraint into its generated article, but in the stylistically correct form.

A.4 Details of sample construction

CSV Sales Report Analysis (SR): 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.

Code Fixing with Flake8 Compliance (CF): 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 instances for static analysis tools

Table 8: Configuration of large language models, their backends and decoding parameters provided by different suppliers

Provider	**Backend**	**Model**	**`Decoding Parameters`**
Meta	vLLM	Meta-Llama-3.1-8B-Instruct / Meta-Llama-3.1-70B-Instruct	`temperature: 0.7, top_p: 0.8, max_tokens: 8192, stream: False`
	Aliyun Dashscope	llama-4-scout-17b-16e-instruct	`temperature: 0.7, max_tokens: 8192, stream: True`
		llama-4-maverick-17b-128e-instruct	`temperature: 0.7, max_tokens: 8192, stream: True`
OpenAI	OpenAI API	gpt-4o-2024-11-20	`temperature: 0.7, max_tokens: 8192, stream: True`
		o3-mini-2025-01-31	`temperature: 0.7, max_tokens: 8192, stream: True`
		gpt-4o-mini-2024-07-18	`temperature: 0.7, max_tokens: 8192, stream: True`
Google	OpenAI API	gemini-2.0-flash	`temperature: 0.7, max_tokens: 8192, stream: True`
	vLLM	gemma-3-12b-it / gemma-3-27b-it	`temperature: 0.7, max_tokens: 8192, stream: False`
Zhipu AI	vLLM	LongWriter-glm4-9b	`max_tokens: 8192, stream: False`
Microsoft	vLLM	Phi-4-mini-instruct	`max_tokens: 8192, stream: False`
DeepSeek	Aliyun Dashscope	deepseek-v3 / deepseek-r1	`temperature: 0.7, max_tokens: 8192, stream: True`
	vLLM	DeepSeek-R1-Distill-Qwen-32B	`max_tokens: 8192, stream: False`
Alibaba	vLLM	Qwen2.5-14B-Instruct-1M	`max_tokens: 8192, stream: False`
	Aliyun Dashscope	qwen-long	`temperature: 0.7, max_tokens: 8192, stream: True`
	vLLM	Qwen2.5-3B/7B/14B/72B-Instruct	`temperature: 0.7, top_p: 0.8, max_tokens: 8192, stream: False`
	vLLM	qwen3-14b-r	`temperature: 0.7, top_p: 0.8, max_tokens: 32768, stream: False`
	Aliyun Dashscope	qwen3-14b	`temperature: 0.7, top_p: 0.8, max_tokens: 8192`
	vLLM	qwen3-32b	`temperature: 0.7, top_p: 0.8, max_tokens: 8192, stream: False`
	Aliyun Dashscope	qwen3-8b / qwen3-14b / qwen3-32b	`temperature: 0.7, max_tokens: 8192, stream: True`
	Aliyun Dashscope	QWQ-plus / qwen-max series	`temperature: 0.7, top_p: 0.8, presence_penalty: 1.5, max_tokens: 8192, stream: True`

"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 8: Generation and Evaluation Rules for Constraints in the News Writing Task. Ten dimensions, each containing rules, positive examples, and error cases that guided the creation of verifiable test instances.

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 iterator 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.

KG to Text Biography Generation (BioG): 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, Entrepreneur), 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 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.

AP Style News Writing (NW): 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 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.

A.5 Evaluation Efficiency

Given the scale of our evaluation (23 models across 5,600 samples), understanding these efficiency aspects is crucial.

For this analysis, we utilized two nodes, each equipped with 8 NVIDIA A100 GPUs (totaling 16). One node was dedicated to deploying the evaluator model, Qwen2.5 72B, while the other hosted the model under test, Llama3.1 8B. To establish a clear baseline, we measured performance in a single-threaded mode. We recorded the inference and evaluation times for a single sample across seven distinct tasks, varying the input context length at four levels: 1k, 2k, 4k, and 8k tokens.

The results of this single-threaded performance analysis are presented in Table 9. As shown, the evaluation time can be a significant component of the total processing time, particularly for tasks like News Writing (NW), which involve complex rubric-based assessments. For tasks such as Knowledge Graph (KVG) and Sales Message (SMS), the

Table 9: Efficiency analysis of the evaluation pipeline. Each value represents the time in seconds to process a single sample in single-threaded mode. The analysis was conducted using Qwen2.5 72B as the evaluator and Llama3.1 8B as the model being tested.

Metric (Input Length)	CF	BioG	SR	NW	KVG	SMS	PR
Inference (1k)	29.72	20.58	16.29	45.10	139.20	134.95	35.99
Evaluation (1k)	7.31	9.38	4.54	87.04	0.00	0.00	0.01
Inference (2k)	28.86	16.49	24.63	22.44	129.35	133.46	23.75
Evaluation (2k)	5.34	2.42	8.89	120.05	0.00	0.00	0.00
Inference (4k)	41.12	121.53	16.04	32.95	161.03	146.42	31.42
Evaluation (4k)	2.54	20.13	12.17	155.70	0.00	0.00	0.01
Inference (8k)	106.56	33.62	36.04	32.91	193.31	149.92	90.09
Evaluation (8k)	5.49	30.32	12.96	1439.32	0.00	0.00	0.01

evaluation is nearly instantaneous as it relies on simple keyword matching. The full names for the task abbreviations can be found in Table 2 of the main paper.

To optimize efficiency in our main experiments, we employed parallel processing. For inference, we used 16 parallel threads. For the evaluation stage, we utilized 5 parallel threads with appropriate batch sizes tailored to the task (e.g., a batch size of 5 for Sales Report and Knowledge Graph tasks, and 10 for AP Style News). The evaluation pipeline ran on eight A100 GPUs, while inference was performed either on a separate set of eight A100 GPUs for local models or via API calls for proprietary models. This parallelized setup significantly reduced the overall wall-clock time required for our large-scale benchmark.

A.6 Additional Data Examples

To provide a concrete understanding of the tasks within the LongWeave benchmark, this section presents illustrative examples of the input prompts used during evaluation. These examples showcase the structure of the input materials, the detailed instructions, and the specific constraints that models must follow. The figures below cover all seven tasks: KG to Text Biography Generation (Figure 9); Code Fixing (Figure 10 and 11); AP Style News Writing, which includes the topic, factual statements, and style rubric (Figure 12,13, and14); Paragraph Reordering (Figure 15); State Machine Simulation and KV Dictionary Generation (Figure 16); and CSV Sales Report Analysis, which details the data and questions (Figure 17 and 18). Collectively, these examples demonstrate the diversity of challenges posed by LongWeave and provide insight into the practical implementation of CoV-Eval.

A.7 Details of Task Generator

To ensure transparency and reproducibility, all test samples in LongWeave are synthetically generated through rule-based pipelines. The generation process for each task, illustrated in Figures 19 to 25, follows a consistent three-stage process. First, **Attribute Sampling** defines the core parameters and complexity of each task instance. Second, **Joint Generation** uses these attributes to procedurally create the aligned triad of `Material` (X_{raw}), `Constraint` (C), and `Verifier` (V). Finally, the third stage in each figure provides a **concrete example** of the generated `Material`, `Constraint`, and `Verifier`.

Example of Task: KG to Text Biography Generation Generation (BioG)

Model Input

Role: Biographer / Content Write

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):

- A Manifesto for Incubate Impactful E-Markets - authored by - Ambassador Cathy Allen
- A Manifesto for Incubate Impactful E-Markets - publication year - 1874
- Ambassador Cathy Allen - participated in - Major Promotion (1821)
- Ashley Bell - birth year - 1961
- Barry Ballard - birth year - 1979
- Barry Ballard - job - Merchandiser, retail
- Barry Ballard - socioeconomic background - Middle Class

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:

LLM-as-a-Judge

```
eval_prompt = f"""
**Task:** Evaluate if the core factual information conveyed by each numbered 'Target Sentence' below
is accurately and adequately covered or represented, either directly or semantically, within the
provided 'Generated Biography Text'.

    **Generated Biography Text:**
    --- START BIOGRAPHY ---
    {biography_text}
    --- END BIOGRAPHY ---

    **Target Sentences (Facts to find):**
    {numbered_sentences}

    **Evaluation Criteria:**
    For each numbered target sentence (from 1 to {batch_size}), determine if its essential factual
statement is present in the 'Generated Biography Text'. Exact wording is not required, but the core fact
must be included in the biography. Judge based *only* on the presence of the information, not the
writing style or fluency. Answer 'true' if the fact is present, 'false' otherwise.

    **Output Format:**
    Respond ONLY with a single JSON list containing boolean values (true/false), corresponding *in
order* to the numbered Target Sentences (1 to {batch_size}). The list must have exactly {batch_size}
elements. Do not include any explanations or other text outside the JSON list.

    **Example Output (if batch_size was 3):**
    [true, false, true]

    **Your JSON Output:**
    """
```

Figure 9: An illustrative example for the KG to Text Biography Generation (BioG) task.

Example of Task: Code Fixing with Flake8 Compliance (CF)

Model Input

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 isrunnable 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:

––– START OF CODE –––

```
def   LoadDataSource(source_path):
    """Retrieve and parse input stream."""
    print('Parse Data:',sourcepath)
    raw_data_struct = [
        {
            'handle':-35,
            'id'  :   'ID_100',
            'fetch':'OtSAP5Bn0',
            'generate_log':False,
            'save':False
        },
...

def MainEntryPoint():
    InputDataset=LoadDataSource('./data/source.json')
    filtered_data=ValidateRecords(InputDataset)
    computed_metrics=CalculateStats(filtered_data)
    print_summary_report(computed_metrics)
    UpdateGlobalState()  # Modify global state
    unusedRecord=None
    LongVariableName838 = 'result status product status event cache record log state result id user report status cach'
    CalculateStats_1(update_global_flag) # Call existing func

if name == " main "
    MainEntryPoint()
  】
```

Figure 10: An illustrative example for the Code Fixing (CF) task (Part 1/2).

Example of Task: Code Fixing with Flake8 Compliance (CF)

LLM-as-a-Jude

```
eval_prompt = f"""
**Task:** Evaluate if 'FIXED CODE' is a relevant and complete code response to the 'ORIGINAL CODE'.
**Evaluation Criteria:**

1.  **Content Relevance (Answering the Request):**
    *   **Relevant (true):** The 'FIXED CODE' directly attempts to modify, fix, or refactor the provided
'ORIGINAL CODE'. It addresses the implicit request to correct or improve the original snippet.
    *   **Not Relevant (false):** The 'FIXED CODE' does not address the original code. It might be
unrelated code, a refusal, a question, an explanation *instead* of code, or generic template/placeholder
code not adapted to the original.

2.  **Code Completeness (Providing Full Code):**
    *   **Complete (true):** The 'FIXED CODE' provides a full, runnable (or intended to be runnable)
Python code snippet that represents the proposed modification or fix. It's not just a fragment, comment,
or instruction.
    *   **Incomplete (false):** The 'FIXED CODE' is not a complete code solution. It might be:
        *   Only an explanation or commentary about the fix.
        *   A code fragment (e.g., only a single corrected line without context).
        *   Instructions on how to fix the code (e.g., "You should change line 5 to...").
        *   An empty response or placeholder like "# [Your corrected code here]".

**Overall Judgment:**
 Return `true` ONLY IF BOTH criteria (Content Relevance AND Code Completeness) are met. Otherwise, return
`false`.

**ORIGINAL CODE:**
xxx

**FIXED CODE:**
xxx

**Output:** Respond ONLY with JSON: {{"is_relevant_and_complete": true/false}}
"""
```

Figure 11: An illustrative example for the Code Fixing (CF) task (Part 2/2).

Example of Task 6: AP Style News Writing (NW)

Model Input (Part 1)

Write a news report titled 'BioLumina Forests: A Bold Experiment in Genetic Engineering and Environmental Restoration.' Cover:

1. The Project : Introduce the fictional nation of 'Aurora' and its groundbreaking 'BioLumina Forest Initiative,' which uses genetically modified plants to emit natural light, reduce energy consumption, and combat climate change.
2. The Science : Explain how fluorescent proteins from deep-sea organisms are used to create glowing trees that absorb CO2, purify air, and glow softly at night, creating a surreal, dreamlike environment.
3. Controversy : Highlight debates over potential ecological risks, including disruption of native species, unintended consequences for food chains, and concerns about over-reliance on technological fixes.
4. Pilot Results : Discuss early successes in degraded areas—improved air quality and tourism—but note declines in local wildlife populations, raising questions about habitat impact.
5. Philosophical Reflection : Explore the broader implications of using advanced technology to restore nature—does it represent progress or hubris? Is humanity truly ready to manage such interventions responsibly?

Creative Details: Use elements like 'Aurora,' 'BioLumina Forest,' and vivid imagery of glowing landscapes to craft an engaging narrative balancing hope with caution.

You MUST strictly adhere to the AP News Style guidelines provided below.
Your article will be evaluated on two equally-weighted dimensions: (1) Recall** of ALL required information, and (2) Compliance with the AP Style rules.**

IMPORTANT: Each AP Style category includes example sentences that violate its rules. Rewrite and include all of them in your article, following AP Style and keeping their meaning. Missing or uncorrected items will reduce your score.

=== CLARITY AND BREVITY ===
Scoring Criteria:

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

Incorrect Examples:

- 'The aforementioned individual arrived at the location.'
- 'This is a highly technical subject matter.'

Correct Examples:

- 'The person arrived at the site.'
- 'This is a technical topic.'

Content Requirements for 'Clarity and Brevity': Include and Rewrite EACH of the following statements to comply with the 'Clarity and Brevity' AP Style guidelines detailed above.

1. The BioLumina Forest Initiative, launched by the fictional nation of Aurora, uses genetically modified plants to emit natural light, reduce energy consumption, and combat climate change.
2. The glowing trees, created by inserting fluorescent proteins from deep-sea organisms, not only absorb CO2 but also purify the air, creating a surreal, dreamlike landscape that glows softly at night.
3. While the BioLumina project has shown early success in improving air quality and boosting tourism, it has also raised concerns about declines in local wildlife populations.
4. Critics argue that the project could disrupt local ecosystems by introducing genetically modified species, potentially upsetting the balance of food chains.
5. In addition to its environmental benefits, the glowing trees offer an innovative solution to reducing energy use, but they also spark debates on the ethics of genetic engineering.
6. The BioLumina Forest project represents a bold attempt to blend environmental restoration with cutting-edge technology, but its long-term effects on biodiversity remain uncertain.

Figure 12: An illustrative example for the AP Style News Writing (NW) task (Part 1/3).

Example of Task 6: AP Style News Writing (NW)

Model Input (Part 2)

7. Supporters of the initiative believe it could provide a model for sustainable development, but critics worry it may be a technological fix that overshadows deeper ecological issues.
8. As the project expands, the need for careful oversight grows, as many question whether humanity is ready to manage such large-scale genetic interventions in nature.
9. While the BioLumina initiative holds promise, it brings to light the ongoing struggle between technological progress and the need for responsible environmental stewardship.
10. Ultimately, the future of the BioLumina Forests depends on finding a balance between technological innovation and environmental conservation, a challenge that requires global cooperation and careful planning.

=== PUNCTUATION ===

Scoring Criteria:

- Quotation marks:
  Periods and commas always go inside quotation marks.
- Oxford comma:
  Avoid using the Oxford comma in lists.
- Space rules:
  Use only one space after a period.
- Colons:
  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'.
- Red, white, and blue.
- This is a sentence. Another one follows.
- 'The following: rules.'

Correct Examples:

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

Content Requirements for 'Punctuation': Include and Rewrite EACH of the following statements to comply with the 'Punctuation' AP Style guidelines detailed above.

11. The nation of Aurora unveiled its BioLumina Forest Initiative, a project that uses glowing trees to reduce energy consumption and combat climate change.
12. Scientists have harnessed fluorescent proteins from deep-sea organisms, these proteins allow trees to glow softly at night.
13. Critics argue that the project could disrupt native species, harm food chains, and create an over-reliance on technology.
14. Early results show improved air quality and increased tourism, however, local wildlife populations have declined.
15. The glowing forests of Aurora are described as surreal dreamlike environments that captivate visitors.
16. Does this bold experiment represent progress or hubris? The answer remains unclear.
17. Aurora's BioLumina Forests emit light absorb CO2, and purify the air all at once.
18. Local residents report mixed feelings about the initiative, some praise its innovation while others fear ecological risks.
19. Tourists flock to see the glowing landscapes, they describe them as otherworldly and awe-inspiring.
20. The philosophical debate centers around whether humanity can responsibly manage such interventions, the stakes are high.

Length Specifications:

- TARGET WORD COUNT: Aim foraround 1024 words.

Begin writing your ~1024-word AP-style article below:

Figure 13: An illustrative example for the AP Style News Writing (NW) task (Part 2/3).

Example of Task 6: AP Style News Writing (NW)

Model Output

The nation of Aurora unveiled its BioLumina Forest Initiative, a project that uses glowing trees to reduce energy consumption and combat climate change. The initiative, launched by the fictional country, has sparked global interest and controversy. Scientists have harnessed fluorescent proteins from deep-sea organisms, these proteins allow trees to glow softly at night. The glowing trees, created by inserting fluorescent proteins from deep-sea organisms, not only absorb CO2 but also purify the air, creating a surreal, dreamlike landscape that glows softly at night. The BioLumina Forest Initiative, launched by the fictional nation of Aurora, uses genetically modified plants to emit natural light, reduce energy consumption, and combat climate change.

Aurora, a fictional nation known for its commitment to environmental innovation, has taken a dramatic step toward sustainability with its BioLumina Forests. These forests, unlike any seen before, are not just green spaces but glowing …

LLM-as-a-Judge

```
evaluation_prompt = (
        "Please read the article and complete the EVALUATION TASK below:\n\n"

        f"=== ARTICLE CONTENT ===\n{response}\n\n"

        f"=== SCORING CRITERIA FOR '{category}' ===\n"
        f"Scoring Criteria:\n{rubric['Scoring_Criteria']}\n"
        f"Incorrect Examples:\n{rubric['Incorrect_Examples']}\n"
        f"Correct Examples:\n{rubric['Correct_Examples']}\n\n"

        "EVALUATION TASK:\n"
        "For each statement listed below, perform the following evaluations:\n"

        "1. Determine whether the statement exists in the article (verbatim or semantically equivalent).
If it exists, extract the exact matching content from the article.\n"
        "2. If the statement exists, determine whether it follows the AP rules as per the scoring
criteria.\n"
        "3. Provide clear reasoning for your evaluation.\n\n"

        "Output format (JSON):\n"
        "{\n"
        '  {\n'
        '    "statement_id": "Unique ID of the statement",\n'
        '    "statement": "Original statement",\n'
        '    "matched_content": "Exact matching content from the article (or empty string if not
found)",\n'
        '    "thinking": "Explanation of the evaluation process and reasoning",\n'
        '    "exists_in_article": true/false,\n'
        '    "follows_rules": true/false\n'
        '  },\n'
        '  ...\n'
        "}\n\n"
        "Do NOT include any additional text or explanations outside the JSON array.\n"
        "Ensure that the JSON is valid and can be directly parsed by a JSON parser.\n\n"

        "STATEMENTS TO EVALUATE:\n"
    )

    for i, stmt in enumerate(stmts, start=1):
        evaluation_prompt += (
            f"Statement {i}: {stmt['Statement']}\n"
            f"   Why This Is Incorrect: {stmt['Reason_for_Deduction']}\n"
            f"   How It Should Be Written: {stmt['Correct_Expression']}\n\n"
        )

    evaluation_prompt += "Please provide the evaluation results."
```

Figure 14: An illustrative example for the AP Style News Writing (NW) task (Part 3/3).

Example of Task 7: Paragraph Reordering (PR)

Model Input

Please rearrange the following paragraphs into a logically coherent article:

[[Segment 0]]
In the end, implied powers was used as justification for finishing the deal. [3] Later, directly borrowing from Hamilton, Chief Justice John Marshall invoked the implied powers of government in the United States Supreme Court case, McCulloch v. Maryland. [4] In 1816, the United States Congress passed legislation creating the Second Bank of the United States. The state of Maryland attempted to tax the bank. The state argued the United States Constitution did not explicitly grant Congress the power to establish banks. In 1819, the Court decided against the state of Maryland. Chief Justice Marshall argued that Congress had the right to establish the bank, as the Constitution grants to Congress certain implied powers beyond those explicitly stated. In the case of the United States Government, implied powers are powers Congress exercises that the Constitution does not explicitly define, but are necessary and proper to execute the powers. The legitimacy of these Congressional powers is derived from the Taxing and Spending Clause, the Necessary and Proper Clause, and the Commerce Clause. Implied powers are those that can reasonably be assumed to flow from express powers,[5] though not explicitly mentioned. This theory has flown from domestic constitutional law[6] to International law,[7] and European Union institutions have accepted the basics of the implied powers theory. [8] ^ They implied powers into the united states. . . . . . .

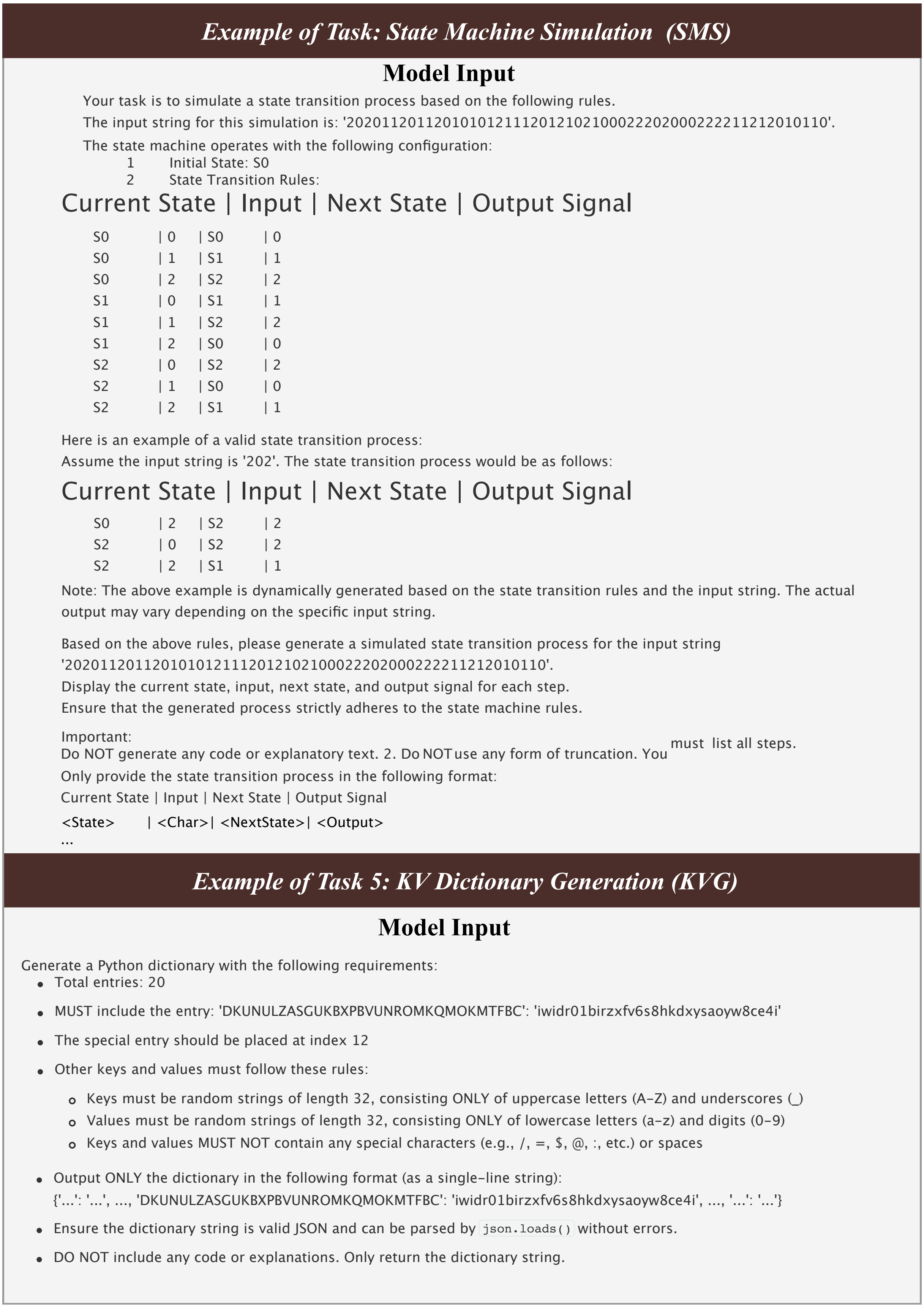
[[Segment 1]]
Language links are at the top of the page across from the title. What links hereRelated changesUpload fileSpecial pagesPermanent linkPage informationCite this pageWikidata item In the United States, implied powers are powers that, although not directly stated in the Constitution, are implied to be available based on previously stated powers. When George Washington asked Alexander Hamilton to defend the constitutionality of the First Bank of the United States against the protests[1] of Thomas Jefferson, James Madison, and Attorney General Edmund Randolph, Hamilton produced what has now become the doctrine of implied powers. [2] Hamilton argued that the sovereign duties of a government implied the right to use means adequate to its ends. Although the United States government was sovereign only as to certain objects, it was impossible to define all the means it should use, because it was impossible for the founders to anticipate all future exigencies. Hamilton noted that the "general welfare clause" and the "necessary and proper clause" gave elasticity to the Constitution. Hamilton won the argument and Washington signed the bank bill into law. Another instance of the usage of implied powers was during the Louisiana Purchase, where, in 1803, the United States was offered $15 million for French territory. James Monroe was sent by Thomas Jefferson to France to negotiate, with permission to spend up to $10 million on the port of New Orleans and parts of Florida. However, a deal to purchase the entirety of French territory in the United States for $15 million was reached, even though this exceeded the given amount of $10 million. Although the decision was very popular and widely praised, it was unknown whether or not Jefferson had the power to negotiate the territory without the permission of Congress.

[[Segment 2]]
^ Also outside President and Congress: for the Judiciary, see Incidental or Implied Powers of Federal Courts, by Harris, Robert Jennings, Chapter II, 1 Judicial Power of the United States (1940). ^ Especially in the common law legal community: see Sagar Arun, Notes towards a Theory of Implied Powers in (Indian) Constitutional Law, NUJS Law Review, Vol. 7, Issue 3-4 (2014), pp. 249-262. ^ International Legal Personality and Implied Powers of International Organizations, by Rama-Montaldo, Manuel, British Yearbook of International Law, Vol. 44, pp. 111-156 (1970). ^ Andrea Giardina, Rule of Law and Implied Powers in the European Communities, The Italian Yearbook of International Law, Vol. 1, pp. 99-111. Categories: Constitutional lawDeductive reasoningLegal doctrines and principlesHidden categories: Articles with short descriptionShort description matches WikidataArticles with J9U identifiersArticles with LCCN identifiers This page was last edited on 2 March 2023, at 12:05(UTC). additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc. , a non-profit organization.

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.

Figure 15: An illustrative example for the Paragraph Reordering (PR) task.



Example of Task: State Machine Simulation (SMS)

Model Input

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:
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>
...
```

Example of Task 5: KV Dictionary Generation (KVG)

Model Input

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 16: Illustrative examples for the State Machine Simulation (SMS) and KV Dictionary Generation (KVG).

Example of Task: CSV Sales Report Analysis (SR)

Model Input

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):

OrderID	OrderDate	Region	City	SalespersonID	SalespersonName	SalespersonTarget	CustomerID	CustomerName	IsNewCustomer	ProductID	...
ORD-2025-04-00001	2025-04-07	East Region	Philadelphia	EMP008	Meghan Rodriguez	130000	CUST-9151-371	Tran-Griffith	False	PROD-O01	...
ORD-2025-04-00002	2025-04-29	East Region	Pittsburgh	EMP009	Danielle Hart	93000	CUST-3153-277	Mullen, Johnson and Sosa	False	PROD-S03	...
...											

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 5: Which product had the highest sales volume in units?

... (and 15 more questions)

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:

Model Output

In April 2025, the East Region generated $1,526,782.59 in revenue from 200 transactions, with an average deal size of $7,633.91. Performance varied greatly among staff: Kimberly Odonnell MD (EMP016) led with $195,079.78 in sales, while Jeffrey Wagner (EMP018) was the lowest performer with $59,904.92. Only 6 of 20 reps met their targets. The top product by revenue was the 'Network Switch Pro' ($143,438.37), with hardware and software being the dominant sales categories...

LLM-as-a-Jude

Task: Evaluate the 'SALES REPORT' against a list of 'Question/Target Answer' pairs.
For each item, determine:

1. Answered: Did the reportattemptto answer the specific 'Question'? (true/false)
2. Correct: If the question was answered, does the answer provided in the report align with the 'Target Answer'? (true/false). If the question was not answered, this MUST be false.

Figure 17: An illustrative example for the CSV Sales Report Analysis (SR) task (Part 1/2).

Example of Task: CSV Sales Report Analysis (SR)

Evaluation Guidance:

- Focus on the substance of the question and answer, not exact wording.
- 'Answered' means the report addresses the core topic of the question, even briefly.
- 'Correct' means the information given in the report matches the meaning of the 'Target Answer'. Minor phrasing differences are acceptable. Numerical values should be reasonably close if applicable.
- If the report doesn't mention the topic of the question at all, 'answered' is false and 'correct' is automatically false.

Question/Target Answer Pairs:
--- START PAIRS ---
Item 1:
Question: Who was the top sales representative by revenue and what was their contribution?
Target Answer: Chris Allen (EMP014) led the team with USD 948,895 in sales (31.6% of total).

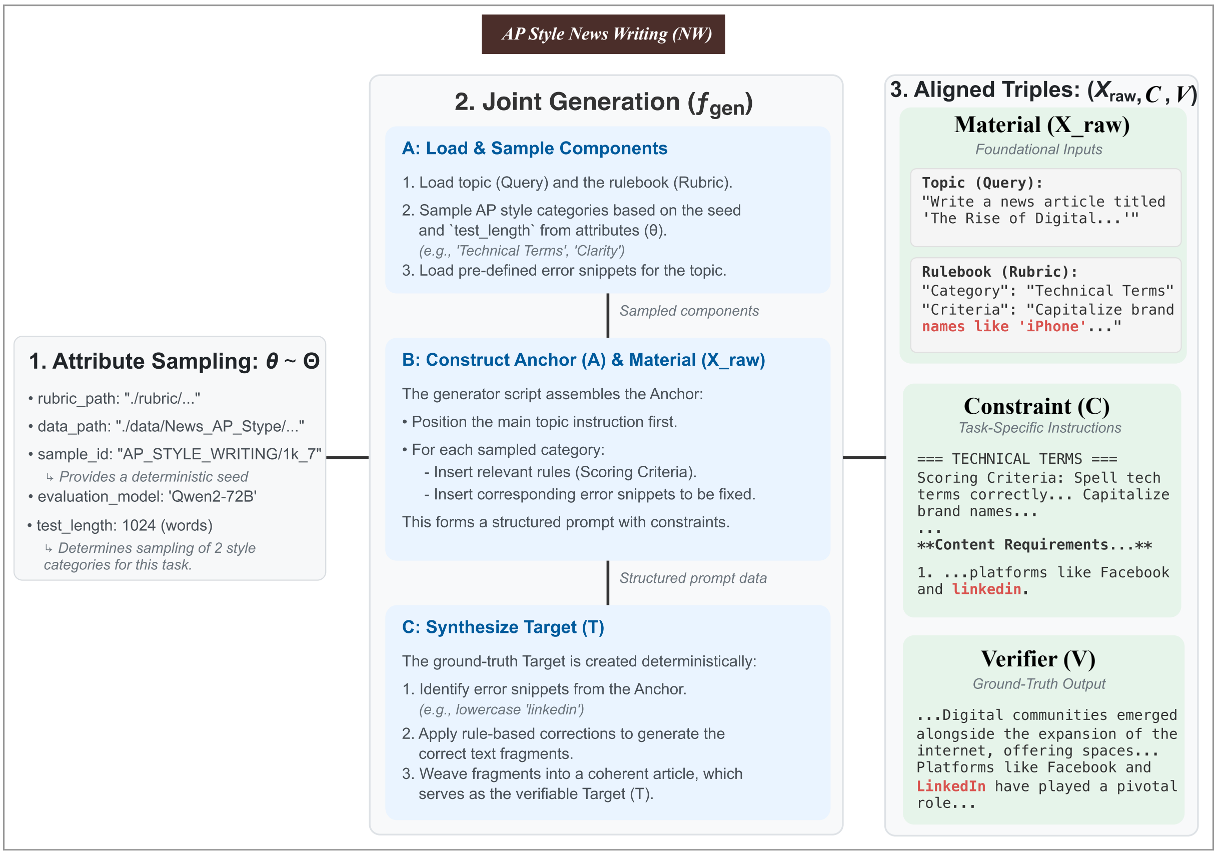
Item 2:
Question: How many sales representatives met or exceeded their sales targets?
Target Answer: 10 out of 20 reps with targets met or exceeded their goal (10 exceeded).

Item 3:
Question: What was the average deal size across all transactions?
Target Answer: The average deal size across 200 transactions was USD 14,993.
...
--- END PAIRS ---
SALES REPORT:
--- START REPORT ---
In April 2025, the East Region generated $1,526,782.59 in revenue from 200 transactions, with an average deal size of $7,633.91. Performance varied greatly among staff: Kimberly Odonnell MD (EMP016) led with $195,079.78 in sales, while Jeffrey Wagner (EMP018) was the lowest performer with $59,904.92. Only 6 of 20 reps met their targets. The top product by revenue was the 'Network Switch Pro' ($143,438.37), with hardware and software being the dominant sales categories...
--- END REPORT ---

Output Format:
Respond ONLY with a single JSON list containing exactly 5 objects. Each object must correspond in order to the 'Item' number above and have two keys: "answered" (boolean) and "correct" (boolean).

Example Output (for 2 items):
[
{"answered": true, "correct": true},
{"answered": false, "correct": false}
]

Figure 18: An illustrative example for the CSV Sales Report Analysis (SR) task (Part 2/2).



AP Style News Writing (NW)

1. Attribute Sampling: $\theta \sim \Theta$

- rubric_path: "./rubric/..."
- data_path: "./data/News_AP_Stype/..."
- sample_id: "AP_STYLE_WRITING/1k_7"
  ↳ *Provides a deterministic seed*
- evaluation_model: 'Qwen2-72B'
- test_length: 1024 (words)
  ↳ *Determines sampling of 2 style categories for this task.*

2. Joint Generation (f_{gen})

A: Load & Sample Components

1. Load topic (Query) and the rubric book (Rubric).
2. Sample AP style categories based on the seed and `test_length` from attributes (θ).
   (e.g., 'Technical Terms', 'Clarity')
3. Load pre-defined error snippets for the topic.

Sampled components

B: Construct Anchor (A) & Material (X_raw)

The generator script assembles the Anchor:

- Position the main topic instruction first.
- For each sampled category:
  - Insert relevant rules (Scoring Criteria).
  - Insert corresponding error snippets to be fixed.

This forms a structured prompt with constraints.

Structured prompt data

C: Synthesize Target (T)

The ground-truth Target is created deterministically:

1. Identify error snippets from the Anchor.
   (e.g., lowercase 'linkedin')
2. Apply rule-based corrections to generate the correct text fragments.
3. Weave fragments into a coherent article, which serves as the verifiable Target (T).

3. Aligned Triples: (X_{raw}, C, V)

Material (X_raw)

Foundational Inputs

```
Topic (Query):
"Write a news article titled
'The Rise of Digital...'"
```

```
Rulebook (Rubric):
"Category": "Technical Terms"
"Criteria": "Capitalize brand
names like 'iPhone'..."
```

Constraint (C)

Task-Specific Instructions

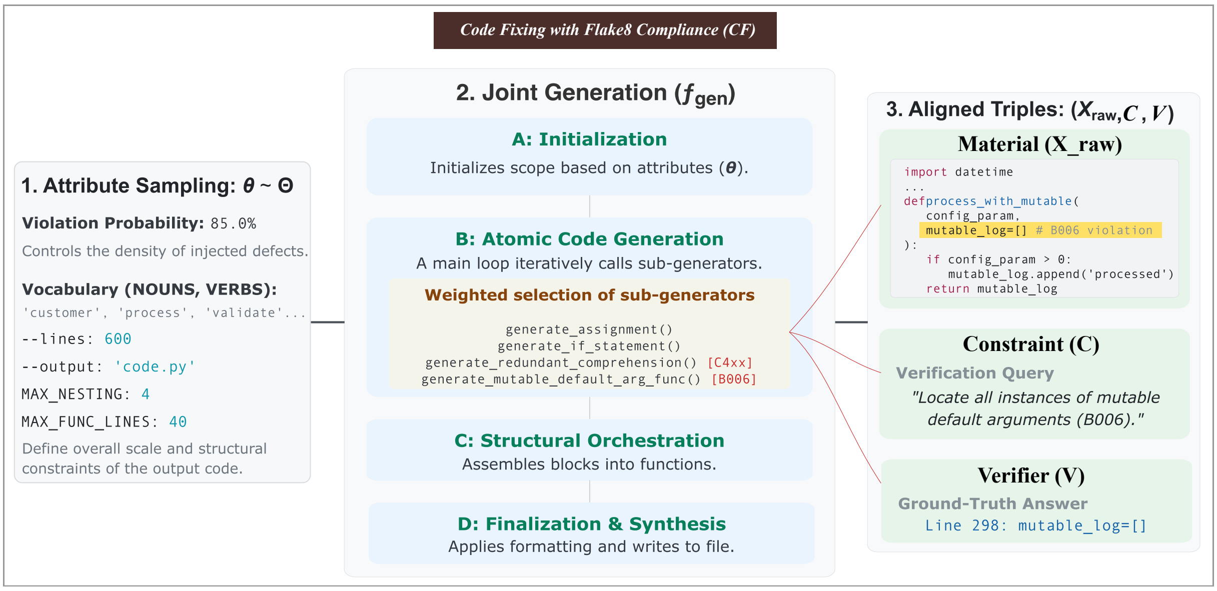
```
=== TECHNICAL TERMS ===
Scoring Criteria: Spell tech
terms correctly... Capitalize
brand names...
...
**Content Requirements...**
1. ...platforms like Facebook
and linkedin.
```

Verifier (V)

Ground-Truth Output

```
...Digital communities emerged
alongside the expansion of the
internet, offering spaces...
Platforms like Facebook and
LinkedIn have played a pivotal
role...
```

Figure 19: Overview of the data generation pipeline for the AP Style News Writing (NW) task.



Code Fixing with Flake8 Compliance (CF)

1. Attribute Sampling: $\theta \sim \Theta$

Violation Probability: `85.0%`

Controls the density of injected defects.

Vocabulary (NOUNS, VERBS):

```
'customer', 'process', 'validate'...
--lines: 600
--output: 'code.py'
MAX_NESTING: 4
MAX_FUNC_LINES: 40
```

Define overall scale and structural constraints of the output code.

2. Joint Generation (f_{gen})

A: Initialization

Initializes scope based on attributes (**θ**).

B: Atomic Code Generation

A main loop iteratively calls sub-generators.

Weighted selection of sub-generators

```
generate_assignment()
generate_if_statement()
generate_redundant_comprehension() [C4xx]
generate_mutable_default_arg_func() [B006]
```

C: Structural Orchestration

Assembles blocks into functions.

D: Finalization & Synthesis

Applies formatting and writes to file.

3. Aligned Triples: (X_{raw}, C, V)

Material (X_raw)

```
import datetime
...
defprocess_with_mutable(
    config_param,
    mutable_log=[] # B006 violation
):
    if config_param > 0:
        mutable_log.append('processed')
    return mutable_log
```

Constraint (C)

Verification Query

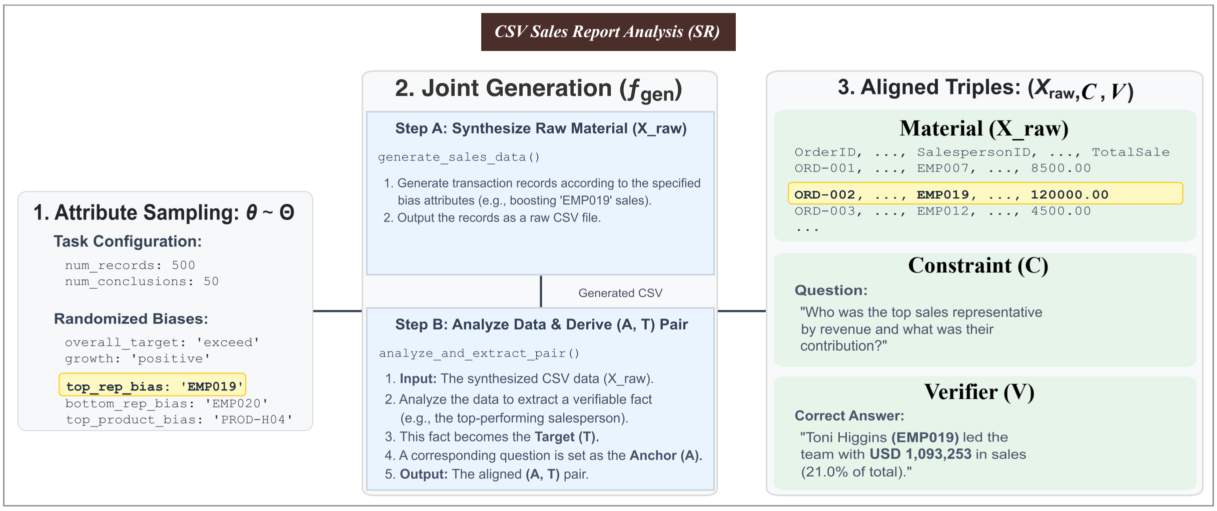
"Locate all instances of mutable default arguments (B006)."

Verifier (V)

Ground-Truth Answer

```
Line 298: mutable_log=[]
```

Figure 20: Overview of the data generation pipeline for the Code Fixing (CF) task.


CSV Sales Report Analysis (SR)
1. Attribute Sampling: θ ~ Θ
Task Configuration:
Randomized Biases:
2. Joint Generation (f_{gen})
Step A: Synthesize Raw Material (X_raw)
Generated CSV
Step B: Analyze Data & Derive (A, T) Pair
3. Aligned Triples: (X_{raw}, C, V)
Material (X_raw)
Constraint (C)
Question:
Verifier (V)
Correct Answer:

```
num_records: 500
num_conclusions: 50
```

```
overall_target: 'exceed'
growth: 'positive'
top_rep_bias: 'EMP019'
bottom_rep_bias: 'EMP020'
top_product_bias: 'PROD-H04'
```

`generate_sales_data()`

1. Generate transaction records according to the specified bias attributes (e.g., boosting 'EMP019' sales).
2. Output the records as a raw CSV file.

`analyze_and_extract_pair()`

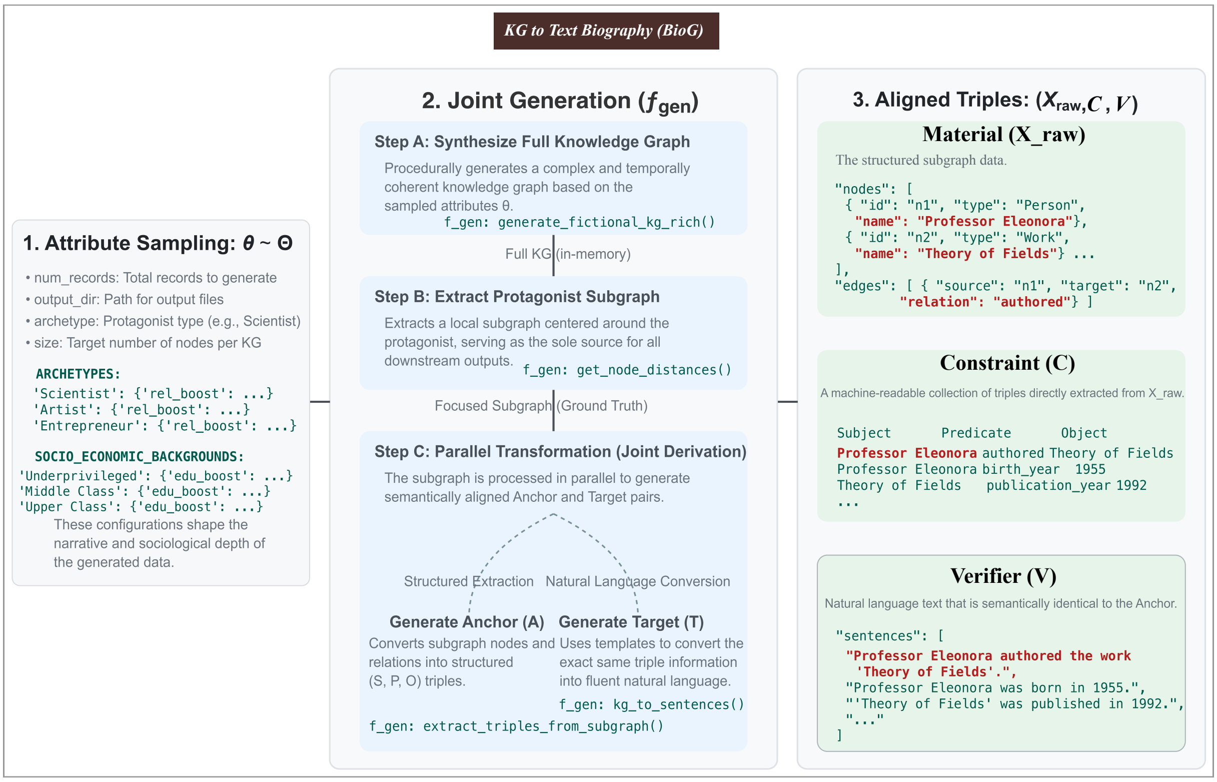
1. **Input:** The synthesized CSV data (X_raw).
2. Analyze the data to extract a verifiable fact (e.g., the top-performing salesperson).
3. This fact becomes the **Target (T)**.
4. A corresponding question is set as the **Anchor (A)**.
5. **Output:** The aligned **(A, T)** pair.

```
OrderID, ..., SalespersonID, ..., TotalSale
ORD-001, ..., EMP007, ..., 8500.00
ORD-002, ..., EMP019, ..., 120000.00
ORD-003, ..., EMP012, ..., 4500.00
...
```

"Who was the top sales representative by revenue and what was their contribution?"

"Toni Higgins **(EMP019)** led the team with **USD 1,093,253** in sales (21.0% of total)."

Figure 21: Overview of the data generation pipeline for the CSV Sales Report Analysis (SR) task.


KG to Text Biography (BioG)
1. Attribute Sampling: θ ~ Θ
2. Joint Generation (f_{gen})
Step A: Synthesize Full Knowledge Graph
Full KG (in-memory)
Step B: Extract Protagonist Subgraph
Focused Subgraph (Ground Truth)
Step C: Parallel Transformation (Joint Derivation)
Structured Extraction
Natural Language Conversion
Generate Anchor (A)
Generate Target (T)
3. Aligned Triples: (X_{raw}, C, V)
Material (X_raw)
Constraint (C)
Verifier (V)

- num_records: Total records to generate
- output_dir: Path for output files
- archetype: Protagonist type (e.g., Scientist)
- size: Target number of nodes per KG

```
ARCHETYPES:
'Scientist': {'rel_boost': ...}
'Artist': {'rel_boost': ...}
'Entrepreneur': {'rel_boost': ...}

SOCIO_ECONOMIC_BACKGROUNDS:
'Underprivileged': {'edu_boost': ...}
'Middle Class': {'edu_boost': ...}
'Upper Class': {'edu_boost': ...}
```

These configurations shape the narrative and sociological depth of the generated data.

Procedurally generates a complex and temporally coherent knowledge graph based on the sampled attributes θ.
`f_gen: generate_fictional_kg_rich()`

Extracts a local subgraph centered around the protagonist, serving as the sole source for all downstream outputs. `f_gen: get_node_distances()`

The subgraph is processed in parallel to generate semantically aligned Anchor and Target pairs.

Converts subgraph nodes and relations into structured (S, P, O) triples.

Uses templates to convert the exact same triple information into fluent natural language.
`f_gen: kg_to_sentences()`

`f_gen: extract_triples_from_subgraph()`

The structured subgraph data.

```
"nodes": [
 { "id": "n1", "type": "Person",
  "name": "Professor Eleonora"},
 { "id": "n2", "type": "Work",
  "name": "Theory of Fields"} ...
],
"edges": [ { "source": "n1", "target": "n2",
        "relation": "authored"} ]
```

A machine-readable collection of triples directly extracted from X_raw.

```
Subject            Predicate        Object
Professor Eleonora authored Theory of Fields
Professor Eleonora birth_year  1955
Theory of Fields   publication_year 1992
...
```

Natural language text that is semantically identical to the Anchor.

```
"sentences": [
 "Professor Eleonora authored the work
  'Theory of Fields'.",
 "Professor Eleonora was born in 1955.",
 "'Theory of Fields' was published in 1992.",
 "..."
]
```

Figure 22: Overview of the data generation pipeline for the KG to Text Biography (BioG) task.

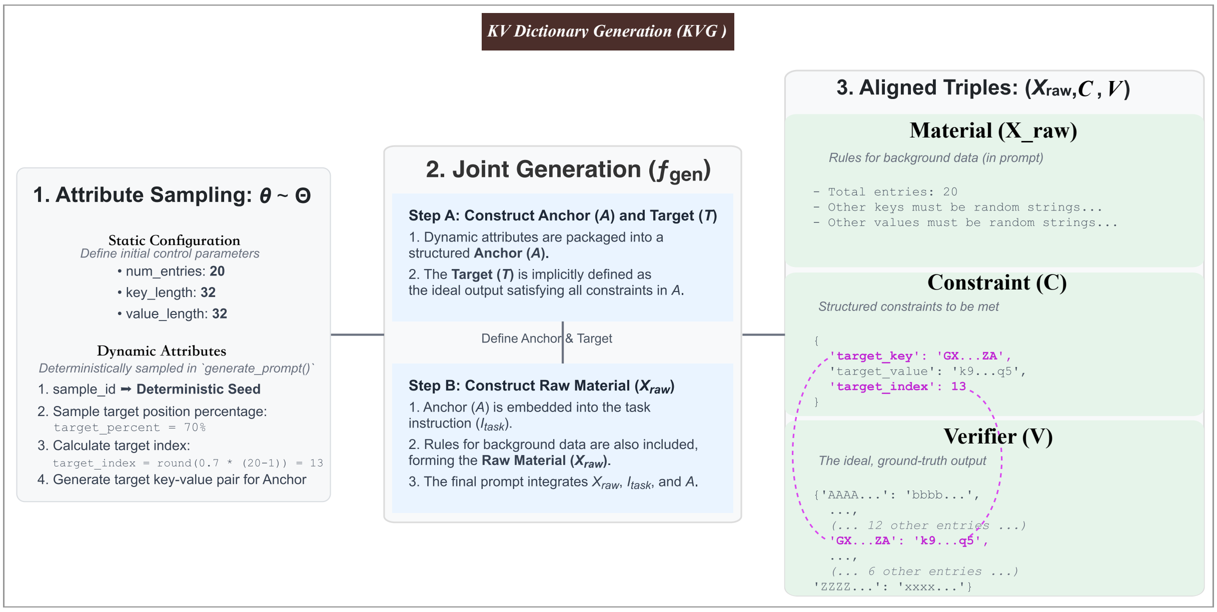

KV Dictionary Generation (KVG)
1. Attribute Sampling: $\theta \sim \Theta$
Static Configuration
Define initial control parameters
• num_entries: 20
• key_length: 32
• value_length: 32
Dynamic Attributes
Deterministically sampled in `generate_prompt()`
1. sample_id ➡ Deterministic Seed
2. Sample target position percentage:
target_percent = 70%
3. Calculate target index:
target_index = round(0.7 * (20-1)) = 13
4. Generate target key-value pair for Anchor
2. Joint Generation (f_{gen})
Step A: Construct Anchor (A) and Target (T)
1. Dynamic attributes are packaged into a structured Anchor (A).
2. The Target (T) is implicitly defined as the ideal output satisfying all constraints in A.
Define Anchor & Target
Step B: Construct Raw Material (X_{raw})
1. Anchor (A) is embedded into the task instruction (I_{task}).
2. Rules for background data are also included, forming the Raw Material (X_{raw}).
3. The final prompt integrates X_{raw}, I_{task}, and A.
3. Aligned Triples: (X_{raw}, C, V)
Material (X_raw)
Rules for background data (in prompt)
- Total entries: 20
- Other keys must be random strings...
- Other values must be random strings...
Constraint (C)
Structured constraints to be met
{
'target_key': 'GX...ZA',
'target_value': 'k9...q5',
'target_index': 13
}
Verifier (V)
The ideal, ground-truth output
{'AAAA...': 'bbbb...',
...,
(... 12 other entries ...)
'GX...ZA': 'k9...q5',
...,
(... 6 other entries ...)
'ZZZZ...': 'xxxx...'}

Figure 23: Overview of the data generation pipeline for the KV Dictionary Generation (KVG) task.

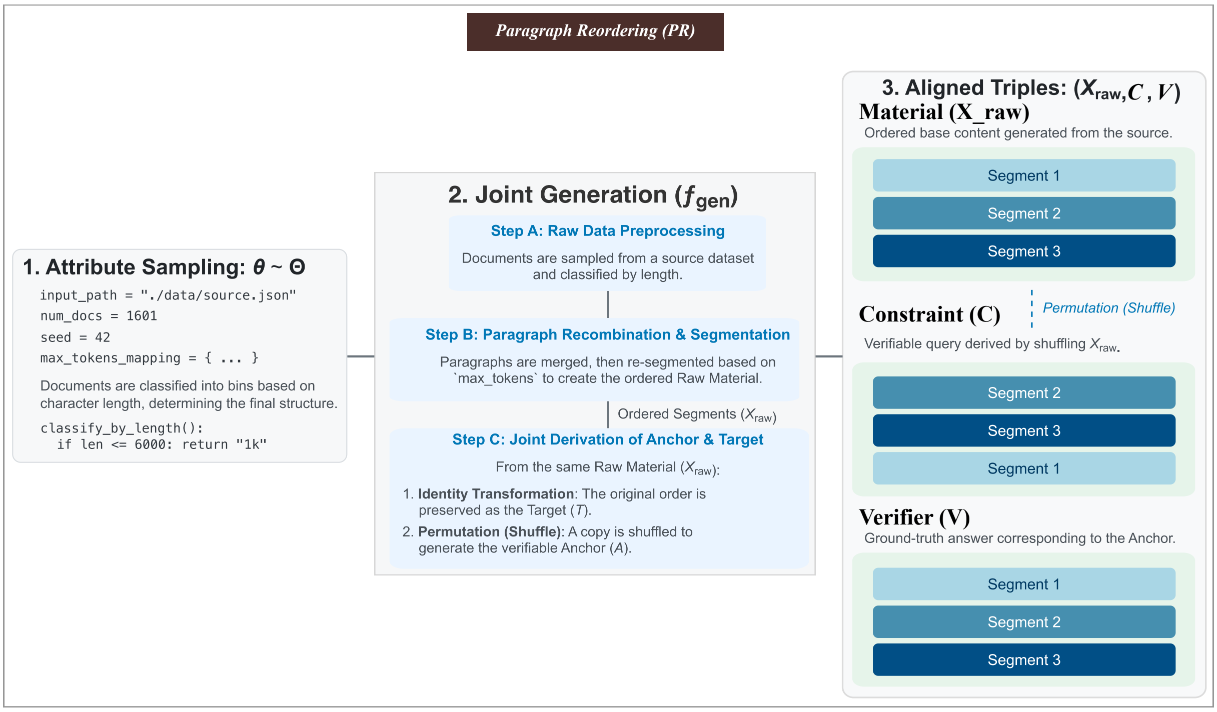
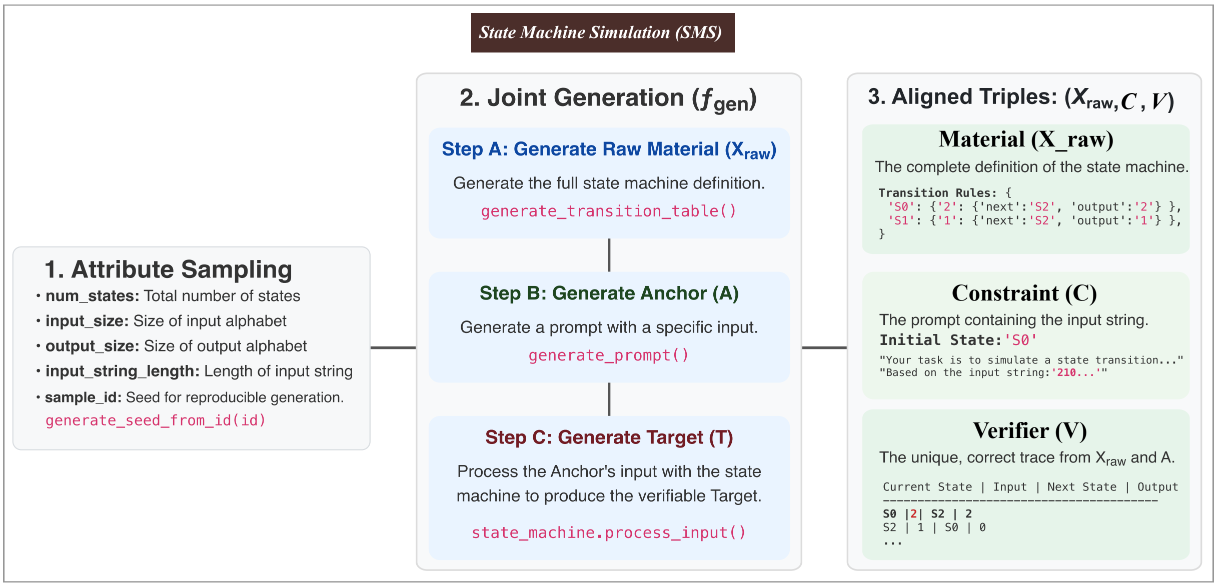

Paragraph Reordering (PR)
1. Attribute Sampling: $\theta \sim \Theta$
input_path = "./data/source.json"
num_docs = 1601
seed = 42
max_tokens_mapping = { ... }
Documents are classified into bins based on character length, determining the final structure.
classify_by_length():
  if len <= 6000: return "1k"
2. Joint Generation (f_{gen})
Step A: Raw Data Preprocessing
Documents are sampled from a source dataset and classified by length.
Step B: Paragraph Recombination & Segmentation
Paragraphs are merged, then re-segmented based on `max_tokens` to create the ordered Raw Material.
Ordered Segments (X_{raw})
Step C: Joint Derivation of Anchor & Target
From the same Raw Material (X_{raw}):
1. Identity Transformation: The original order is preserved as the Target (T).
2. Permutation (Shuffle): A copy is shuffled to generate the verifiable Anchor (A).
3. Aligned Triples: (X_{raw}, C, V)
Material (X_raw)
Ordered base content generated from the source.
Segment 1
Segment 2
Segment 3
Constraint (C)
Permutation (Shuffle)
Verifiable query derived by shuffling X_{raw}.
Segment 2
Segment 3
Segment 1
Verifier (V)
Ground-truth answer corresponding to the Anchor.
Segment 1
Segment 2
Segment 3

Figure 24: Overview of the data generation pipeline for the Paragraph Reordering (PR) task.


State Machine Simulation (SMS)
1. Attribute Sampling
num_states: Total number of states
input_size: Size of input alphabet
output_size: Size of output alphabet
input_string_length: Length of input string
sample_id: Seed for reproducible generation.
generate_seed_from_id(id)
2. Joint Generation (f_{gen})
Step A: Generate Raw Material (X_{raw})
Generate the full state machine definition.
generate_transition_table()
Step B: Generate Anchor (A)
Generate a prompt with a specific input.
generate_prompt()
Step C: Generate Target (T)
Process the Anchor's input with the state machine to produce the verifiable Target.
state_machine.process_input()
3. Aligned Triples: (X_{raw}, C, V)
Material (X_raw)
The complete definition of the state machine.
Constraint (C)
The prompt containing the input string.
Verifier (V)
The unique, correct trace from X_{raw} and A.

```
Transition Rules: {
  'S0': {'2': {'next':'S2', 'output':'2'} },
  'S1': {'1': {'next':'S2', 'output':'1'} },
}
```

```
Initial State:'S0'
"Your task is to simulate a state transition..."
"Based on the input string:'210...'"
```

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

Figure 25: Overview of the data generation pipeline for the State Machine Simulation (SMS) task.