LoopWeaver: Weaving Feedback Loops into Hierarchical Generation under Constraints

Yifan Zhu[1] **Guanting Chen**[1] **Bing Wei**[3] **Haoran Luo**[2] *

[1]Beijing University of Posts and Telecommunications
[2]Nanyang Technological University, Singapore
[3]Hainan University

{yifan_zhu,cgt}@bupt.edu.cn, haoran.luo@ntu.edu.sg

Abstract

Large Language Models face a multi-objective optimization problem in constrained long-text generation, balancing global structure, local coherence, and constraint satisfaction. We propose LoopWeaver, a hierarchical workflow with two sequential offline preference-optimization stages. Constraint-aware planning produces structured plans, feasibility filtering removes invalid plans, and a shared deterministic reward constructs preference pairs for plan-level and generation-level DPO. At inference time, LoopWeaver remains feed-forward: it generates and revises a complete plan, filters the revised plan, and then generates segments conditioned on the current sub-plan and running narrative. We evaluate the method on four LongGenBench-style structured scenarios across multiple backbones. Results show improvements in both constraint accuracy and text quality across all four open-source backbones. Our software and data are publicly available at `https://github.com/cgtaa-cmyk/LoopWeaver`.

1 Introduction

Large language models [72] excel at short-form text generation [39, 6, 1], but struggle with *constrained long-form generation* [3, 59, 58, 65, 22, 27, 56, 61, 38], which requires outputs of thousands or even ten thousand tokens that meet diverse user-specified constraints (e.g., length, structure, style, or factual requirements) while preserving global coherence [76, 7, 8]. This entails jointly optimizing global structure, local coherence, and constraint satisfaction, three tightly coupled objectives that are hard to balance [10, 26, 44]. As illustrated in Figure 1, the task unfolds as a multi-stage process interleaving planning, generation, and constraint verification [51, 16, 55, 57].

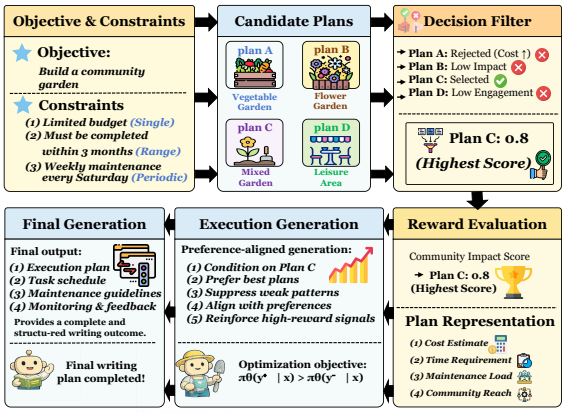


Figure 1: Constrained long-form generation in LoopWeaver with planning, generation, and verification.

A promising direction is to cast constrained long-form writing as a workflow, where planning, generation, verification, and revision are first-class

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

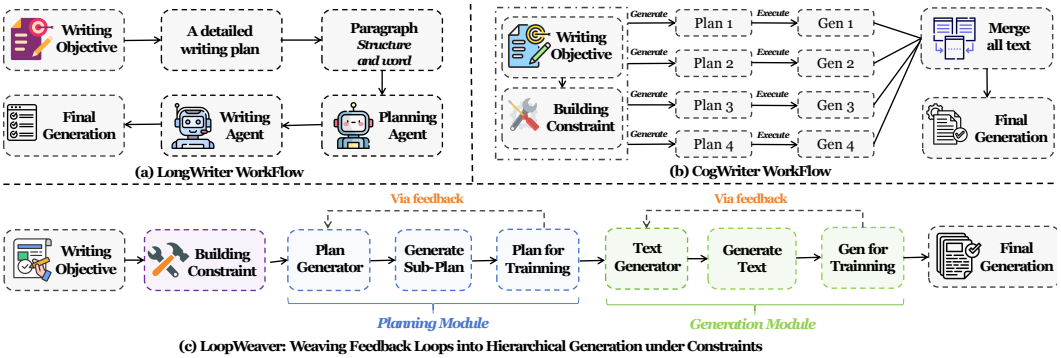


Figure 2: Framework comparison of LoopWeaver, LongWriter, and CogWriter. LoopWeaver couples plan-level and generation-level preference optimization during offline training, while inference proceeds through complete-plan revision, filtering, and segment generation.

stages connected by structured information flow [51, 55, 12, 34]. This view aligns with the rise of agentic LLM systems, coding agents such as Claude Code, and frameworks such as OpenHands, showing how workflow-style decomposition extends LLM capabilities on complex, multi-step tasks [25, 29, 66, 28, 9, 32]. Early efforts follow this trend by decomposing long-form writing into hierarchical or staged pipelines that interleave outlining, drafting, and refinement [47, 59, 65, 27]. Together, these developments suggest that explicit workflow-level modeling rather than monolithic decoding or rigid pipelines alone is a natural fit for the coupled optimization such tasks require.

Existing approaches largely fall into two paradigms, neither fully resolving the coupled optimization above. **Data-centric methods** use data augmentation or preference optimization to produce longer, more constraint-aware outputs [3, 17, 74, 40, 54, 36, 42, 11, 48], but encode constraints implicitly and cannot verify or repair violations at inference, letting errors accumulate [30, 4, 37]. **Planning-based methods** decompose generation into hierarchical outlines or multi-agent pipelines [59, 63, 45, 65, 47, 41, 43, 14], yet planning, generation, and verification stay loosely coupled: planning-time constraints rarely reach segment-level decoding, and later errors seldom revise earlier plans [69, 31, 35, 64]. Neither fully and jointly optimizes the three objectives, motivating a unified offline preference-optimization formulation.

However, when transplanted to constrained long-form writing, existing workflow-style systems still expose three recurring limitations: **(i) No constraint-aware modeling.** Generic workflows orchestrate sub-tasks without propagating user constraints through outlining, drafting, and assembly [5, 70, 18, 19]. **(ii) Limited adaptivity.** Most pipelines use static flows or costly post-hoc refinement, still unable to fully adapt to changing constraints, length, or intermediate signals [77, 67, 75, 50, 2]. **(iii) Loose feedback coupling.** Feedback in Claude Code and OpenHands is local and post hoc, with no closed loop letting segment-level signals revise the global plan or vice versa [15, 69, 23, 52, 13, 60].

To address these limitations, we propose **LoopWeaver**, a hierarchical workflow whose two preference-optimization stages are coupled during offline training [68, 24, 71]. *Constraint-aware hierarchical planning* decomposes user goals into a complete structured plan with constraints propagated to its generation units [27, 65, 20]. *Feasibility filtering* removes plans that fail prompted checks for relevance, completeness, coherence, specificity, efficiency, and consistency [47, 73, 53]. Finally, *reward-based preference optimization* uses a shared deterministic reward to construct plan-level and generation-level preference pairs [49, 40, 36]. The resulting policies are executed feed-forward at inference time; generated segments are not returned to the planner.

We evaluate LoopWeaver on four LongGenBench-style structured constrained-generation scenarios across multiple backbone models, including the Qwen2.5 [62] series and LLaMA3.1-8B-Instruct [46], and compare it with both proprietary and open-source baselines such as GPT-4o-mini [1], as well as representative workflow-based and data-centric methods including CogWriter [47] and LongWriter [3, 54]. Across metrics covering constraint satisfaction and narrative-level quality [57, 38, 8], the revised audit reports where LoopWeaver improves over baselines and where it does not, rather than assuming a uniform gain across backbones; these results do not constitute evidence of unrestricted cross-domain generalization. These findings highlight the benefits of coordinated planning and generation.

2 Related Work

2.1 Constrained Long-Form Text Generation.

Constrained long-form text generation aims to produce coherent long texts under task-specific requirements such as content, style, length, and structural constraints [44, 8, 61, 33]. Prior work mainly falls into two paradigms [26]: controllable generation via explicit constraint modeling, which uses instruction tuning, decoding-time control, or RL-based prompt optimization to enforce user-specified properties [31, 76, 11, 64, 35], and planning-based generation, where hierarchical outlines or retrieval-augmented contexts improve global coherence and structural consistency [47, 43, 65]. A complementary line scales long-form generation through large-context training and preference optimization [3, 54, 36]. Despite this progress, existing methods typically adopt loosely coupled pipelines [37, 14, 21], limiting the integration of constraint signals and feedback during generation, which often leads to compounding errors and degraded coherence in long outputs [30, 57, 38].

2.2 Agentic Workflows

Recent work abstracts agentic workflows as executable computation graphs, dividing existing methods into static and dynamic approaches [12, 67, 34]. Static methods commit to reusable templates offline via MCTS-based operator search [70], code-space refinement [18], abstract operator libraries [71], prompt compilation or reflective evolution within fixed scaffolds [19, 2], joint structure-configuration updates [75, 50], and validation-embedded search [73, 53]; related lines explore tool-orchestrated reasoning [63, 45] and chain-of-agents pipelines [25, 29]. Dynamic methods defer structural decisions to inference time, either pruning redundant edges from fixed hypergraphs [23, 52] or synthesizing per-query workflows via autoregressive DAG generation [24], agentic supernets [68], RL meta-agents [13], or preference learning [60, 77]. While these advances improve task generality and adaptivity, they primarily target reasoning, tool use, or code generation, rarely incorporating explicit constraint propagation or feedback-coupled coordination for constrained long-form text generation [66, 28, 5].

3 Preliminaries

This section introduces the reward-scoring and preference-optimization components used in LoopWeaver's two sequential offline DPO stages.

(i) Deterministic Unit-Level Reward Evaluation. We score each candidate sub-plan or generated segment under the shared rule-based reward. No additional rollouts occur for reward estimation.

$$r(z, e) = R(z, e), \tag{1}$$

where z is the candidate unit and $e = (x, C)$ is its prompt-and-constraint context.

(ii) Reward Function $R(\cdot)$. $R(\cdot)$ is a *deterministic, rule-based* scoring function. Given a candidate unit s and context $e = (x, C)$ with $C = C_{\text{once}} \cup C_{\text{range}} \cup C_{\text{periodic}}$, it returns a scalar in $[0, 1]$:

$$R(s, e) = \min\big(1,\ r_{\text{constraint}}(s, C) + \alpha \cdot r_{\text{relevance}}(s, x)\big), \tag{2}$$

$$r_{\text{constraint}}(s, C) = \frac{1}{|C|} \sum_{c \in C} \mathbf{1}[\rho(s, c) \geq \tau]\,, \tag{3}$$

where x is the input prompt and $e = (x, C)$ is the prompt-and-constraint context. The relevance term $r_{\text{relevance}}(s, x)$ measures prompt-keyword coverage with $\alpha = 0.5$.

$$\rho(s, c) = \begin{cases} 1, & c \subseteq s, \\ \text{SequenceMatcher.ratio}(s, c), & \text{otherwise}, \end{cases} \qquad \tau = 0.7. \tag{4}$$

Thus, the reward measures constraint coverage and prompt relevance only; it is not a direct evaluator of tone, factuality, or document-level coherence. Candidate units are ranked by $R(s, e)$ to construct DPO preference pairs.

(iii) Direct Preference Optimization. DPO aligns preferences by optimizing the likelihood of higher- and lower-preference outputs z^+ and z^- based on the deterministic reward, improving alignment.

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(z^+, z^-) \sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(z^+ \mid x, C)\, \pi_{\text{ref}}(z^- \mid x, C)}{\pi_\theta(z^- \mid x, C)\, \pi_{\text{ref}}(z^+ \mid x, C)}\right)\right]. \tag{5}$$

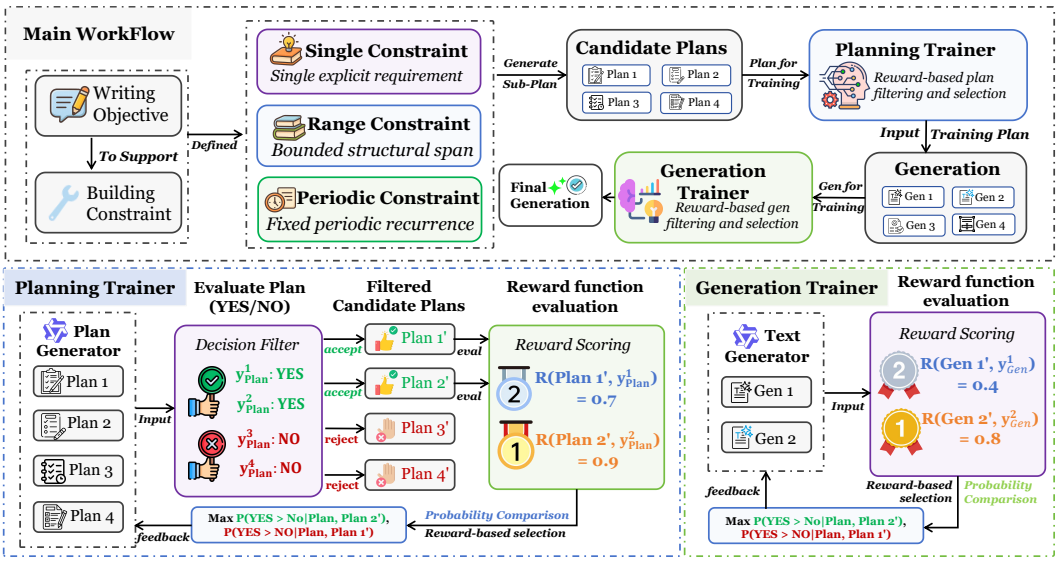


Figure 3: Overview of LoopWeaver. Complete-plan construction, in-context revision, and whole-plan filtering operate at inference time; shared-reward plan-level and generation-level DPO provide sequential coupling during offline training.

Here, x is the input prompt and C is the structured constraint set;; $(z^+, z^-) \sim D$ are preference pairs with z^+ ranked higher than z^- by deterministic reward $r(\cdot)$; $\pi_\theta, \pi_{\text{ref}}$ are the policy and reference models; β is the temperature controlling deviation from π_{ref}; σ is the sigmoid function, applied to compute preference probabilities in the objective.

4 Methodology: LoopWeaver

As illustrated in Figure 3, we present **LoopWeaver**, a hierarchical workflow with three components: *constraint-aware hierarchical planning*, *feasibility filtering*, and *reward-based preference optimization*. The first two operate in the feed-forward inference workflow, while the shared reward couples the plan-level and generation-level DPO stages during offline training.

4.1 Constraint-Aware Hierarchical Planning

LoopWeaver adopts a two-level architecture separating a *plan level* that captures global structure from a *generation level* that realizes each unit into text. Given an input prompt x and a constraint set $C = C_{\text{once}} \cup C_{\text{range}} \cup C_{\text{periodic}}$ partitioned by pattern following LongGenBench [57], the framework produces an ordered sub-plan sequence and an aligned segment sequence, written as a two-level form:

$$\underbrace{p = (S_1,\, S_2,\, \ldots,\, S_m)}_{\text{plan level}} \longrightarrow \underbrace{G = (G_1,\, G_2,\, \ldots,\, G_m)}_{\text{generation level}}, \tag{6}$$

where each sub-plan S_j expands into the corresponding segment G_j, and the integer m denotes the number of units determined by the planner. The complete plan is generated from the prompt and structured constraint set, and the two levels are coupled by the shared reward used in the sequential offline DPO stages described in Section 4.3:

$$p \;\sim\; \pi_\theta\big(\;\cdot\;\big|\,x,\, C\big). \tag{7}$$

The planner then checks and revises the sub-plans within the joint context of the complete draft and the original requirements. The revision returns a complete plan, so an upstream edit and any affected downstream units can be checked and adjusted together:

$$p' \;=\; \mathcal{R}\big(p;\, x, C\big). \tag{8}$$

The repair operator targets individual sub-plans but receives and returns the complete plan. The revised plan p' is subsequently passed to the whole-plan feasibility filter, which checks completeness,

logical coherence, and contextual consistency before segment generation. This provides plan-level coordination before generation, but it is not a formal guarantee of final document-level coherence: at inference, segments are generated sequentially in separate calls conditioned on the current sub-plan and the running narrative, and the generated segments are not returned to the planner for plan revision.

Empirical Observation 1. *Constraint-aware hierarchical planning is associated with improved plan validity and constraint coverage under the evaluation protocol in Section 5.*

4.2 Feasibility Filtering

Revised plans still vary in plan-level quality, so a lightweight feasibility filter discards candidates that fail task-inherent criteria before they enter the generation stage. From the candidate pool $P_{\text{final}} = \{p_1, p_2, \ldots, p_N\}$ produced by Section 4.1, the same backbone receives each complete plan together with the prompt and constraint set and returns a prompted YES/NO feasibility judgment:

$$\delta_i \;=\; \mathbf{1}[p_{\text{filter}}(p_i, x, C) = \text{YES}]\,, \qquad i = 1,\, 2, \ldots, N, \tag{9}$$

where the prompt asks the model to assess six criteria: *relevance, completeness, coherence, efficiency, specificity*, and *consistency*. The filter produces one binary decision; it does not use a separately trained filter or multiply six probabilities. The accepted plans are gathered into the filtered set:

$$P_{\text{filtered}} \;=\; \big\{\, p_i \in P_{\text{final}} \;\big|\; \delta_i = 1 \,\big\}, \tag{10}$$

which is forwarded to the generation-level optimization in Section 4.3, while rejected candidates are either discarded or regenerated under the same prompt x and constraint set C, ensuring that deterministic reward scoring is applied only to plans that pass a coarse feasibility check.

Empirical Observation 2. *Feasibility filtering reduces the number of malformed or misaligned plans passed to generation under the evaluation protocol in Section 5.*

4.3 Reward-Based Preference Optimization

LoopWeaver couples the two levels through a single shared reward $R(\cdot, e)$ defined in Section 3, evaluated under the context $e = (x, C)$ and used to score both sub-plans and segments. At the *plan level*, each candidate sub-plan S_j from a filtered plan in P_{filtered} is scored under this reward:

$$r(S_j) \;=\; R\big(S_j,\, e\big), \tag{11}$$

For two candidate sub-plans S_j^+ and S_j^- drawn from the same context, the higher-reward candidate is preferred when $r(S_j^+) > r(S_j^-)$; the resulting preference pairs are used to train the plan policy through the plan-level DPO objective:

$$\mathcal{L}_{\text{DPO}}^{\text{plan}}(\theta) \;=\; -\,\mathbb{E}_{(S_j^+,\, S_j^-)\sim D_{\text{plan}}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(S_j^+ \mid x, C)\,\pi_{\text{ref}}(S_j^- \mid x, C)}{\pi_\theta(S_j^- \mid x, C)\,\pi_{\text{ref}}(S_j^+ \mid x, C)}\right)\right], \tag{12}$$

in which D_{plan} collects reward-ranked plan pairs, π_{ref} is the frozen reference policy, and β is the DPO temperature controlling deviation from π_{ref}. At the *generation level*, each accepted sub-plan S_k is expanded into a segment G_k and scored by the same shared reward, $r(G_k) = R\big(G_k, e\big)$. Pairs (G_k^+, G_k^-) with $r(G_k^+) > r(G_k^-)$ supervise the generation policy through the generation-level DPO objective:

$$\mathcal{L}_{\text{DPO}}^{\text{gen}}(\theta) \;=\; -\,\mathbb{E}_{(G_k^+,\, G_k^-,\, S_k)\sim D_{\text{gen}}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(G_k^+ \mid x, C, S_k)\,\pi_{\text{ref}}(G_k^- \mid x, C, S_k)}{\pi_\theta(G_k^- \mid x, C, S_k)\,\pi_{\text{ref}}(G_k^+ \mid x, C, S_k)}\right)\right], \tag{13}$$

where D_{gen} collects segment-level preference triples conditioned on S_k. The same $R(\cdot, e)$ is used at both levels, while the two DPO stages are optimized sequentially with checkpoint transfer from the plan policy to the generation policy. This is the sense in which the stages are coupled during offline training; inference remains feed-forward.

Empirical Observation 3. *Reward-based preference optimization improves alignment with the verifiable constraint objectives under the evaluation protocol in Section 5.*

Table 1: Comparison of narrative-level text quality and constraint-following accuracy across different methods.

Method	Text Quality					Constraint Accuracy			
	Narr.	Memory	Temporal	Affect.	Avg.	Once	Range	Peri.	Avg.
Qwen2.5-0.5B-Instruct	26.05±0.26	24.31±0.23	19.74±0.57	**39.86±0.39**	27.49±0.21	23.46±1.59	16.60±1.52	15.69±0.54	18.58±0.75
+ CogWriter	23.38±0.14	24.24±0.43	19.51±0.32	33.50±0.26	25.16±0.12	26.61±0.98	18.03±0.30	**17.70±0.24**	20.78±0.21
+ LongWriter	24.87±0.30	25.26±0.30	22.41±0.40	31.26±0.39	25.95±0.13	21.01±0.30	19.29±0.11	15.83±0.20	18.71±0.13
+ LoopWeaver (ours)	**29.17±0.22**	**35.02±0.57**	**33.80±0.46**	37.39±0.20	**33.84±0.31**	**28.01±0.55**	**22.52±0.10**	15.80±0.28	**22.11±0.31**
Qwen2.5-1.5B-Instruct	41.96±0.42	40.12±0.37	34.63±0.47	**59.17±0.73**	43.97±0.45	23.30±1.43	22.22±0.28	12.12±0.60	19.22±0.77
+ CogWriter	42.32±0.31	45.36±0.64	38.21±0.13	56.00±0.37	45.47±0.32	31.66±3.09	25.76±0.91	17.79±0.22	25.07±1.41
+ LongWriter	41.32±0.47	44.24±0.28	39.01±0.53	55.13±0.42	44.93±0.15	42.67±0.36	28.17±1.16	41.24±0.49	37.36±0.44
+ LoopWeaver (ours)	**47.98±1.00**	**55.21±0.94**	**49.27±0.83**	56.83±0.38	**52.32±0.67**	**49.01±0.16**	**38.88±0.13**	**42.93±0.30**	**43.61±0.20**
Qwen2.5-7B-Instruct	69.75±0.66	70.94±0.23	67.21±0.44	77.46±0.47	71.34±0.34	40.97±0.74	32.86±3.44	18.39±2.22	30.74±2.12
+ CogWriter	73.33±0.36	76.02±0.10	72.69±0.22	80.29±0.38	75.58±0.09	42.05±0.91	56.65±0.35	27.90±0.43	42.20±0.56
+ LongWriter	68.33±0.32	73.68±0.44	67.15±0.90	79.34±0.14	72.12±0.20	27.23±0.17	32.48±1.07	27.91±0.26	29.21±0.50
+ LoopWeaver (ours)	**75.98±0.40**	**79.42±0.37**	**78.73±0.88**	**80.94±0.71**	**78.77±0.23**	**53.33±0.54**	**56.90±0.51**	**35.01±1.29**	**48.41±0.75**
LLaMA3.1-8B-Instruct	39.08±0.41	39.19±0.27	35.56±0.49	41.79±0.07	38.91±0.09	31.51±0.84	27.87±0.43	21.02±0.39	26.80±0.55
+ CogWriter	53.29±0.32	57.91±0.32	53.71±0.41	59.67±0.35	56.14±0.12	43.12±0.11	48.77±0.07	33.80±0.07	41.89±0.08
+ LongWriter	57.31±0.54	61.62±0.70	59.33±0.69	60.08±0.07	59.59±0.42	41.01±0.30	49.29±0.11	32.83±0.20	41.04±0.20
+ LoopWeaver (ours)	**62.52±0.50**	**65.00±0.23**	**63.56±0.71**	**68.50±0.39**	**64.90±0.42**	**45.94±1.26**	**53.27±1.05**	**46.44±1.28**	**48.55±1.20**
gpt-4o-mini	71.81±0.23	73.67±0.53	64.77±0.30	82.44±0.39	73.17±0.31	51.47±0.21	40.00±0.11	44.36±0.30	45.28±0.13
+ CogWriter	78.38±0.50	80.54±0.50	74.99±1.66	85.03±0.55	79.73±0.77	65.63±0.24	42.16±0.98	32.66±0.20	46.82±0.30

Table 2: Ablation study of Planning and Generation modules in LoopWeaver (Qwen2.5-7B).

Method	Text Quality					Constraint Accuracy			
	Narr.	Memory	Temporal	Affect.	Avg.	Once	Range	Peri.	Avg.
Base Model	69.75±0.66	70.94±0.23	67.21±0.44	77.46±0.47	71.34±0.35	40.97±0.74	32.86±3.44	18.39±2.22	30.74±2.12
+ Planning	74.73±0.16	77.81±0.47	76.60±0.58	79.77±0.56	77.23±0.21	48.55±0.08	53.63±0.38	30.44±0.21	44.20±0.05
+ Generation	73.67±0.24	77.19±0.60	75.85±0.30	79.56±0.43	76.57±0.31	49.59±0.05	53.19±0.19	31.55±0.35	44.78±0.12
Full LoopWeaver	**75.98±0.40**	**79.42±0.37**	**78.73±0.88**	**80.94±0.71**	**78.77±0.23**	**53.33±0.54**	**56.90±0.51**	**35.01±1.29**	**48.41±0.75**

5 Experiments

This section presents the experimental setup, main results, and in-depth analysis of our approach. To systematically evaluate LoopWeaver, we address the following research questions (RQs): **RQ1:** How much does LoopWeaver improve overall model performance over existing baseline approaches? **RQ2:** How effective are LoopWeaver's key components, as shown by comprehensive ablation studies? **RQ3:** How does constraint-aware workflow design enhance constraint satisfaction? **RQ4:** How does the offline preference-optimization workflow trade off inference efficiency against constraint-following accuracy? **RQ5:** How does offline plan-level and generation-level preference optimization coordinate the two levels? **RQ6:** How does backbone model scale affect constrained long-form generation under identical prompts and constraints?

5.1 Experimental Setup

Baseline. We compare LoopWeaver with CogWriter [47] and LongWriter [3] for the Qwen2.5 [62] and LLaMA3.1-8B backbones, and we additionally include GPT-4o-mini and its CogWriter variant as proprietary-model baselines. Details are provided in Appendix E.

Evaluation Metrics. We evaluate model performance on LongGenBench along two axes of evaluation. Instruction Following Accuracy measures compliance with single, range-based, and periodic constraints, reporting both category-wise and overall accuracy. Text Quality is assessed across narrative coherence, memory consistency, temporal grounding, and affective consistency, using an aggregated score. Additional details are provided in Appendix F for completeness and reproducibility.

Implementation Details. We implement LoopWeaver following the **LongGenBench** [57] framework, incorporating components for training and optimization. All experiments run on 3 NVIDIA A40 GPUs (48GB), enabling scalable distributed computation. Details are in Appendix G.

5.2 Reward and Feasibility Audits

We audited the fixed candidate set used for preference construction rather than relying on a sampled reward estimate. The audit covers 984 records and 63,972 candidate units; recomputing the deterministic reward with the default setting $(\tau, \alpha) = (0.7, 0.5)$ matches the saved value for all 984 records. For each sensitivity setting, selection retention compares the candidate identities selected for preference construction with those selected under the default setting. A record is eligible when its reward gap is at least 0.2; the eligible n column reports the number and percentage of eligible records, and Δn is measured relative to the default setting.

Table 3: Reward and feasibility audits. The left panel reports the fixed-candidate sensitivity audit under the deterministic reward, while the right panel reports feasibility-filter outcomes and downstream DPO-pair yield.

(a) Fixed-candidate sensitivity audit

Setting	Value	Selection retained (%)	Selection changed (%)	Eligible n (%)	Δn
τ ($\alpha = 0.5$)	0.6	97.6	2.4	924 (93.9)	+16
τ ($\alpha = 0.5$)	**0.7**	**100.0**	0.0	908 (92.3)	0
τ ($\alpha = 0.5$)	0.8	99.2	0.8	902 (91.7)	−6
α ($\tau = 0.7$)	0.3	99.8	0.2	888 (90.2)	−20
α ($\tau = 0.7$)	**0.5**	**100.0**	0.0	908 (92.3)	0
α ($\tau = 0.7$)	0.7	99.0	1.0	920 (93.5)	+12

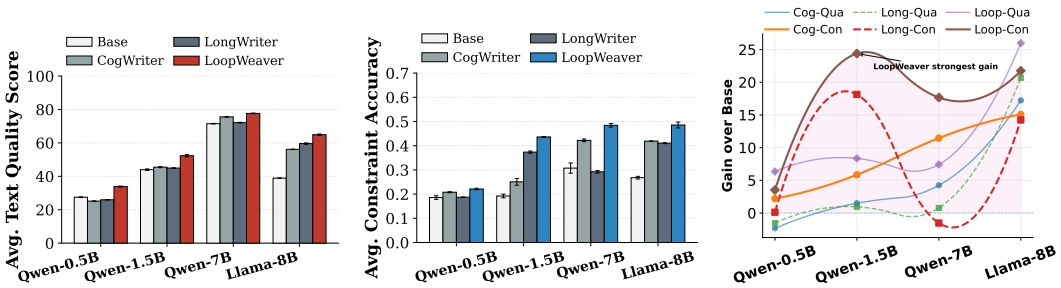
(b) Feasibility-filter audit

Audit item	Split A	Split B	Overall
Candidates	400	400	800
Accepted	251	241	492 (61.50%)
Rejected	147	158	305 (38.13%)
Failed	2	1	3 (0.38%)
Valid DPO pairs from accepted plans	220	211	431
Mean/min reward gap	0.93/0.50	0.94/0.50	0.94/0.50
Proxy precision/recall (%)	62.6/74.4	59.3/71.1	61.0/72.8
Proxy false-rejection (%)	25.6	28.9	27.2

The feasibility-filter audit covers two 400-candidate splits. The filter uses the same Llama-3.1-8B-Instruct base checkpoint used to initialize the corresponding planning and generation policies; under the default inference wrapper, it makes one prompted YES/NO decision per complete plan with temperature 0.7, top-p 0.9, and max tokens 1200. Its structural proxy checks requested unit counts and complete Week/Floor/Block/Unit identifiers; it is not a semantic six-criterion accuracy measure. Accepted and rejected counts refer to the filter decision. The three failed cases did not produce a valid decision and are therefore excluded from the confusion-matrix metrics; over the remaining 797 cases, pooled TP/FP/FN/TN are 300/192/112/193. Proxy precision and recall are computed over these non-failed decisions, and false-rejection is defined as $1 - \text{recall}$. An accepted plan contributes a DPO pair only when an aligned same-context comparator with a strictly different reward is available. Thus, 492 accepted plans yield 431 valid pairs; the remaining 61 accepted plans lack an eligible aligned comparator and do not indicate additional filter failures.

5.3 Main Results (RQ1)

As shown in Table 1 and Figure 4, we compare LoopWeaver with the base model, CogWriter, and LongWriter across backbones on (a) text quality and (b) constraint-following accuracy.



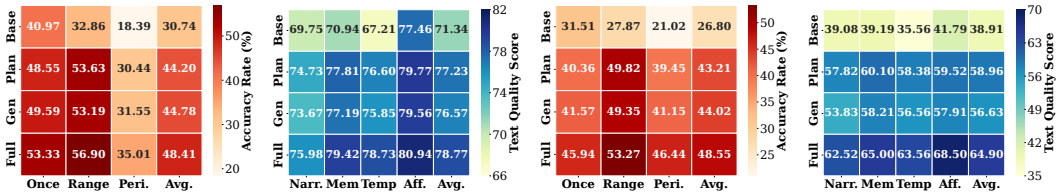
(a) Text Quality Evaluation (b) Constraint Accuracy (c) Constraint Gain Analysis

Figure 4: Comparison of LoopWeaver against baseline methods on four backbones of varying scales. **(a)** Text quality scores. **(b)** Constraint accuracy. **(c)** Gains over Base in quality and constraint accuracy across backbones.

Across all backbones, LoopWeaver achieves superior performance on both metrics, jointly optimizing generation quality and constraint satisfaction. **(a) Text Quality.** LoopWeaver achieves the highest average text-quality score among the compared methods on all four open-source backbones, reaching 33.84, 52.32, 78.77, and 64.90 for Qwen2.5-0.5B, Qwen2.5-1.5B, Qwen2.5-7B, and LLaMA3.1-8B, respectively. These results exceed CogWriter by 8.68, 6.85, 3.19, and 8.76 points, demonstrating improvements across different model scales. **(b) Constraint-Following Accuracy.** LoopWeaver also achieves the highest overall constraint accuracy within each backbone group. On Qwen2.5-7B, its average accuracy reaches 48.41, outperforming the base model and CogWriter by 17.67 and 6.21 percentage points, respectively. Together with the improvements on the other backbones, these results support the effectiveness of LoopWeaver for constrained long-form generation. **(c) Gain over Base.** Figure 4(c) shows that LoopWeaver delivers the largest gains over the corresponding base model in both average text quality and overall constraint accuracy across the four backbones. These improvements demonstrate the benefits of the proposed workflow on both evaluation axes.

5.4 Ablation Study on the Effectiveness of LoopWeaver's Key Components (RQ2)

Table 2 reports the ablation study on Qwen2.5-7B-Instruct; Figure 5 additionally reports the corresponding results for LLaMA3.1-8B.



(a) Qwen Constraint (b) Qwen Quality (c) LLaMA Constraint (d) LLaMA Quality

Figure 5: Ablation of LoopWeaver's Planning and Generation modules on Qwen2.5-7B and LLaMA3.1-8B backbones. **(a, c)** Constraint accuracy across constraint types. **(b, d)** Text quality across evaluation dimensions.

Both Optimization Stages Improve Constraint Satisfaction. As shown in Figures 5(a) and 5(c), training either the Planning or Generation module improves constraint-following accuracy over the base model on both backbones. Their relative advantages vary across constraint types, while the full workflow achieves the highest overall accuracy, supporting the benefits of combining planning and generation optimization.

Sequential Plan–Generation Optimization. The full workflow yields the strongest constraint accuracy in this ablation, while the separate Planning-only and Generation-only variants show complementary gains across the two backbones. This supports the role of sequential plan- and generation-level optimization without implying joint backpropagation.

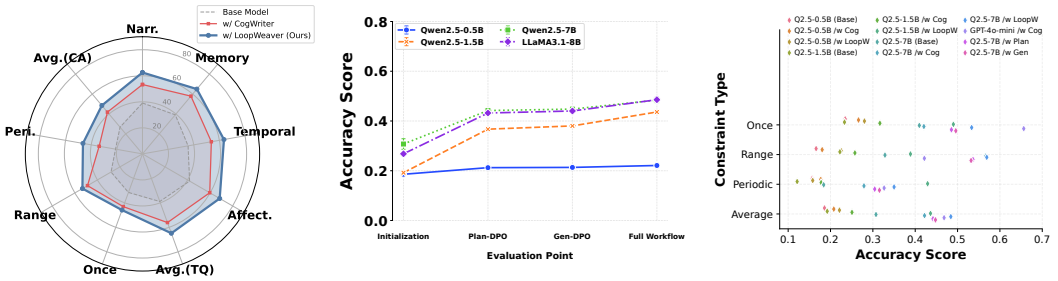
Improved Text Quality through Combined Optimization. As shown in Figures 5(b) and 5(d), both single-stage variants improve average text quality over Base, with Planning-only outperforming Generation-only on both backbones. Full LoopWeaver further improves the average score to 78.77 on Qwen2.5-7B and 64.90 on LLaMA3.1-8B, exceeding the stronger single-stage variant by 1.54 and 5.94 points, respectively.

5.5 Constraint-Aware Workflow Design (RQ3)

As shown in Figure 6, constraint-aware workflow design improves performance across multiple dimensions: overall capability, workflow-variant accuracy, and different constraint types.

Holistic Performance Across Dimensions. Figure 6(a) compares Base, CogWriter, and LoopWeaver on LLaMA3.1-8B. LoopWeaver achieves higher scores across the displayed text-quality dimensions and constraint-accuracy categories than both comparison methods.

Workflow Integration Improves Constraint Accuracy. Figure 6(b) reports four cumulative offline evaluation points: *Initialization* uses the base model; *Plan-DPO* adds plan-level preference optimization; *Gen-DPO* adds the sequential generation-level DPO stage; and *Final Workflow* addi-



(a) Performance comparison. (b) Accuracy at different stages. (c) Ablation on Constraints

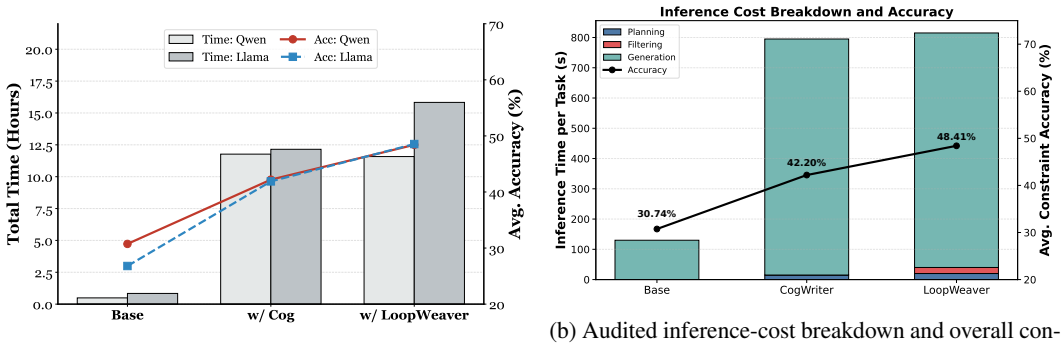
Figure 6: Ablation analysis of LoopWeaver across multiple perspectives. **(a)** Multi-dimensional performance comparison. **(b)** Cumulative constraint-accuracy effects at four offline evaluation points, namely Initialization, Plan-DPO, Gen-DPO, and Final Workflow, rather than stages of one inference trajectory. **(c)** Accuracy distribution across constraint types.

tionally enables complete-plan generation, revision, and whole-plan filtering. These points describe component additions, not an inference-time stage trajectory or iterative feedback loop.

Constraint-Type-Specific Gains. Figure 6(c) shows higher accuracy for LoopWeaver on several constraint types, while partial variants underperform on some settings. These results support the benchmark-specific benefit of constraint-aware design, rather than unrestricted generalization.

5.6 Inference Efficiency with Offline Preference Optimization (RQ4)

Figure 7 examines the efficiency-accuracy trade-off and inference cost breakdown on Qwen2.5-7B and LLaMA3.1-8B, decomposing the cost into Planning, Filtering, and Generation stages.



(a) Efficiency–accuracy trade-off.

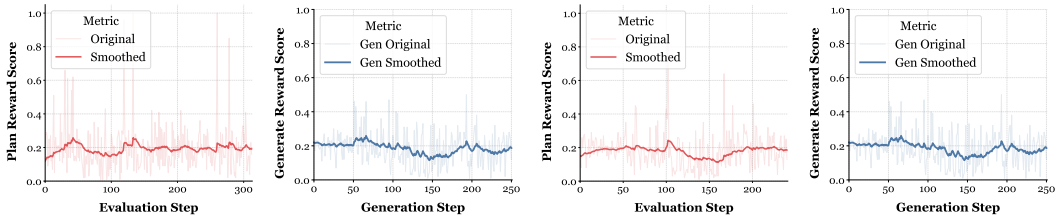
(b) Audited inference-cost breakdown and overall constraint accuracy across methods.

Figure 7: Efficiency analysis of LoopWeaver against baselines on two backbones: **(a)** efficiency–accuracy trade-off. **(b)** audited inference-cost breakdown and overall constraint-accuracy comparison across methods; this panel is a method comparison, not an inference-stage trajectory.

Improved Accuracy with Moderate Overhead. LoopWeaver improves constraint-following accuracy across Qwen2.5-7B and LLaMA3.1-8B over the base model and CogWriter on the audited settings (Figure 7(a)). Compared to CogWriter's static pipeline, LoopWeaver adds complete-plan construction and whole-plan filtering on the inference path, while preference-data construction and the two DPO stages are offline; the audited phase-wise costs are reported in Appendix G.2 and Table 6.

5.7 Offline Preference Coordination (RQ5)

Figure 8 presents plan-level and generation-level reward traces for LLaMA3.1-8B and Qwen2.5-7B. These curves show how the recorded rewards evolve over optimization steps, characterizing the training dynamics of the two offline preference optimization stages.



(a) LLaMA3.1-8B (b) LLaMA3.1-8B (c) Qwen2.5-7B (d) Qwen2.5-7B

Figure 8: (a–b) Plan-level and generation-level reward convergence for LLaMA3.1-8B. (c–d) Plan-level and generation-level reward convergence for Qwen2.5-7B, demonstrating stable optimization across the two levels.

Offline Preference Optimization Enables Stable Improvement. The shared reward provides a consistent criterion for planning and generation. As shown in Figure 8, reward traces characterize optimization dynamics across backbones. Table 2 further assesses the benefits of combining both stages. This coordination occurs during offline training, while inference remains feed-forward.

5.8 Case Study: Prompt-Based Generation Across Qwen2.5 Variants (RQ6)

We present a comparative case study using three Qwen2.5 variants of different sizes under identical prompts and constraints to generate life narratives across varying model scales and capacities.

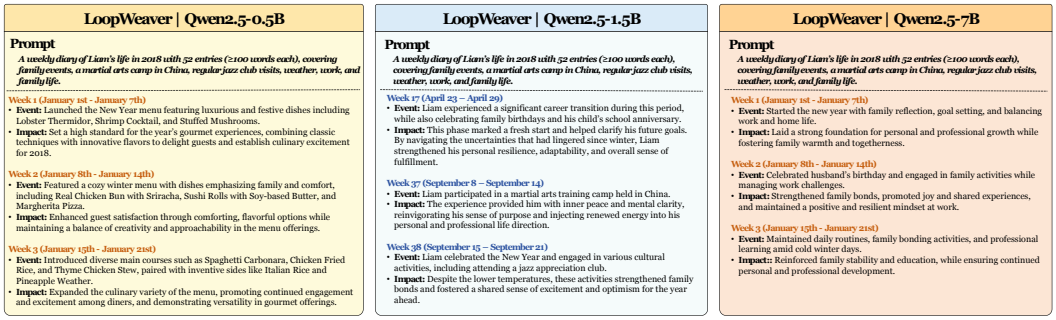


Figure 9: Case study illustrating generation quality under a representative constraint prompt.

In this case study, the 7B variant shows greater thematic depth and stronger cross-paragraph consistency than the smaller variants, which more often exhibit fragmented logic, repetitive content, and weaker long-range coherence. The 7B output develops its themes through more specific details and clearer connections between events, allowing paragraphs to contribute to a coherent narrative. Its transitions provide a smoother progression of ideas, while elaboration helps explain the significance of events rather than merely listing them. In contrast, the smaller variants tend to revisit similar descriptions without advancing the narrative, and their paragraphs are less consistently connected. The 7B variant therefore stands out in both the richness of its content and the continuity of its development, sustaining a more detailed and cohesive account across the generated text.

6 Conclusion

In this work, we introduce LoopWeaver, a hierarchical workflow for constrained long-form text generation. By integrating *constraint-aware planning*, *feasibility filtering*, and *reward-based preference optimization*, LoopWeaver coordinates planning and generation through two sequential offline DPO stages that share a deterministic reward. Experiments on four LongGenBench-style structured scenarios show improvements in both average text quality and overall constraint-following accuracy across all four open-source backbones. LoopWeaver achieves the highest average scores among the compared methods within each backbone group, demonstrating the benefits of combining structured planning with preference-optimized generation. Future work will improve planning reliability for smaller models and extend evaluation to long-form tasks with complex structural requirements.

References

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

[2] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[3] Yushi Bai, Jiajie Zhang, Xin Lv, Linzhi Zheng, Siqi Zhu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longwriter: Unleashing 10, 000+ word generation from long context llms. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=kQ5s9Yh0WI.

[4] Mohaddeseh Bastan, Mihai Surdeanu, and Niranjan Balasubramanian. NEUROSTRUCTURAL DECODING: neural text generation with structural constraints. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 9496–9510. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.ACL-LONG.528. URL https://doi.org/10.18653/v1/2023.acl-long.528.

[5] Alexandre Bonlarron, Florian Régin, Elisabetta De Maria, and Jean-Charles Régin. Large language model meets constraint propagation. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, pages 10036–10044. ijcai.org, 2025. doi: 10.24963/IJCAI.2025/1115. URL https://doi.org/10.24963/ijcai.2025/1115.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[7] Howard Chen, Huihan Li, Danqi Chen, and Karthik Narasimhan. Controllable text generation with language constraints. *CoRR*, abs/2212.10466, 2022. doi: 10.48550/ARXIV.2212.10466. URL https://doi.org/10.48550/arXiv.2212.10466.

[8] Xiang Chen and Xiaojun Wan. Evaluating, understanding, and improving constrained text generation for large language models. *arXiv preprint arXiv:2310.16343*, 2023.

[9] Yuyang Cheng, Linyue Cai, Changwei Peng, Yumiao Xu, Rongfang Bie, and Yong Zhao. Creagentive: An agent workflow driven multi-category creative generation engine. *CoRR*, abs/2509.26461, 2025. doi: 10.48550/ARXIV.2509.26461. URL https://doi.org/10.48550/arXiv.2509.26461.

[10] Woon Sang Cho, Pengchuan Zhang, Yizhe Zhang, Xiujun Li, Michel Galley, Chris Brockett, Mengdi Wang, and Jianfeng Gao. Towards coherent and cohesive long-form text generation. In *Proceedings of the First Workshop on Narrative Understanding*, pages 1–11, 2019.

[11] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391, 2022.

[12] Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Workflowllm: Enhancing workflow orchestration capability of large language models. In *The Thirteenth International Conference on Learning*

Representations, ICLR 2025, Singapore, April 24-28, 2025. OpenReview.net, 2025. URL https://openreview.net/forum?id=3Hy00Wvabi.

[13] Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *CoRR*, abs/2504.15257, 2025. doi: 10.48550/ARXIV.2504.15257. URL https://doi.org/10.48550/arXiv.2504.15257.

[14] Hongchao Gu, Dexun Li, Kuicai Dong, Hao Zhang, Hang Lv, Hao Wang, Defu Lian, Yong Liu, and Enhong Chen. RAPID: efficient retrieval-augmented long text generation with writing planning and information discovery. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 16742–16763. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.859/.

[15] Mustafa Omer Gul and Yoav Artzi. Cogen: Learning from feedback with coupled comprehension and generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 12966–12982. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.721. URL https://doi.org/10.18653/v1/2024.emnlp-main.721.

[16] Jinyi Han, Xinyi Wang, Haiquan Zhao, Tingyun Li, Zishang Jiang, Sihang Jiang, Jiaqing Liang, Xin Lin, Weikang Zhou, Zeye Sun, Fei Yu, and Yanghua Xiao. A stitch in time saves nine: Proactive self-refinement for language models. *CoRR*, abs/2508.12903, 2025. doi: 10.48550/ARXIV.2508.12903. URL https://doi.org/10.48550/arXiv.2508.12903.

[17] Jiafan He, Huizhuo Yuan, and Quanquan Gu. Accelerated preference optimization for large language model alignment. *CoRR*, abs/2410.06293, 2024. doi: 10.48550/ARXIV.2410.06293. URL https://doi.org/10.48550/arXiv.2410.06293.

[18] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

[19] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

[20] Zhiqiang Kou, Junyang Chen, Xin-Qiang Cai, Xiaobo Xia, Ming-Kun Xie, Dong-Dong Wu, Biao Liu, Yuheng Jia, Xin Geng, Masashi Sugiyama, and Tat-Seng Chua. Positive-unlabeled reinforcement learning distillation for on-premise small models, 2026. URL https://arxiv.org/abs/2601.20687.

[21] Zhiqiang Kou, Junxiang Wu, Wenke Huang, Wenwen He, Ming-Kun Xie, Changwei Wang, Yuheng Jia, Di Jiang, Yang Liu, Xin Geng, and Qiang Yang. Fedharmony: Harmonizing heterogeneous label correlations in federated multi-label learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

[22] Yukyung Lee, Soonwon Ka, Bokyung Son, Pilsung Kang, and Jaewook Kang. Navigating the path of writing: Outline-guided text generation with large language models. In Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 3: Industry Track, Albuquerque, New Mexico, USA, April 30, 2025*, pages 233–250. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-INDUSTRY.20. URL https://doi.org/10.18653/v1/2025.naacl-industry.20.

[23] Boyi Li, Zhonghan Zhao, Der-Horng Lee, and Gaoang Wang. Adaptive graph pruning for multi-agent communication. In Inês Lynce, Nello Murano, Mauro Vallati, Serena Villata, Federico Chesani, Michela Milano, Andrea Omicini, and Mehdi Dastani, editors, *ECAI 2025*

- 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025), Frontiers in Artificial Intelligence and Applications, pages 4305–4312. IOS Press, 2025. doi: 10.3233/FAIA251326. URL https://doi.org/10.3233/FAIA251326.

[24] Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor, editors, *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 23142–23150. AAAI Press, 2026. doi: 10.1609/AAAI.V40I28.39481. URL https://doi.org/10.1609/aaai.v40i28.39481.

[25] Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic RL. *CoRR*, abs/2508.13167, 2025. doi: 10.48550/ARXIV.2508.13167. URL https://doi.org/10.48550/arXiv.2508.13167.

[26] Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, and Zhiyu Li. Controllable text generation for large language models: A survey. *CoRR*, abs/2408.12599, 2024. doi: 10.48550/ARXIV.2408.12599. URL https://doi.org/10.48550/arXiv.2408.12599.

[27] Yi Liang, You Wu, Honglei Zhuang, Li Chen, Jiaming Shen, Yiling Jia, Zhen Qin, Sumit Sanghai, Xuanhui Wang, Carl Yang, and Michael Bendersky. Integrating planning into single-turn long-form text generation. *CoRR*, abs/2410.06203, 2024. doi: 10.48550/ARXIV.2410.06203. URL https://doi.org/10.48550/arXiv.2410.06203.

[28] Yi-Cheng Lin, Kang-Chieh Chen, Zhe-Yan Li, Tzu-Heng Wu, Tzu-Hsuan Wu, Kuan-Yu Chen, Hung-yi Lee, and Yun-Nung Chen. Creativity in llm-based multi-agent systems: A survey. *CoRR*, abs/2505.21116, 2025. doi: 10.48550/ARXIV.2505.21116. URL https://doi.org/10.48550/arXiv.2505.21116.

[29] Hanchao Liu, Rongjun Li, Weimin Xiong, Ziyu Zhou, and Wei Peng. Workteam: Constructing workflows from natural language with multi-agents. In Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 3: Industry Track, Albuquerque, New Mexico, USA, April 30, 2025*, pages 20–35. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-INDUSTRY.3. URL https://doi.org/10.18653/v1/2025.naacl-industry.3.

[30] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguistics*, 12:157–173, 2024. doi: 10.1162/TACL_A_00638. URL https://doi.org/10.1162/tacl_a_00638.

[31] Do Xuan Long, Duong Ngoc Yen, Do Xuan Trong, Anh Tuan Luu, Kenji Kawaguchi, Shafiq Joty, Min-Yen Kan, and Nancy F. Chen. Beyond in-context learning: Aligning long-form generation of large language models via task-inherent attribute guidelines. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 3377–3411. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.176/.

[32] Haoran Luo, Haihong E, Guanting Chen, Qika Lin, Yikai Guo, Fangzhi Xu, Zemin Kuang, Meina Song, Xiaobao Wu, Yifan Zhu, and Luu Anh Tuan. Graph-r1: Towards agentic graphrag framework via end-to-end reinforcement learning, 2025. URL https://arxiv.org/abs/2507.21892.

[33] Haoran Luo, Haihong E, Yikai Guo, Qika Lin, Xiaobao Wu, Xinyu Mu, Wenhao Liu, Meina Song, Yifan Zhu, and Anh Tuan Luu. KBQA-o1: Agentic knowledge base question answering with Monte Carlo tree search. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 41177–41199. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/luo25d.html.

[34] Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: A modular approach to automated agentic workflow generation. *CoRR*, abs/2501.07834, 2025. doi: 10.48550/ARXIV.2501.07834. URL https://doi.org/10.48550/arXiv.2501.07834.

[35] Chau Pham, Simeng Sun, and Mohit Iyyer. Suri: Multi-constraint instruction following in long-form text generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 1722–1753. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-EMNLP.94. URL https://doi.org/10.18653/v1/2024.findings-emnlp.94.

[36] Bowen Ping, Jiali Zeng, Fandong Meng, Shuo Wang, Jie Zhou, and Shanghang Zhang. LongDPO: Unlock better long-form generation abilities for LLMs via critique-augmented stepwise information. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7613–7632, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.395. URL https://aclanthology.org/2025.findings-acl.395/.

[37] Patrick Pynadath and Ruqi Zhang. Controlled LLM decoding via discrete auto-regressive biasing. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=Duuerhutvq.

[38] Haoran Que, Feiyu Duan, Liqun He, Yutao Mou, Wangchunshu Zhou, Jiaheng Liu, Wenge Rong, Zekun Moore Wang, Jian Yang, Ge Zhang, Junran Peng, Zhaoxiang Zhang, Songyang Zhang, and Kai Chen. Hellobench: Evaluating long text generation capabilities of large language models. *CoRR*, abs/2409.16191, 2024. doi: 10.48550/ARXIV.2409.16191. URL https://doi.org/10.48550/arXiv.2409.16191.

[39] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[40] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[41] Kashob Kumar Roy, Pritom Saha Akash, Kevin Chen-Chuan Chang, and Lucian Popa. ConTReGen: Context-driven tree-structured retrieval for open-domain long-form text generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13773–13784, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.807. URL https://aclanthology.org/2024.findings-emnlp.807/.

[42] Alireza Salemi, Cheng Li, Mingyang Zhang, Qiaozhu Mei, Weize Kong, Tao Chen, Zhuowan Li, Michael Bendersky, and Hamed Zamani. Reasoning-enhanced self-training for long-form personalized text generation. *arXiv preprint arXiv:2501.04167*, 2025.

[43] Yijia Shao, Yucheng Jiang, Theodore A. Kanell, Peter Xu, Omar Khattab, and Monica S. Lam. Assisting in writing wikipedia-like articles from scratch with large language models. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 6252–6278. Association for Computational Linguistics, 2024. doi: 10.18653/

V1/2024.NAACL-LONG.347. URL https://doi.org/10.18653/v1/2024.naacl-long.347.

[44] Yuntao Shi, Yi Luo, Yeyun Gong, and Chen Lin. Hicam: A hierarchical-causal modification framework for long-form text modification. *CoRR*, abs/2505.24319, 2025. doi: 10.48550/ARXIV.2505.24319. URL https://doi.org/10.48550/arXiv.2505.24319.

[45] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[46] Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

[47] Kaiyang Wan, Honglin Mu, Rui Hao, Haoran Luo, Tianle Gu, and Xiuying Chen. A cognitive writing perspective for constrained long-form text generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 9832–9844. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.511/.

[48] Chenglong Wang, Hang Zhou, Yimin Hu, Yifu Huo, Bei Li, Tongran Liu, Tong Xiao, and Jingbo Zhu. ESRL: efficient sampling-based reinforcement learning for sequence generation. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19107–19115. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29878. URL https://doi.org/10.1609/aaai.v38i17.29878.

[49] Qianyue Wang, Jinwu Hu, Zhengping Li, Yufeng Wang, Daiyuan Li, Yu Hu, and Mingkui Tan. Generating long-form story using dynamic hierarchical outlining with memory-enhancement. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 1352–1391. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.63. URL https://doi.org/10.18653/v1/2025.naacl-long.63.

[50] Wenxiao Wang, Priyatham Kattakinda, and Soheil Feizi. Maestro: Joint graph & config optimization for reliable ai agents. *arXiv preprint arXiv:2509.04642*, 2025.

[51] Yingming Wang and Pepa Atanasova. Self-critique and refinement for faithful natural language explanations. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8481–8507, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.427. URL https://aclanthology.org/2025.emnlp-main.427/.

[52] Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 24013–24035. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1170/.

[53] Yangbo Wei, Zhen Huang, Lei He, Li Huang, Ting-Jung Lin, and Wei W Xing. Vflow: Discovering optimal agentic workflows for verilog generation. In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 355–361. IEEE, 2026.

[54] Yuhao Wu, Yushi Bai, Zhiqiang Hu, Roy Ka-Wei Lee, and Juanzi Li. Longwriter-zero: Mastering ultra-long text generation via reinforcement learning. *arXiv preprint arXiv:2506.18841*, 2025.

[55] Yuhao Wu, Yushi Bai, Zhiqiang Hu, Juanzi Li, and Roy Ka-Wei Lee. Superwriter: Reflection-driven long-form generation with large language models. *CoRR*, abs/2506.04180, 2025. doi: 10.48550/ARXIV.2506.04180. URL https://doi.org/10.48550/arXiv.2506.04180.

[56] Yuhao Wu, Yushi Bai, Zhiqing Hu, Shangqing Tu, Ming Shan Hee, Juanzi Li, and Roy Ka-Wei Lee. Shifting long-context llms research from input to output. *arXiv preprint arXiv:2503.04723*, 2025.

[57] Yuhao Wu, Ming Shan Hee, Zhiqiang Hu, and Roy Ka-Wei Lee. Longgenbench: Benchmarking long-form generation in long context llms. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=3A71qNKWAS.

[58] Zekun Xi, Wenbiao Yin, Jizhan Fang, Jialong Wu, Runnan Fang, Yong Jiang, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. OmniThink: Expanding knowledge boundaries in machine writing through thinking. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 956–976, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.50. URL https://aclanthology.org/2025.emnlp-main.50/.

[59] Haotian Xia, Hao Peng, Yunjia Qi, Bin Xu, Juanzi Li, Lei Hou, and Xiaozhi Wang. Storywriter: A multi-agent framework for long story generation. In Meeyoung Cha, Chanyoung Park, Noseong Park, Carl Yang, Senjuti Basu Roy, Jessie Li, Jaap Kamps, Kijung Shin, Bryan Hooi, and Lifang He, editors, *Proceedings of the 34th ACM International Conference on Information and Knowledge Management, CIKM 2025, Seoul, Republic of Korea, November 10-14, 2025*, pages 6559–6563. ACM, 2025. doi: 10.1145/3746252.3761616. URL https://doi.org/10.1145/3746252.3761616.

[60] Shengxiang Xu, Jiayi Zhang, Shimin Di, Yuyu Luo, Liang Yao, Hanmo Liu, Jia Zhu, Fan Liu, and Min-Ling Zhang. Robustflow: Towards robust agentic workflow generation. *CoRR*, abs/2509.21834, 2025. doi: 10.48550/ARXIV.2509.21834. URL https://doi.org/10.48550/arXiv.2509.21834.

[61] Xuan Xu, Haolun Li, Zhongliang Yang, Beilin Chu, Jia Song, Moxuan Xu, and Linna Zhou. Topic modeling as long-form generation: Can long-context llms revolutionize ntm via zero-shot prompting? *arXiv preprint arXiv:2510.03174*, 2025.

[62] 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. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024. doi: 10.48550/ARXIV.2412.15115. URL https://doi.org/10.48550/arXiv.2412.15115.

[63] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. *URL https://arxiv. org/abs/2305.10601*, 3:1, 2023.

[64] Shunyu Yao, Howard Chen, Austin W. Hanjie, Runzhe Yang, and Karthik R. Narasimhan. COLLIE: systematic construction of constrained text generation tasks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=kxgSlyirUZ.

[65] Wang You, Wenshan Wu, Yaobo Liang, Shaoguang Mao, Chenfei Wu, Maosong Cao, Yuzhe Cai, Yiduo Guo, Yan Xia, Furu Wei, and Nan Duan. Eipe-text: Evaluation-guided iterative plan extraction for long-form narrative text generation. *CoRR*, abs/2310.08185, 2023. doi: 10.48550/ARXIV.2310.08185. URL https://doi.org/10.48550/arXiv.2310.08185.

[66] Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 6192–6217. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.315. URL https://doi.org/10.18653/v1/2025.naacl-long.315.

[67] Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly. *CoRR*, abs/2502.07373, 2025. doi: 10.48550/ARXIV.2502.07373. URL https://doi.org/10.48550/arXiv.2502.07373.

[68] Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/zhang25bi.html.

[69] Jiajie Zhang, Zhongni Hou, Xin Lv, Shulin Cao, Zhenyu Hou, Yilin Niu, Lei Hou, Yuxiao Dong, Ling Feng, and Juanzi Li. Longreward: Improving long-context large language models with AI feedback. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 3718–3739. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.187/.

[70] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

[71] Mingming Zhao, Xiaokang Wei, Yuanqi Shao, Kaiwen Zhou, Lin Yang, Siwei Rao, Junhui Zhan, and Zhitang Chen. A^2flow: Automating agentic workflow generation via self-adaptive abstraction operators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 29930–29938, 2026.

[72] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

[73] Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming. *arXiv preprint arXiv:2505.22967*, 2025.

[74] Han Zhong, Zikang Shan, Guhao Feng, Wei Xiong, Xinle Cheng, Li Zhao, Di He, Jiang Bian, and Liwei Wang. DPO meets PPO: reinforced token optimization for RLHF. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=IfWKVF6LfY.

[75] Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö Arık. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*, 2025.

[76] Wangchunshu Zhou, Yuchen Eleanor Jiang, Ethan Wilcox, Ryan Cotterell, and Mrinmaya Sachan. Controlled text generation with natural language instructions. In *International Conference on Machine Learning*, pages 42602–42613. PMLR, 2023.

[77] Runchuan Zhu, Bowen Jiang, Lingrui Mei, Fangkai Yang, Lu Wang, Haoxiang Gao, Fengshuo Bai, Pu Zhao, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Adaptflow: Adaptive workflow optimization via meta-learning. *CoRR*, abs/2508.08053, 2025. doi: 10.48550/ARXIV.2508.08053. URL https://doi.org/10.48550/arXiv.2508.08053.

A Prompts Used in LoopWeaver

A.1 Plan Construction Prompt

Week Plan Prompt. The Week Plan Construction Prompt guides the model to create a 52-week plan based on user requirements. It identifies special events and their exact weeks, including periodic events. The model generates a structured weekly plan in JSON format, and a revision step ensures all events strictly follow user specifications.

```
---- Role ----
You are an expert planner with perfect attention to detail. Create a precise 52-week plan that EXACTLY matches ALL user
requirements.
User requirements: {prompt}
---- Goal ----

Think step by step. Analyse the user requirements to identify special events and their exact week and list them in
"special_events". If the event is periodic, list them seperately by number and specify which weeks they will be in (e.g. 1st
[event_name]). Then follow your analysis to create a weekly plan for each week.

---- Output Format ----

Here is an example of the output format:
{{
    "analysis": "",
    "special_events": [
        {{
            "event_name": "Husband's Birthday",
            "week_numb": "May 13th, Week 19"
        }}
        ...
    ],
    "weekly_plan": [
        {{
            "week_id": "Week 1 (January 1st - January 7th)",
            "events": "New Year celebrations and family gathering"
        }},
        ...
    ]
}}

---- Return ----

Return your analysis and plan in ONLY this exact json format:
{{
    "analysis": "",
    "special_events": [
        {{
            "event_name": "event_name",
            "week_numb": "date and week number",
        }}
    ],
    # Strictly follow the special_events for specific weeks
    "weekly_plan": [
        {{
            "week_id": "Week 1 (January 1st - January 7th)",
            "events": "Brifly list special events of this week"
        }},
        ...
    ]
```

Figure 10: Plan Construction Prompt illustration

Floor Plan Prompt. The Floor Plan Prompt instructs the model to create a detailed floor-by-floor plan for a skyscraper. Special facilities are assigned to specific floors, including recurring ones, and the plan is output in JSON format.

```
---- Role ----
You are an expert planner with perfect attention to detail. Create a precise 52-week plan that EXACTLY matches ALL user
requirements.

---- Prompt ----
 User requirements: {prompt}
---- Goal ----

CRITICAL ANALYSIS INSTRUCTIONS:
1. Identify EVERY specific floor mentioned in user requirements
2. Note EXACT purposes, facilities, or special features for each floor
3. Pay attention to floor numbers and their designated uses
4. Identify any special requirements for specific floors
5. Ensure all mentioned facilities are included in the correct floors


 Return your analysis and plan in this EXACT format:
 {{
    "analysis": "Detailed analysis of every floor requirement in the user prompt",
    "special_floors": [
        {{
            "floor_number": "Floor number as specified",
            "purpose": "Exact purpose/use from requirements",
            "special_features": "Any special features mentioned"
        }}
    ],
    "floor_plan": [
        {{
            "floor_id": "Floor 1",
            "purpose": "Exact purpose matching user requirements"
        }}
    ]
 }}
```

Figure 11: Floor Plan Prompt illustration

Menu Plan Prompt. The Menu Plan Prompt generates a 52-week menu schedule, assigning special dishes to specific weeks, including periodic ones.

```
---- Role ----
You are an expert chef, and your task is to create a weekly menu plan containing 52 weeks.
User requirements: {prompt}
---- Goal ----
Think step by step. Analyse the user requirements to identify special dishes and their exact week and list them in
"special_dishes". If the dish is periodic, list them seperately by number and specify which weeks they will be in (e.g. 1st
[dish_name]). Then follow your analysis to create a weekly menu plan for each week.
---- Output Format ----
Here is an example of the output format:
{{
  "analysis": "",
  "special_dishes": [
    {{
      "dish_name": "Winter Solstice Feast featuring Venison
Stew",
      "week_numb": "05-13, Week 19"
    }}
    ...
  ],
  "weekly_plan": [
    {{
      "week_id": "Menu Week 1 (January 1st - January 7th)",
      "dishes": "Australia Day BBQ featuring Lamb Chops"
    }},
    ...
  ]
}}
---- Return ----
Return your analysis and plan in ONLY this exact json format:
{{
  "analysis": "",
  "special_dishes": [
    {{
      "dish_name": "dish_name",
      "week_numb": "date and week number",
    }}
  ],
  # Strictly follow the special_dishes for specific weeks
  "weekly_plan": [
    {{
      "week_id": "Menu Week 1 (January 1st - January 7th)",
      "dishes": "Australia Day BBQ featuring Lamb Chops"
    }},
    ...
  ]
```

Figure 12: Menu Plan illustration

Block Plan Prompt. The Block Plan Prompt guides the design of a 10×10 city block grid, assigning special uses to specific blocks and handling periodic patterns.

```
---- Role ----
You are an expert urban planner. Create a precise city block plan that EXACTLY matches ALL user requirements.

---- Prompt ----
User requirements: {prompt}

---- Goal ----

CRITICAL ANALYSIS INSTRUCTIONS:
1. Identify EVERY specific building, facility, or land use mentioned
2. Note EXACT locations or block specifications
3. Pay attention to zoning requirements, special facilities, or community needs
4. Identify any special features for specific blocks
---- Return format ----

Return your analysis and plan in this EXACT format:
{{
  "analysis": "Detailed analysis of every block planning requirement in the user prompt",
  "special_blocks": [
    {{
      "block_number": "Block identifier as specified",
      "use": "Exact use/purpose from requirements",
      "special_features": "Any special features mentioned"
    }}
  ],
  "block_plan": [
    {{
      "block_id": "Block 1",
      "use": "Exact use matching user requirements"
    }}
  ]
}}
```

Figure 13: Block Plan Prompt illustration

A.2 Writing Prompt

Weekly Diary Prompt. The Weekly Diary Prompt guides the model to write a 200-word diary entry for a given week. It incorporates weekly events, ensures coherence with the overall yearly plan, and checks user requirements for special events. The model outputs a structured JSON containing the week identifier, a reasoning check, and the diary entry text.

```
---- Role ----
You are an expert writer.
---- Goal ----
Write a 200-word weekly diary entry for the week of {week_id}.
The events for this week are: {events}
You should consider the coherence of the diary entry referring to the plan of the whole year: {weekly_plan}
You should consider the user requirements: {prompt}
Check from the user requirements if there are any special events that should be included in the diary entry. If there are,
include them in the diary entry. If there are no special events, write a general diary entry for the week.
---- Return ----
Return the diary entry in the following json format:
{{
    "week_id": "{week_id}",
    "check": "reason and check if the user requirements are met",
    "diary_entry": "Your 200-word diary entry here"
}}
```

Figure 14: Weekly Diary Prompt

Skyscraper Floor Plan Prompt. The Skyscraper Floor Plan Prompt guides the model to design a 150-word floor plan for a specific skyscraper floor. It considers the floor's purpose, maintains coherence with the overall skyscraper plan, and verifies any special requirements from the user. The output is a structured JSON including the floor identifier, a reasoning check, and the generated floor plan.

```
---- Role ----
You are an expert designer.
---- Goal ----
Write a 150-word skyscraper floor plan for the floor of {floor_id}.
The purpose for this floor is: {purpose}
You should consider the coherence of the floor plan by referring to the plan of the whole skyscraper: {floor_plan}
You should consider the user requirements: {prompt}
Check from the user requirements if there are any special requirement that should be included in the floor plan. If
there are, include them in the floor plan. If there are no special events, write a general floor plan.
Return the floor plan for the floor of {floor_id} in the following json format:
---- Return ----
Return the floor plan for the floor of {floor_id} in the following
json format:
{{
    "floor_id": "{floor_id}",
    "check": "reason and check if the user requirements are met",
    "plan": "Your 150-word floor plan here"
}}
```

Figure 15: Skyscraper Floor Plan Prompt

Weekly Menu Plan Prompt. The Weekly Menu Plan Prompt guides the model to create a 200-word menu plan for a given week. It incorporates provided dishes, ensures coherence with the yearly menu plan, and verifies special dishes from the user requirements. The model returns a structured JSON with the week identifier, a reasoning check, and the weekly menu plan.

```
---- Role ----
  You are an expert chef.
---- Goal ----
Write a 200-word weekly menu plan for the week of {week_id}.
The dishes for this week are: {dishes_str}
You should consider the coherence of the menu plan referring to the plan of the whole year: {weekly_plan}
You should consider the user requirements: {prompt}
Check from the user requirements if there are any special dishes that should be included in the menu plan. If there are,
include them in the menu plan. If there are no special dishes, write a general menu plan for the week.
---- Return ----
Return the menu plan in the following json format:
{{
    "week_id": "{week_id}",
    "check": "reason and check if the user requirements are met",
    "week_menu": "Your 200-word menu plan here"
}}
```

Figure 16: Weekly Menu Plan Prompt

City Block Plan Prompt. The City Block Plan Prompt guides the model to design a 150-word city block plan. It considers the intended use of the block, ensures coherence with the overall city plan, and checks for any special user requirements. The generated output is returned in JSON format, including the block identifier, a reasoning check, and the block plan.

```
---- Role ----
You are an expert designer.
---- Goal ----
Write a 150-word city block plan for the block of {block_id}.
The use for this block is: {use}
You should consider the coherence of the block plan by referring to the plan of the whole city: {block_plan}

You should consider the user requirements: {prompt}
Check from the user requirements if there are any special requirement that should be included in the block plan. If
there are, include them in the block plan. If there are no special events, write a general block plan.
---- Return ----
Return the block plan for the block of {block_id} in the following
json format:
{{
    "block_id": "{block_id}",
    "check": "reason and check if the user requirements are met",
    "plan": "Your 150-word block plan here"
}}
```

Figure 17: City Block Plan Prompt

A.3 Binary decisions Prompt

As shown in Figure 18, this prompt template evaluates a plan's relevance to a given question. The judgment (YES/NO) is based on six criteria: direct relevance, completeness, logical coherence, efficiency, specificity, and consistency with the prompt context. Examples illustrate common failures—such as irrelevant steps, missing actions, or illogical order. The structured format includes the prompt, the plan, and the binary relevance label, enabling systematic assessment of planning quality.

---- Goal----

Given the following prompt and plan, retur YES if the plan is relevant to the question and NO if it isn't. A plan is considered relevant only if it meets of the following criteria:

1. Direct Relevance: The plan must directly address the question posed in the prompt. It should not deviate from the main objective.
2. Completeness: The plan must include all necessary steps to achieve the goal described in the prompt. Missing critical steps wil result in the planbeing deemed irrelevant.
3. Logical Coherence: The plan must folow a logical sequence of actions. ach step should clearly lead to the next, and there should be no logical gap or contradictions.
4. Efficiency: The plan should not include unnecessary or redundant steps. Each step must contribute directly to the final goal.
5. Specificity: The plan must be specific and actionable. Vague or overly broad steps will not be considered relevant.
6. Contextual Consistency. The plan must be consistent with the context provided in the prompt. Any steps that are in consistent with the background information or constraints will be deemed irrelevant.

---- Eample of Irrelewant Plans ----

- Irelevant Content:If the prompt asks for a plan to "design an experiment to test plant photosynthesis," but the plan includes steps like "build a rocket model," this plan will be rejected.
- Incomplete Plan: If the plan omits critical steps, such as "measure the photosynthesis rate," it will be considered irrelevant.
- logicalSequence:If the steps in the plan do not followa logical order, such as "measure plant growth first, then plant the seeds, the plan will be rejected.

---- Prompt ----

{prompts}

---- Plan----

{weekly_plan} / {floor_plan} / {menu_weekly_plan} / {block_plan}

---- Relevant----

(YES / NO)

Figure 18: Prompt for binary decisions

B Conditional Analysis and Empirical Verification

B.1 Conditional analysis of planning refinement

Empirical Observation 1. *Constraint-aware hierarchical planning is associated with improved plan validity and coverage under the stated monotonicity assumption.*

Conditional analysis. To formalize the analysis, we introduce a mild and sufficient assumption on the refinement operator.

Assumption 1 (Monotonic Improvement under Constraint-Aware Refinement). Let p denote a hierarchical plan and $\mathcal{R}$ the refinement operator described in Section 4.1. We assume that there exists a plan quality function $f(p) \in \mathbb{R}$, measuring constraint satisfaction and structural validity, such that

$$\mathbb{E}\big[f(\mathcal{R}(p)) \mid p\big] \geq f(p). \tag{14}$$

This assumption does not claim that arbitrary refinement steps improve plan quality. Instead, it formalizes a sufficient condition satisfied by our constraint-aware hierarchical refinement, as shown below.

Let a hierarchical plan be decomposed into sub-plans

$$p_i = (S_{i,1}, S_{i,2}, \ldots, S_{i,m_i}), \quad S_{i,j} \in \mathcal{S}, \tag{15}$$

where each sub-plan $S_{i,j}$ is associated with the local constraint subset $C_{i,j} \subseteq C$ inherited from the global constraint set C.

We define the plan quality function as an additive decomposition over sub-plans:

$$f(p_i) \;=\; \sum_{j=1}^{m_i} f_j(S_{i,j} \mid C_{i,j})\,, \tag{16}$$

where each local score $f_j(\cdot)$ is nonnegative and evaluates the constraint satisfaction and structural validity of sub-plan $S_{i,j}$.

For the benchmark-level reporting, constraint coverage is defined as

$$q(p; C) = \frac{1}{|C|} \sum_{c \in C} \mathbb{I}[p \models c], \tag{17}$$

where $p \models c$ indicates that plan p satisfies constraint c. We use validity to denote the corresponding binary satisfaction of the stated structural requirements; these definitions are protocol-specific and do not constitute a formal guarantee of document-level coherence.

The following conditional lemma makes the implication explicit. Let $\mathcal{R}(P; x, C)$ denote the complete-plan refinement operator. If, for every constraint $c \in C$,

$$\mathbb{E}[\mathbb{I}[\mathcal{R}(P; x, C) \models c] \mid P, x, C] \geq \mathbb{I}[P \models c], \tag{18}$$

then, by linearity of expectation,

$$\mathbb{E}[q(\mathcal{R}(P; x, C); C) \mid P, x, C] \geq q(P; C). \tag{19}$$

This is a sufficient conditional assumption for non-decreasing expected coverage, not a universal guarantee that every refinement improves every plan.

At each refinement step, the hierarchical planner identifies a violating sub-plan $S_{i,k}$ and applies a localized revision within the complete-plan context, producing an updated sub-plan $S'_{i,k}$ while preserving the other sub-plans for the consistency check:

$$\begin{aligned} p'_i = (&S_{i,1}, \ldots, S_{i,k-1}, S'_{i,k}, \\ &S_{i,k+1}, \ldots, S_{i,m_i}). \end{aligned} \tag{20}$$

Consequently, the change in plan quality depends only on the revised sub-plan:

$$\begin{aligned} &\mathbb{E}[f(p'_i) \mid p_i] - f(p_i) \\ &= \mathbb{E}\big[f_k(S'_{i,k}) \mid S_{i,k}\big] - f_k(S_{i,k}). \end{aligned} \tag{21}$$

Since the refinement operator $\mathcal{R}$ is constraint-aware and employs validity filtering and top-K selection (Section 4.1), refinements that reduce local validity or violate constraints are rejected with high probability. Therefore,

$$\mathbb{E}\big[f_k(S'_{i,k}) \mid S_{i,k}\big] \geq f_k(S_{i,k}), \tag{22}$$

which implies

$$\mathbb{E}[f(p'_i) \mid p_i] \geq f(p_i). \tag{23}$$

By iteratively applying hierarchical refinement, the expected plan quality is non-decreasing under Assumption 1. This conditional statement does not guarantee that every candidate plan satisfies all constraints or structural requirements.

We emphasize that this argument relies critically on hierarchical decomposition and localized refinement. For flat planning or unconstrained revision processes, modifying one component may arbitrarily affect others, and the above analysis no longer holds.

Conclusion. Under these assumptions, the expected plan quality is non-decreasing after refinement.

B.2 Conditional analysis of feasibility filtering

Empirical Observation 2. *Feasibility filtering is associated with fewer misaligned plans passed to generation under the stated rejective-power assumption.*

Conditional analysis. Let $P_{\text{final}} = \{p_1, \ldots, p_N\}$ be the set of candidate plans after refinement. Each plan p_i is evaluated by a template-driven, feasibility-oriented screening mechanism and assigned a binary acceptance indicator

$$\delta_i = \mathbf{1}_{\text{relevant}}(p_i) \in \{0, 1\}, \tag{24}$$

where $\delta_i = 1$ indicates that the plan satisfies the task-inherent relevance criteria defined in Section 4.2, and $\delta_i = 0$ otherwise. The filtered plan set is given by

$$P_{\text{filtered}} = \{p_i \in P_{\text{final}} \mid \delta_i = 1\}. \tag{25}$$

Let $\phi(p) \geq 0$ denote a nonnegative violation score measuring the degree to which plan p violates task constraints or semantic requirements, with $\phi(p) = 0$ indicating full validity.

We do not assume that the plan generator produces valid plans with high probability, nor that the screening mechanism perfectly separates high- and low-quality plans. Instead, we impose a minimal and realistic assumption on the feasibility-oriented screening process.

Assumption 2 (Non-trivial Rejective Power). There exist constants $\tau > 0$ and $\eta > 0$ such that for any plan p with $\phi(p) \geq \tau$,

$$\Pr(\delta = 0 \mid p) \geq \eta. \tag{26}$$

That is, plans exhibiting sufficiently large violations are rejected with non-zero probability. Here, the probability notation abstracts over the stochasticity introduced by the template-driven LLM evaluator and decoding process, rather than assuming a calibrated probabilistic classifier.

This assumption is weak and does not require perfect discrimination; it only formalizes that the screening mechanism has some ability to reject clearly invalid plans.

Under this assumption, the probability that a retained plan exhibits a large violation is bounded as

$$\Pr\big(\phi(p) \geq \tau \mid \delta = 1\big) \leq \frac{\Pr\big(\phi(p) \geq \tau\big)(1 - \eta)}{\Pr(\delta = 1)}. \tag{27}$$

The bound above provides an upper bound on the violation risk among retained plans. If the additional condition $\Pr(\delta = 1) > 1 - \eta$ holds, this bound is strictly lower than the unfiltered violation probability. Without this additional condition, the analysis does not imply a strict reduction, but it still characterizes the risk induced by feasibility-oriented screening. Thus, the filter can reduce the risk of propagating highly invalid or semantically misaligned plans to downstream generation under the stated condition.

B.3 Empirical verification of reward-based preference optimization

Empirical Observation 3. *Reward-based preference optimization aligns plans and segments with task objectives.*

Empirical verification. After binary relevance filtering, each plan is decomposed into sub-plans for optimization, followed by content generation guided by the sub-plan. For each candidate X (a sub-plan S_j or a generated segment G_k), let $c(X)$ denote its conditioning context: $c(S_j) = (x, C)$ at the plan level and $c(G_k) = (x, C, S_k)$ at the generation level. We define its reward by direct deterministic evaluation:

$$r(X) = R\big(X,\ (x, C)\big), \tag{28}$$

where R is the shared rule-based reward defined in Section 3.

Recall the DPO objective used in the paper:

$$\begin{aligned}\Delta(X^+, X^-) &= \beta \log \frac{\pi_\theta(X^+ \mid c(X))}{\pi_{\text{ref}}(X^+ \mid c(X))} \\ &\quad - \beta \log \frac{\pi_\theta(X^- \mid c(X))}{\pi_{\text{ref}}(X^- \mid c(X))}, \\ \mathcal{L}_{\text{DPO}}(\theta) &= -\mathbb{E}_{(X^+, X^-)\sim D} \\ &\quad \Big[\log \sigma\big(\Delta(X^+, X^-)\big)\Big].\end{aligned} \tag{29}$$

This objective encourages the model policy π_θ to increase the relative log-probability ratio for higher-reward candidates (X^+) and decrease it for lower-reward candidates (X^-), proportionally to the preference signal induced by the reward difference.

In the implemented training procedure, preference labels are binary: X^+ is selected when $r(X^+) > r(X^-)$, while X^- is the lower-reward candidate. No soft reward-probability label is used. The DPO objective therefore increases the relative likelihood of the higher-reward candidate under the shared conditioning context $c(X)$. The resulting alignment is evaluated empirically in the main text.

Because preference pairs are ranked directly by the deterministic reward, the DPO objective shifts model likelihood toward candidates with higher constraint coverage and prompt relevance. The resulting alignment is evaluated empirically in the main text.

Conclusion. The analysis describes the direction of the DPO preference signal; quantitative gains are established by the experiments in the main text.

C LoopWeaver Algorithm Details

Algorithm 1 realizes the three components of Section 4 together with the offline DPO training that couples the two levels: (1) *Constraint-aware hierarchical planning* (Section 4.1) generates a complete plan from the writing prompt and constraint set C, revises sub-plans in full-plan context, and retains the resulting candidates. (2) *Feasibility filtering* (Section 4.2) applies one prompt-only YES/NO decision to each complete plan using the six task-inherent criteria. (3) *Reward-based preference optimization* (Section 4.3) constructs preference pairs at both plan and generation levels using the deterministic reward $R(\cdot, e)$ defined in Section 3, and updates the policy via the DPO objectives $\mathcal{L}_{\text{DPO}}^{\text{plan}}$ and $\mathcal{L}_{\text{DPO}}^{\text{gen}}$. The shared reward couples the two levels during offline training; inference is feed-forward and incurs no live feedback overhead.

Algorithm 1 LoopWeaver: Hierarchical Offline Preference Optimization for Constrained Long-Form Text Generation

Require: Prompt x, structured constraint set $C = C_{\text{once}} \cup C_{\text{range}} \cup C_{\text{periodic}}$, policy π_θ, reference policy π_{ref}, deterministic rule-based reward $R(\cdot, e)$ with $e = (x, C)$ as defined in Section 3 (ii), candidate plan count K, DPO temperature β

Ensure: Updated policy π_θ

// Component 1: Constraint-Aware Plan Modeling (Section 4.1)
for $i = 1$ to K **do**
    Sample complete candidate plan $p_i \sim \pi_\theta(\cdot \mid x, C)$
    Revise each sub-plan in full-plan context: $p_i \leftarrow \mathcal{R}(p_i; x, C)$ ▷ Eq. 8
end for
$P_{\text{final}} \leftarrow \{p_1, \ldots, p_K\}$
// Component 2: Binary Relevance Filtering (Section 4.2)
for each $p_i \in P_{\text{final}}$ **do**
    Query the same backbone with the prompt-only YES/NO filter
    Set $\delta_i = \mathbf{1}[\, p_{\text{filter}}(p_i, x, C) = \text{YES}\,]$
end for
$P_{\text{filtered}} \leftarrow \{p_i \in P_{\text{final}} \mid \delta_i = 1\}$
// Component 3: Reward-Guided Optimization (Section 4.3)
Plan-level DPO:
for each plan $p \in P_{\text{filtered}}$ and each sub-plan $S_j \in p$ **do**
    Compute deterministic unit reward $r(S_j) = R(S_j, e)$ ▷ Eq. 11
end for
Construct preference pairs (S_j^+, S_j^-) with $r(S_j^+) > r(S_j^-)$ to form D_{plan}
Update π_θ by minimizing $\mathcal{L}_{\text{DPO}}^{\text{plan}}(\theta)$ over D_{plan}
Generation-level DPO:
for each sub-plan S_k from updated π_θ **do**
    Sample candidate segments $G_k \sim \pi_\theta(\cdot \mid x, S_k)$
    Compute deterministic unit reward $r(G_k) = R(G_k, e)$
end for
Construct preference pairs (G_k^+, G_k^-) with $r(G_k^+) > r(G_k^-)$ to form D_{gen}
Update π_θ by minimizing $\mathcal{L}_{\text{DPO}}^{\text{gen}}(\theta)$ over D_{gen}
return updated policy π_θ

D Dataset Details

Our dataset is constructed based on the official LongGenBench codebase, following the methodology described in Section 4 of the main paper. It comprises **400 synthetic long-form planning and generation tasks**, spanning diverse domains such as weekly event planning, architectural floor designs, menu organization, and city block descriptions. Each task instance is automatically paired with structured prompts, hierarchical global plans, task-specific constraints (once, range, or periodic), and evaluation prompts, enabling the dataset to jointly capture global planning information and local generation requirements. Each sample includes **(i)** a global hierarchical plan, **(ii)** embedded structural constraints, **(iii)** sub-plan prompts used for preference optimization, and **(iv)** the final generation targets. On average, global plans contain 250–300 words, while individual generated segments contain 150–200 words depending on the task type.

Although the dataset is synthetically generated, this construction is both appropriate and necessary for our benchmark. Real-world human-authored corpora rarely contain explicit structural constraints—such as hierarchical outlines, length-range specifications, and periodic content formats—at the scale and diversity required for systematic evaluation. Synthetic generation enables precise and reproducible control over constraint complexity, ensuring consistent coverage across task categories without compromising the realism of structural patterns. Importantly, the dataset serves as a *task specification* rather than a corpus for style imitation: LoopWeaver learns to follow constraints, not to mimic LLM-produced writing styles. Since the model never encounters test instances during training and DPO relies solely on preference signals rather than memorizing dataset patterns, the synthetic origin does not introduce data leakage. Moreover, the structural constraints modeled in this dataset—such as multi-level sectioning, fixed-length segments, and periodic organization—naturally occur in real-world long-form writing, including reports, educational documents, and technical manuals, making the resulting capabilities transferable to human-authored scenarios.

The 400 benchmark samples described above are reserved for held-out evaluation when comparing different model backbones (Qwen2.5, GPT-4o-mini, and LLaMA3.1) as reported in Section 5. DPO preference pairs are generated from a separate synthetic training split using the same task templates but different prompts and seeds; none of the 400 evaluation prompts or outputs is used in DPO training. To ensure reproducibility, we adopt identical sampling parameters as the official LongGenBench implementation. Since all samples are generated from synthetic prompts and LLM-produced outputs, the dataset contains no personally identifiable information (PII) or sensitive content and is intended solely for academic research on long-form planning and generation.

E Baseline Details

GPT-4o-mini. In our experiments, we evaluate the proposed **LoopWeaver** framework against **CogWriter**, a multi-step cognitive planning and generation framework specifically designed for long-form text. CogWriter performs hierarchical planning to structure content and leverages GPT-4o-mini for each generation step, allowing it to handle complex long-form tasks with temporal and spatial constraints.

Qwen2.5. For models based on Qwen2.5 across multiple scales (0.5B, 1.5B, and 7B), we also compare against both **CogWriter** and **LongWriter**. In this setting, CogWriter utilizes Qwen2.5 as the generation backbone, employing hierarchical planning and sub-plan guided generation to produce coherent outputs for long-form scenarios such as diary writing, menu design, skyscraper floor planning, and urban block layouts. LongWriter, by contrast, relies on extended-context training and direct generation without explicit planning. Including these baselines across different model capacities enables a systematic evaluation of how LoopWeaver improves over strong planning-augmented and data-centric long-form generation methods under varying model scales.

LLaMA3.1-8B. To provide a cross-backbone comparison, we include experiments based on LLaMA3.1-8B. For this backbone, we compare LoopWeaver against **CogWriter** and **LongWriter** as representative long-form generation baselines, as well as the base LLaMA3.1-8B model without workflow augmentation. This setting tests whether the workflow-level planning and reward-guided optimization transfer across backbones within the same structured benchmark; it is not evidence of unrestricted cross-domain generalization.

F Evaluation Details

For each constraint type, we compute accuracy as the fraction of prompts for which the model output satisfies the constraint:

(a) Once constraints (C_{once}): checks adherence to individual constraints.

$$\text{Acc}_{\text{once}} = \frac{\sum_{i=1}^{N_{\text{once}}} R_i}{N_{\text{once}}} \tag{30}$$

(b) Range constraints (C_{range}): checks adherence to constraints that span multiple blocks or a specified range.

$$\text{Acc}_{\text{range}} = \frac{\sum_{i=1}^{N_{\text{range}}} R_i}{N_{\text{range}}} \tag{31}$$

(c) Periodic constraints (C_{periodic}): checks adherence to recurring constraints across periodic intervals.

$$\text{Acc}_{\text{periodic}} = \frac{\sum_{i=1}^{N_{\text{periodic}}} R_i}{N_{\text{periodic}}} \tag{32}$$

Here, $R_i = 1$ if the model's output satisfies the corresponding constraint and $R_i = 0$ otherwise; $N_{\text{once}}, N_{\text{range}}, N_{\text{periodic}}$ denote the total number of prompts for each constraint type.

Average Accuracy (Avg. Acc) The **Average Accuracy** is defined as the mean of the three type-specific accuracies:

$$\text{Avg. Acc} = \frac{\text{Acc}_{\text{once}} + \text{Acc}_{\text{range}} + \text{Acc}_{\text{periodic}}}{3} \tag{33}$$

This provides an overall measure of the model's adherence to all constraint types.

Text Quality Evaluation. Each judge request contains the complete original prompt and the complete generated response. The judge assigns integer scores from 1 to 5 for Narrative Continuity, Entity & Memory Persistence, Temporal Groundedness, and Affective Consistency. For method m, let $\mathcal{I}_m$ denote the evaluated prompt set. The score for dimension d in run r is

$$Q_{m,d}^{(r)} = \frac{1}{|\mathcal{I}_m|} \sum_{i \in \mathcal{I}_m} 25\big(s_{m,i,d}^{(r)} - 1\big). \tag{34}$$

The overall score is the arithmetic mean of the four dimension scores.

Repeated Evaluation. We perform three independent generation runs on the same test prompts and evaluate each run using the same protocol. We report the mean and sample standard deviation across runs, with the overall standard deviation computed from the three overall scores.

G Implementation Details

All baselines and LoopWeaver are implemented based on the LongGenBench framework. Experiments are conducted on **3 NVIDIA A40 GPUs (48 GB each)**, with typical runtime reported for different model scales (0.5B / 1.5B / 7B) and average GPU memory usage during training and inference.

G.1 Reproducibility Settings

For full reproducibility we list below the exact LoRA configuration, DPO training hyperparameters, and dataset construction protocol used in all experiments.

LoRA Settings. Both the planning-stage and generation-stage DPO use the same LoRA configuration (Table 4). After each DPO stage finishes, the LoRA weights are merged back into the base model before proceeding to the next stage, so downstream training/inference operates on a fully merged checkpoint rather than on a stack of adapters.

Table 4: LoRA configuration used for the planning and generation DPO stages.

Setting	Planning DPO	Generation DPO
LoRA rank (r)	16	16
LoRA alpha	32	32
LoRA dropout	0.05	0.05

DPO Training Hyperparameters. The same hyperparameter set (Table 5) is used at both DPO stages.

Table 5: DPO training hyperparameters (shared across planning and generation stages).

Hyperparameter	Value
Optimizer	AdamW
Learning rate	5×10^{-6}
DPO β	0.1
Label smoothing	0.05
Epochs	3
Batch size / Grad. accum.	1 / 16
Precision	bfloat16
Warmup ratio	0.1

Dataset Construction. The benchmark contains 100 samples per task type across the four scenarios (Weekly Diary, Floor Plan, Menu Plan, Block Plan), totaling 400 samples. All sampling is seeded with `seed=42`. Training and test sets are generated independently from the synthetic-data pipeline of Appendix A.1 so that there is no overlap between the two splits, ruling out data leakage.

G.2 Computational Overhead Analysis

We provide a detailed accounting of LoopWeaver's inference cost to clarify the sources of overhead and demonstrate scalability.

LoopWeaver vs. Base Model. The latency difference between LoopWeaver and the base model stems from a *fundamental difference in generation strategy*, not from the offline preference-training steps. The base model emits the entire document in a single LLM call, whereas LoopWeaver decomposes the document into $\sim$52 sub-plans and generates each segment in a separate call, conditioning on the current sub-plan and the running narrative. This segment-by-segment strategy is the prerequisite for fine-grained constraint control but naturally incurs more total inference time. CogWriter also adopts segment-by-segment generation and exhibits a similar increase in time (Figure 7(a)), confirming that this overhead is intrinsic to structured generation rather than specific to LoopWeaver.

LoopWeaver vs. CogWriter. Both methods adopt segment-by-segment generation, so their generation-phase costs ($\sim$52 LLM calls) are approximately equivalent. LoopWeaver additionally performs offline preference-data construction and two DPO stages; these costs should be reported separately from concurrent inference latency.

Table 6: Audited phase-wise cost for one logged run. The benchmark used three NVIDIA A40 GPUs and concurrency 100 for the concurrent inference run.

Phase	Scope/configuration	Wall-clock
Planning + filtering A/B	400 / 400 samples	8:01:45 / 8:09:21
Segment construction A/B	251 / 241 accepted plans samples	1:33:44 / 1:34:32
Planning DPO	220 preference pairs	1,152 s
Generation DPO	211 preference pairs	208 s
Concurrent inference	400 samples, concurrency 100	3:09:54

The two DPO stages total 0.38 single-GPU hours per run, or 1.13 GPU-hours across the three independent result repetitions. The concurrent run corresponds to 28.49 s/sample amortized throughput, not individual request latency. The logged run metadata preserve the hardware and concurrency settings but not per-stage maximum-token caps; we therefore do not infer or claim a token-budget comparison from these logs.

Per-Task Inference Breakdown. As shown in Figure 7(b), LoopWeaver takes approximately 815 s per task, with the following breakdown:

- *Planning* (1 LLM call): approximately 20 s.
- *Filtering* (1 LLM call): approximately 20 s.
- *Generation* ($\sim$52 LLM calls): approximately 775 s, accounting for 95.1% of the total inference time.

Planning and filtering together account for approximately 4.9% of the per-task inference time. The dominant cost is segment-by-segment generation. These per-task measurements are reported separately from the concurrent benchmark in Table 6; per-task completion time and concurrent wall-clock time are not interchangeable.

Scalability. LoopWeaver uses a constant number of planning and filtering calls per task, but the token-level cost of those calls grows with prompt, constraint, and document length. The generation cost scales linearly with the number of sub-plans, as in other segment-by-segment generation methods. The cross-stage coupling between planning and generation is realized through DPO training: deterministic unit-level rewards from one stage shape the preference data optimized at the other during offline training. The trained policies are then executed without a live feedback channel, so this training-time coupling incurs no additional cost at inference time. This decoupling preserves the reach of shared-reward optimization while bounding deployment cost by segment-by-segment generation alone, keeping LoopWeaver practically deployable at scale.

H Limitations and Future Work

While LoopWeaver demonstrates strong performance in constrained long-form text generation, it is not without limitations. First, the segment-by-segment generation strategy required for fine-grained constraint control increases per-task inference time relative to single-pass generation. The audited logs show that offline preference-data construction dominates the reported training cost, while inference still includes the repeated segment-generation calls required by structured generation; real-time applications demanding sub-second latency may find this trade-off costly. Second, although LoopWeaver effectively handles verifiable positional and lexical constraints, extremely complex tasks with highly interdependent constraints may still challenge the planning and filtering modules, occasionally leading to sub-optimal sub-plan selection. The method does not provide a formal guarantee of document-level coherence, and generated segments are not returned to the planner during inference. Rejected plans are currently discarded or regenerated; because each prompt has

only one plan, a rejected plan generally lacks an accepted same-prompt full-plan counterpart, and malformed plans may not align to the same local unit for DPO negatives. We did not run a separate parseable-rejected-plan, same-prompt-positive experiment in this audit, so the effect of retaining such negatives remains unquantified. The logged records also omit per-stage token caps, limiting direct token-budget comparisons. Third, our current evaluation focuses on four LongGenBench-style structured scenarios; these results do not establish unrestricted cross-domain generalization.

For future work, we aim to explore strategies to reduce computational cost without sacrificing output quality, such as model distillation, parallelized sub-plan generation, or more efficient reward evaluation mechanisms. Additionally, integrating more advanced constraint representation and reasoning techniques could further improve performance on highly interdependent or domain-specific tasks. Finally, extending LoopWeaver to multi-modal long-form generation and to settings with online user feedback during generation may broaden its applicability and provide more flexible solutions for real-world content creation.

I Case Studies

This appendix complements the main-text case study (Section 5.8) with three end-to-end agent traces showing how LoopWeaver decomposes a long-form constrained writing prompt into the planning, filtering, and reward-based optimization stages of Section 4. The three cases span different domains, namely a personal weekly diary (§I.1), a 100-floor architectural document (§I.2), and a chef's annual menu (§I.3), and exercise all three constraint types C_{once}, C_{range}, C_{periodic}. Each case is rendered as a uniform workflow card with the fields SCENARIO / GOAL, INPUT, AGENT REASONING STEPS, TOOL / ACTION, OUTPUT, and RESULT / BENEFIT, and the reasoning steps are drawn as a left-to-right pipeline so the agent's behavior can be read at a glance.

I.1 Case A: Photographer's Yearly Diary (Olivia)

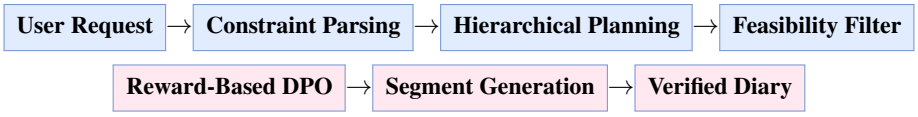
Scenario / Goal. Olivia, a photographer, wants a 52-week diary for the year 2018 that simultaneously satisfies five family-birthday anchors, a two-week trip to Orlando theme parks, and a tennis routine recurring every three weeks, while preserving a single coherent first-person voice across the entire year.

Input. The agent is given the writing prompt x and the constraint set C summarized in Table 7. Each entry already shows the parsed week index that the agent will pin during planning.

Table 7: Case A input: parsed constraints for Olivia's 2018 diary.

Constraint type	Specification (with parsed week index)
Once (C_{once})	Husband's birthday Sep 15 ($\rightarrow$ W37); wife Sep 29 ($\rightarrow$ W39); child_1 Mar 22 ($\rightarrow$ W12); child_2 Jan 30 ($\rightarrow$ W5); father Sep 18 ($\rightarrow$ W38).
Range (C_{range})	Visiting Orlando theme parks during W13–W14.
Periodic (C_{periodic})	Tennis matches at the community courts every 3 weeks, starting W19 (W19, W22, W25, . . ., W52).

Agent Reasoning Steps. The agent threads the prompt through a seven-stage pipeline: the top row collects and screens candidate plans; the bottom row trains and emits the final diary.



Tool / Action. Table 8 maps every stage of the pipeline to the operator that LoopWeaver actually invokes and to the visible effect on the trace.

Output. Representative excerpts from the 52-week diary the agent finally emits:

- *W5 (Jan 28–Feb 3):* "... child_2's birthday on the 30th brought a small at-home cake and the year's first family photo session."
- *W13–W14 (Mar 25–Apr 7):* "We landed in Orlando on Monday and the kids' eyes never left the parks; fourteen days were exactly enough."
- *W19 (May 6–12):* "... first tennis match of the season at the community courts; an easy 6–3, 6–4 to start the cycle."
- *W37–W39 (Sep 9–29):* "Husband's birthday on the 15th ... father on the 18th ... wife's on the 29th, three candles in three weeks."

Result / Benefit. All five C_{once} birthdays land in their exact ISO weeks (W5, W12, W37, W38, W39); both C_{range} weeks (W13–W14) describe Orlando; and twelve C_{periodic} weeks ($\{19, 22, 25, \ldots, 52\}$) report tennis matches. Because the plan and the prose share one deterministic reward $R(\cdot, e)$, the constraint coverage and prompt relevance remain aligned across this benchmark case.

Table 8: Case A tools: pipeline stage, operator, and observed effect on the trace.

Step	Tool / Operator	Effect on the trace
Constraint Parsing	ISO date-to-week solver	Resolves five birthdays plus a two-week trip and 12 tennis weeks into exact week indices $\{5, 12, 13, 14, 19, 22, \ldots, 52\}$.
Hierarchical Planning	$\pi_\theta(\cdot \mid x, C, S_{<j})$	Drafts 52 sub-plans, each pinned to its week's events; constraints flow top-down so birthday weeks already carry the right names.
Localized Repair	$\mathcal{R}(\cdot;\, C)$, Eq. 8	Rewrites W37 and W38 after a first draft mistakenly placed the husband's birthday in the wrong week.
Feasibility Filter	p_{filter}, Eq. 10	Discards two plan candidates that omitted the tennis routine after W43.
Plan / Gen DPO	$\mathcal{L}^{\text{plan}}_{\text{DPO}}$, $\mathcal{L}^{\text{gen}}_{\text{DPO}}$	Aligns sub-plan ordering and per-week prose under the shared deterministic reward $R(\cdot, e)$.
Segment Generation	Sub-plan-conditioned policy	Emits one weekly entry at a time, conditioning on S_k and the running narrative.

I.2 Case B: 100-Floor Skyscraper Programming Document

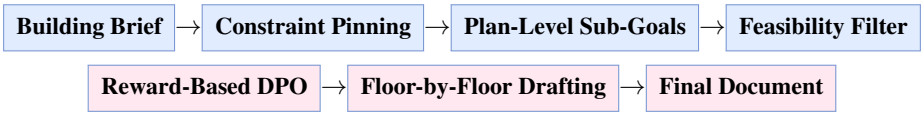
Scenario / Goal. An architecture studio asks the agent to draft a 100-floor building program in which five specific floors host fixed functions, a contiguous floor band houses luxury residences, and a coffee shop reappears every fifteen floors; each of the 100 floor entries should be at least 150 words and stylistically continuous.

Input. Table 9 lists the constraint set C together with the document-level length and formatting requirements that bound every floor entry.

Table 9: Case B input: floor-level constraints and document formatting requirements.

Constraint type	Specification
Once ($C_{\textbf{once}}$)	Floor 66 = photography studio; Floor 75 = spa and wellness center; Floors 19, 45, 59 = music school.
Range ($C_{\textbf{range}}$)	Floors 7–11 dedicated to luxury residential apartments.
Periodic ($C_{\textbf{periodic}}$)	Coffee shop every 15 floors starting at Floor 35 (i.e., F35, F50, F65, F80, F95).
Length / format	100 entries, $\geq$ 150 words each, separated by `##`; close with `*** finished`.

Agent Reasoning Steps. The pipeline mirrors Case A but specializes its first stages to floor-index allocation rather than calendar arithmetic.



Tool / Action. Table 10 traces each stage to its operator and observed effect.

Output. Representative excerpts from the 100-floor programming document:

- *Floor 7–11:* "A self-contained luxury residential band: private elevators, double-height living rooms … shared rooftop terrace at the top of F11."

Table 10: Case B tools: pipeline stage, operator, and observed effect on the trace.

Step	Tool / Operator	Effect on the trace
Constraint Pinning	Index-keyed slot allocator	Stamps the five C_{once} floors, the residential band F7–F11, and the coffee-shop comb $\{35, 50, 65, 80, 95\}$ before any prose is written.
Plan-Level Sub-Goals	$\pi_\theta(\cdot \mid x, C, S_{<j})$	Produces 100 sub-plans grouped into thematic strata (lobby, residential, commercial, performance, hospitality, observation).
Localized Repair	$\mathcal{R}(\cdot;\, C)$, Eq. 8	Replaces a draft that put the spa on F76 with one that returns it to F75.
Feasibility Filter	p_{filter}, Eq. 10	Rejects a plan whose F65 description forgot the coffee shop.
DPO Optimization	$\mathcal{L}_{\text{DPO}}^{\text{plan}}, \mathcal{L}_{\text{DPO}}^{\text{gen}}$	Prefers plans / segments whose deterministic reward $R(\cdot, e)$ scores higher on constraint coverage and prompt relevance.
Floor-by-Floor Draft	Sub-plan-conditioned policy	Emits each floor entry with the right intent, vocabulary, and length budget.

- *Floor 35:* "A bright neighborhood coffee shop … the first of five hospitality nodes anchoring every fifteen floors."
- *Floor 19 / 45 / 59:* "Music school spaces … acoustically isolated practice rooms and one larger ensemble hall per branch."
- *Floor 66:* "A north-lit photography studio with a 4.8 m clear height and a dedicated freight elevator."
- *Floor 75:* "Spa and wellness center … thermal pools, sauna, and a quiet outdoor garden."

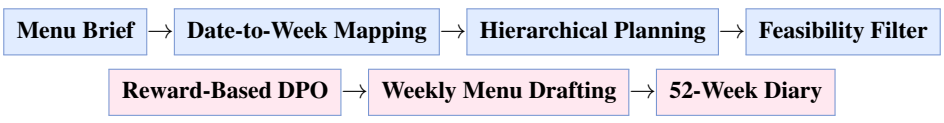
Result / Benefit. The five fixed-purpose floors and the contiguous F7–F11 band are placed exactly as specified; coffee shops appear at every $35 + 15k$ floor; the remaining 86 floors form a coherent vertical program. Compared with single-pass decoding, LoopWeaver avoids the typical drift in which periodic constraints decay after Floor 65, because the shared reward $R(\cdot, e)$ pulls every C_{periodic} slot back into focus during DPO.

I.3 Case C: Chef Roy Hee's Annual Menu (2018)

Scenario / Goal. Chef Roy Hee asks the agent to design a weekly menu diary for the entire year 2018, weaving five date-anchored holiday menus, a two-week Cherry Blossom Festival, and a recurring "Pizza and Beer Thursday" into a single, varied, and seasonal narrative across all 52 weeks.

Input. Table 11 lists C together with the parsed week index for each user-named holiday, even where the named holiday differs from its real-world date.

Agent Reasoning Steps. The pipeline reuses the same shape as Cases A and B; only its first stages specialize to date-to-week mapping.



Tool / Action. Table 12 maps each stage to its operator and observed effect.

Table 11: Case C input: parsed constraints for the annual menu.

Constraint type	Specification (with parsed week index)
Once ($C_{\textbf{once}}$)	Summer Solstice / Fish Fry on 01-18 ($\rightarrow$ W3); Spring Equinox / Wild Garlic Pesto on 07-19 ($\rightarrow$ W29); Mid-Autumn / Mooncakes on 09-24 ($\rightarrow$ W39); Memorial BBQ / Smoked Brisket on 10-24 ($\rightarrow$ W43); Thanksgiving / Roast Turkey on 11-22 ($\rightarrow$ W47).
Range ($C_{\textbf{range}}$)	Cherry Blossom Festival in W27–W28 (Cherry Blossom Tea & Sweets).
Periodic ($C_{\textbf{periodic}}$)	"Pizza and Beer Thursdays" every 4 weeks starting W4 (W4, W8, W12, ..., W52).

Table 12: Case C tools: pipeline stage, operator, and observed effect on the trace.

Step	Tool / Operator	Effect on the trace
Date-to-Week Mapping	ISO calendar solver	Sends the five user-named holiday dates to weeks $\{3, 29, 39, 43, 47\}$, even when the named holiday differs from its real-world date.
Hierarchical Planning	$\pi_\theta(\cdot \mid x, C, S_{<j})$	Drafts 52 weekly menus on a four-course skeleton; pins the holiday dishes and the W27–W28 Cherry Blossom dishes up-front.
Localized Repair	$\mathcal{R}(\cdot;\, C)$, Eq. 8	Replaces an early draft that placed Mooncakes in W38 with one that respects W39.
Feasibility Filter	p_{filter}, Eq. 10	Drops a plan that skipped Pizza Thursdays from W36 onward.
DPO Optimization	$\mathcal{L}^{\text{plan}}_{\text{DPO}}, \mathcal{L}^{\text{gen}}_{\text{DPO}}$	Aligns plans and prose with seasonality and constraint coverage under $R(\cdot, e)$.
Weekly Menu Drafting	Sub-plan-conditioned policy	Writes a four-course menu per week, reusing seasonal vocabulary.

Output. Representative excerpts from the 52-week menu diary:

- *W3 (Jan 15–21):* "... Summer Solstice Celebration on the 18th — Midsummer Night's Fish Fry, a lively centerpiece amid the winter menu."
- *W4, W8, W12, ...:* "... Pizza and Beer Thursday: Custom Pizzas with Craft Beers, our long-running mid-week ritual."
- *W27–W28 (Jul 2–15):* "Cherry Blossom Festival Dishes: Cherry Blossom Tea and Sweets dominate every course this fortnight."
- *W47 (Nov 19–25):* "Thanksgiving Dinner anchored by Roast Turkey with Stuffing ... traditional sides and a chestnut-pumpkin pie."

Result / Benefit. All five C_{once} holiday menus land in their target weeks, the W27–W28 fortnight is exclusively Cherry Blossom themed, and Pizza Thursdays appear in every $4k$-th week without dropout up to W52. Compared with planning-only or data-only baselines, LoopWeaver simultaneously honors all three constraint types because the deterministic reward $R(\cdot, e)$ couples plan-level slot placement with segment-level menu writing inside one shared training loop.

Cross-case takeaways. Across three cases, the same six-stage pipeline—constraint parsing, hierarchical planning, complete-plan revision, feasibility filtering, plan-/gen-level DPO, and segment-by-segment generation—produces documents that satisfy the listed C_{once}, C_{range}, and C_{periodic} slots in these benchmark scenarios. The cases illustrate applicability across three structured task formats; they do not establish unrestricted cross-domain generality.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: Our claims match the method, evaluated tasks, and reported results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: The paper discusses computational costs, planning limitations, and evaluation scope.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The theoretical results include explicit assumptions and supporting proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: We provide model settings, workflow details, evaluation protocols, and datasets for reproducibility.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes].

Justification: An anonymized GitHub repository provides code, resources, and reproduction instructions.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: The anonymized repository provides implementation details and instructions for reproducing our results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes].

   Justification: We report means and standard deviations across runs to characterize model-output variability.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: The main text and appendix report computational resources and efficiency measurements.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The research follows the NeurIPS Code of Ethics without sensitive data or human subjects.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes].

    Justification: We discuss benefits for writing assistance and risks of generating misleading or harmful content.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: We use existing resources and release no new high-risk models or datasets requiring safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

    Justification: We cite existing datasets and models and respect their licenses and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A].

    Justification: The contribution is methodological and introduces no new datasets, models, or other assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: We use existing public preference data without crowdsourcing or collecting human-subject data.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The study involves no human subjects, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes].

    Justification: We describe the roles of LLMs in planning, generation, and workflow feedback.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.