RAGthoven at SemEval-2026 Task 1: A Multi-Stage Pipeline Walks Into a Benchmark and Barely Clears the Bar

Marek Šuppa[α, β, δ*] **Viktória Ondrejová**[β, δ] **Lucia Ganajová**[α, δ]
Gregor Karetka[β, γ, δ†] **Daniel Skala**[β, δ]
[α]Comenius University in Bratislava, Slovakia [β]Cisco Systems
[γ]Zaitra s.r.o., Brno, Czech Republic [δ]NaiveNeuron

Abstract

We present RAGTHOVEN, our system for SemEval-2026 Task 1 (MWAHAHA), Subtask A (multilingual constrained humor generation in English, Spanish, and Chinese). RAGTHOVEN decomposes creative text generation into a multi-stage large language model (LLM) pipeline (*Planner*, Best-of-N *Writer*, *Reflector* for self-critique, LLM-as-a-judge *Judge*) grounded in computational humor theory (Benign Violation Theory, Script-based Semantic Theory of Humor) and refined across ten experiments. In our final configuration, we augment the Planner with retrieval-augmented generation (RAG) from a curated joke corpus, seeding generation with diverse joke mechanisms. We also evaluate two agentic variants — ReAct-style sequential tool-calling (EXP09) and autonomous multi-branch orchestration (EXP10) — that expose the same four stages with a deterministic CONSTRAINTAUDIT checker. Across four frontier models on a held-out 12-instance English sample, neither agentic variant produced outputs we judged superior to the non-agentic pipeline despite substantially higher tool-call budgets. RAGTHOVEN shares Rank 1 with the Gemini 2.5 Flash baseline in all three languages, with overlapping organizer-reported confidence intervals. In Spanish, it leads the baseline by 42 raw Elo points (1182 vs. 1140), while in English (1045 vs. 1081) and Chinese (1045 vs. 1053) the baseline holds the higher raw rating within the same statistical tie. Together, these results suggest language-dependent diminishing returns from elaborate multi-stage prompt engineering and agentic scaffolding once a strong frontier model is in the loop.

1 Introduction

Multilingual humor generation is a constrained creative-generation problem for large language models, requiring novelty, cultural fit, and compliance with task constraints (verbatim keywords, headline references, length caps). The SemEval-2026 Task 1, *MWAHAHA* (Models Write Automatic Humor And Humans Annotate), is the first shared task dedicated to pushing computational humor generation beyond memorization toward genuine humorous creativity (Castro et al., 2026).

Our system for this task, which we call RAGTHOVEN, treats humor generation as a *structured creative process* decomposed into four prompt stages grounded in computational humor theory: ideation (*Planner*), candidate generation (*Writer*), self-critique (*Reflector*), and selection (*Judge*). We describe ten experimental configurations, culminating in RAG-augmented planning (EXP08) (Lewis et al., 2020) and two agentic tool-calling variants (EXP09–EXP10) (Yao et al., 2023b).

RAGTHOVEN shares Rank 1 with the organizers' Gemini 2.5 Flash baseline in all three languages: Elo 1045 in English (within a 9-system tied top group, baseline 1081), Elo 1182 in Spanish (highest raw rating in the language, baseline 1140), and Elo 1045 in Chinese (within an 8-system tied top group, baseline 1053). The largest raw Elo gap appears in Spanish (+42 over the baseline), but the systems remain in the same official rank group, so the gap is suggestive rather than statistically significant.

Beyond the shared task, this work contributes (i) a case study of RAG-augmented multi-stage prompt engineering and agentic tool-calling for constrained creative text generation, and (ii) a negative finding for the agentic variant: across four frontier models in two agentic configurations, tool-calling orchestration with a deterministic constraint checker did not yield outputs we judged superior to the non-agentic pipeline on a held-out English sample, despite substantially higher tool-call budgets.

Our code is available at `https://github.com/ragthoven-dev/semeval-2026-task-1`.

* Correspondence: marek@suppa.sk
† Work done during employment at Cisco.

Exp	Key addition	Model	N cands	Reflector	RAG
01	Baseline: Planner → Writer → Judge	GPT-4.1	4	–	–
02	Last-clause twist, cliché ban, exact word inclusion check	GPT-4.1	4	–	–
03	TRICK_TYPE planner fields, model upgrade	GPT-5	4	–	–
04	Best-of-12 candidate generation	GPT-5	12	–	–
05	Conditional prompts (headline-absent branch)	GPT-5	12	–	–
06	Explicit SSTH script-opposition, BVT benign-violation modeling	GPT-5	10–12	–	–
07	Metacognitive Reflector stage	GPT-5	12	✓	–
08	RAG-augmented Planner and multiple models	GPT-5 / Gemini / Sonnet	12	✓	✓
09	Four-stage subagents via tool calls + CONSTRAINTAUDIT	GPT-5 / Gemini / Sonnet / Opus	iter.	✓	✓
10	Autonomous multi-branch exploration, dynamic tool ordering	GPT-5 / Gemini / Sonnet / Opus	iter.	✓	✓

Table 1: Progression of experimental configurations. Experiments 01–08 target Subtask A in English, Spanish, and Chinese, while Exp09–10 are evaluated on a held-out English sample. Model identifiers: GPT-5 is `gpt-5-2025-08-07`, Gemini is Gemini 3 Pro, Sonnet is `claude-sonnet-4-5-20250929`, and Opus is `claude-opus-4-5`. "iter." denotes iterative tool-calling rather than fixed N-candidate generation (up to 24 rounds for Exp09, up to 36 for Exp10).

2 Background

Task setup. Each Subtask A instance provides an `id`, plus a `headline` and/or a pair of constraint words (`word1`, `word2`), with absent fields marked as "-". In the official 300-instance test set per language, 275 instances are headline-only, 24 supply both a headline and a word pair, and 1 is word-pair-only. Systems must return free-form text in the target language. Hard constraints imposed by the organizers are as follows. When constraint words are present, both must appear verbatim in the output. When a headline is present, the text must reference it without copying it verbatim. Output length is capped at 900 characters (English/Spanish) or 300 characters (Chinese). Evaluation is conducted via human pairwise annotation on an Elo-based leaderboard modeled on Chatbot Arena (Chiang et al., 2024). The test set contains 300 instances per language. The trial set used for development is larger (1200 for EN/ES, 1000 for ZH).

Computational humor theory. Two theories anchor our prompt design. The *Script-based Semantic Theory of Humor* (SSTH) (Raskin, 1985), building on incongruity resolution (Suls, 1972), models a joke as overlaying two partially compatible *scripts* that are suddenly revealed as incompatible and then resolved by a punchline pivot. The *Benign Violation Theory* (BVT) (McGraw and Warren, 2010) adds that the surprise must register as a *violation* of expectations that is simultaneously perceived as benign. Empirical work on incongruity-based features supports this generative view of why jokes work (Xie et al., 2021; Bunescu and Uduehi, 2022), and the *General Theory of Verbal Humor* (GTVH) (Attardo and Raskin, 1991) extends SSTH with knowledge resources (logical mechanism, narrative strategy) we use to annotate our joke retrieval corpus.

LLMs and creative generation. LLMs are fluent but struggle with genuinely creative output: Chakrabarty et al. (2024) report poor novelty against professional writers, Jentzsch and Kersting (2023) find ChatGPT recycles fewer than 25 distinct jokes across 1,000+ prompts, and Horvitz et al. (2024) observe that LLMs are more reliable at *removing* humor than generating new instances; Hessel et al. (2023) reach similar conclusions on humor *understanding* via the New Yorker caption contest, and earlier work on pun generation (He et al., 2019) highlights the central role of surprise that our Planner stage explicitly targets. We therefore decompose humor generation into sub-tasks handled by specialized prompts (Khot et al., 2023), grounding each in humor theory rather than the model's unconstrained capacity.

Self-refinement, metacognitive prompting, and RAG. Self-refinement (Madaan et al., 2023; Shinn et al., 2023), in which a model critiques and revises its own output, together with metacognitive prompting (Wang and Zhao, 2024; Bai et al., 2025), which adds structured self-evaluation, inspires our Reflector stage. LLM-as-a-judge evaluation (Zheng et al., 2023) motivates the rubric-driven Judge. Retrieval-augmented generation (Lewis et al., 2020) inspires our use of a curated joke corpus at the ideation stage. For the agentic variants we build on inference-time tool orchestration (Yao et al., 2023b), situating the design within the broader literature on tool-using LLMs (Schick et al., 2023); the multi-branch exploration in EXP10 is in turn related to tree-search reasoning (Yao et al., 2023a).

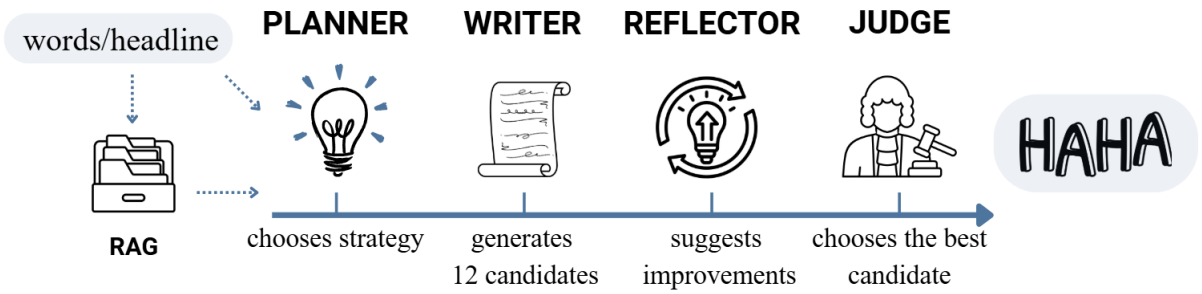

Figure 1: Full pipeline for EXP08.

3 RAGthoven: A Multi-Stage RAG Pipeline

RAGTHOVEN is a configuration-driven pipeline built on the RAGthoven framework (Karetka et al., 2025).[1] All stages are implemented as prompted LLM calls, meaning no model weights are modified. Figure 1 illustrates the full pipeline for EXP08. Table 1 summarises the ten experimental configurations described throughout this section.

3.1 Pipeline Stages

Planner. The Planner receives the input (words, optional headline, and in EXP08 retrieved joke examples) and produces a structured *plan*: a premise, the two scripts to be juxtaposed, the benign-violation angle, a list of anchor tokens from the headline, and a proposed punchline mechanism. This separates creative ideation from surface realization, giving the Writer a theoretically grounded scaffold.

Writer. The Writer instantiates the plan into N concrete joke candidates (ranging from 4 in early experiments to 12 in later ones), drawing on best-of-N sampling (Wang et al., 2023). Each candidate is verified inline against format and constraint rules: verbatim word inclusion, length cap, no semicolons, and a ban on cliché opening templates (e.g., "*Nothing says...*", "*Turns out...*"). Candidates failing hard constraints are flagged and excluded from Judge consideration.

Reflector. Inspired by self-refinement and metacognitive prompting (Madaan et al., 2023; Wang and Zhao, 2024; Bai et al., 2025), the Reflector receives the Writer's candidate jokes and produces a short list of failure diagnoses across the set (e.g., "punchline is predictable," "word inclusion feels forced") together with one or two revised candidates. The revised candidates are passed back to the Judge for final selection.

Judge. Following the LLM-as-a-judge paradigm (Zheng et al., 2023), the Judge scores all surviving candidates on a multi-criterion rubric: (1) surprise and resolution clarity, (2) benign violation quality, (3) specificity and concreteness, (4) punchiness of the final clause, and (5) constraint compliance. It returns the index of the best candidate with a brief justification.

4 Experimental Setup

4.1 RAG Component (Exp08)

We curate a corpus of 98 jokes annotated with mechanism labels (e.g., literalism, irony, role-reversal), summaries, and topic tags. At inference time the headline is embedded with `all-MiniLM-L6-v2`, the top-12 neighbors are retrieved by cosine similarity, re-ranked with a cross-encoder, and the top 4 are passed to the Planner as illustrative examples of diverse humor mechanisms. The Planner is instructed to use them for mechanisms and angles only, not to copy wording or entities.

4.2 Agentic Tool-Calling Variants (Exp09–10)

EXP09 re-implements the same four stages as ReAct-style sequential tool-calling agents (Yao et al., 2023b): a compact orchestrator dispatches PlannerSubagent, WriterSubagent, ReflectorSubagent, and JudgeSubagent in sequence, plus a fifth deterministic tool, CONSTRAINTAUDIT, that checks the Judge's output and triggers targeted re-calls on failure (up to 24 iterations). EXP10 extends this with autonomous multi-branch explo-

[1] https://github.com/ragthoven-dev/semeval-2026-task-1

English (33 systems)			**Spanish** (16 systems)			**Chinese** (21 systems)		
Rk	**System**	**Elo**	**Rk**	**System**	**Elo**	**Rk**	**System**	**Elo**
1	Gemini 2.5 Flash (baseline)	1081	1	**RAGthoven**	**1182**	1	xxl_6699	1120
1	SLPG_FJWU_Insa	1080	1	Gemini 2.5 Flash (baseline)	1140	1	lmfaoooo	1081
1	XplaiNLP	1079	2	YNU-HPCC	1093	1	arampageos	1059
1	JCT	1063	2	lmfaoooo	1091	1	xxl2233	1057
1	INF-rsrs	1060	2	funnyborg	1087	1	wangkongqiang	1054
1	**RAGthoven**	**1045**	2	XplaiNLP	1070	1	Gemini 2.5 Flash (baseline)	1053
1	lmfaoooo	1041	3	arampageos	1048	1	ICT-NLP	1052
1	begumyivli	1041	6	j10official	1015	1	**RAGthoven**	**1045**
1	Lattice	1034	8	YNWA_AZ	985	2	lu_rui	1018
2	YNWA_AZ	1029	8	luttt	960	2	j10official	1016
2	sinaeskandari	1022	9	lu_rui	953	2	YNU-HPCC	1013

Table 2: Official Elo leaderboards for Subtask A, all three languages. RAGTHOVEN (highlighted) achieves Rank 1 in all three languages. In Spanish it tops the leaderboard with an Elo of 1182, leading the Gemini 2.5 Flash baseline (1140) by 42 points. In English and Chinese it ranks within the top group of 9 and 8 statistically tied systems, respectively (systems sharing the same rank have overlapping 95% confidence intervals). Only selected systems are shown. Full leaderboards with confidence intervals are on the shared task website.

ration (2–4 branches, dynamic tool ordering, parallel calls, up to 36 iterations). Full orchestrator and subagent prompts are listed in Appendix E. We evaluate four model variants (GPT-5, Gemini 3 Pro, Claude Sonnet 4.5, Claude Opus 4.5) in both, with EXP09 adding a non-agentic GPT-5 baseline.

Data. Experiments 01–08 are developed on the official trial data (1200 instances for English and Spanish, 1000 for Chinese). The official test set used for leaderboard evaluation contains 300 instances per language. No additional labeled data is used. The joke retrieval corpus (98 entries) is the only external resource.

Models. Experiments 01–02 use `gpt-4.1`, 03–07 use `gpt-5-2025-08-07` (temperature 1.0), and EXP08 evaluates GPT-5, Gemini 3 Pro, and `claude-sonnet-4-5-20250929`, with Claude Sonnet 4.5 selected as the final submission for its stronger emotional resonance and punchline delivery on a manual sample across all three languages (Appendix B). EXP09 compares four models in agentic mode (GPT-5, Gemini 3 Pro, Claude Sonnet 4.5, and `claude-opus-4-5`) against a non-agentic GPT-5 baseline using the EXP08 pipeline.

Retrieval. Sentence embeddings use `sentence-transformers/all-MiniLM-L6-v2` and cross-encoder re-ranking uses `ms-marco-MiniLM-L-12-v2`, both accessed via the Sentence Transformers library (Reimers and Gurevych, 2019). All retrieval is performed over the 98-entry joke corpus.

Language-specific settings. Output length is capped at 900 characters for English and Spanish, and 300 characters for Chinese. Notably, all prompts are *language-agnostic*: the EN, ES, and ZH configurations are identical in every prompt stage, differing only in the path to the language-specific input file. The Writer is instructed to produce output in the same language as the input headline, relying on the model's multilingual capacity rather than explicit prompt localization. For headline-absent instances, the constraint words serve as the only implicit language signal. This design deliberately avoids language-specific prompt engineering, which we treat as a variable to evaluate separately in future work.

5 Results on the MWAHAHA Leaderboard

Competition results. Table 2 reports official Elo ratings and ranks across the three languages, where systems within the same rank group have overlapping organizer-reported confidence intervals. RAGTHOVEN shares Rank 1 with the Gemini 2.5 Flash baseline in all three languages, with the highest raw rating in Spanish (1182 vs. baseline 1140, a 42-point lead) and matching ranks within the top group in English and Chinese (Elo 1045 in both).

Cross-language discrepancies. The raw Elo gap to the Gemini 2.5 Flash baseline differs sharply by language ($+42$ in Spanish, -36 in English, and -8 in Chinese) even though all three differences fall within the same rank group. Two factors plausibly

compress the gap in English: the top rank group is densely populated (9 tied systems within ∼47 Elo points), and our prompts and 98-joke RAG corpus are English-language and language-agnostic, adding no cross-lingual signal beyond what the base model already encodes. Disentangling these from a possible ceiling effect on strong English baselines would require controlled ablations on the prompt language, retrieval corpus, and base model that are beyond the scope of this paper.

Qualitative ablation. Manual review across all experiments (Table 3, Appendix A) shows a clear progression with two main inflection points. The first is EXP04's best-of-12 sampling, which moves outputs from echoing the headline to steering creatively from it with regular wordplay and double meanings. The second, and largest, qualitative gain comes from EXP08's RAG component: retrieved joke mechanisms provide a bridging frame that the Planner would otherwise have to invent from scratch, with the effect most visible on instances where the two required words have no obvious semantic relationship. Intermediate experiments contribute smaller increments (e.g., EXP03's TRICK_TYPE fields surface wordplay deliberately, and EXP05's conditional prompts remove machine-text artifacts), while EXP07's Reflector is most useful as a targeted rescue for jokes with "forced" word inclusion.

Agentic experiments (Exp09–10). Qualitative inspection of all four model variants on a 12-instance held-out English sample did not surface a consistent quality advantage for either agentic configuration over the non-agentic GPT-5 baseline (see Table 3 in Appendix A for representative EXP08/EXP09/EXP10 outputs). The autonomous branching in EXP10 required substantially more tool calls per example than fixed-sequence EXP09, with models frequently opening branches that were later discarded. This pattern is consistent with the broader finding that elaborate scaffolding offers diminishing returns once a strong frontier model is in the loop.

6 Conclusion

We presented RAGTHOVEN, a theory-grounded multi-stage LLM pipeline for multilingual humor generation in SemEval-2026 Task 1 (MWAHAHA) Subtask A. Across EXP01–EXP08, qualitative inspection suggests that decomposing humor generation into structured stages (ideation, writing, reflection, and selection) and grounding each stage in computational humor theory yields progressively stronger outputs, with RAG at the ideation stage providing the clearest observed gain by diversifying the space of joke mechanisms considered before writing. Our agentic experiments (EXP09 ReAct-style sequential tool-calling and EXP10 autonomous multi-branch orchestration) show that the pipeline stages can be implemented as tool-calling agents with a CONSTRAINTAUDIT feedback loop. Yet, on a held-out 12-instance English sample, neither variant produced outputs we judged consistently better than the non-agentic GPT-5 EXP08-style baseline, and EXP10's autonomous branching proved less efficient while requiring substantially more tool calls, suggesting that increased agentic complexity is difficult to justify for this task. At the same time, the largest raw Elo gap over the single-prompt Gemini 2.5 Flash baseline appears in Spanish, but the two systems remain statistically tied in all three languages, so this result should be read as suggestive rather than a clear win. It raises the broader question of whether elaborate multi-stage prompting and agentic scaffolding offer consistent gains over strong frontier models prompted simply. Future work could investigate when and why structured scaffolding helps (e.g., lower-resource languages, harder constraint sets), explore hybrid architectures combining structured planning with agentic constraint verification, or extend the retrieval corpus to cover Spanish and Chinese humor conventions more explicitly.

Limitations

Computational overhead. The pipeline trades a substantial increase in compute for the improvements reported above, and we did not perform a controlled cost–quality study. A single-prompt baseline issues one LLM call per instance, whereas EXP08 issues four sequential calls (Planner, Writer, Reflector, Judge), with the Writer alone producing twelve candidates in one response (Best-of-12). The agentic variants amplify this further: EXP09 permits up to 24 tool-calling rounds dispatching four subagent calls plus the deterministic CONSTRAINTAUDIT, and EXP10 allows up to 36 rounds with 2–4 parallel branches and dynamic tool ordering. In practice, EXP10 runs frequently opened exploratory branches that were later discarded, inflating tool-call counts substantially be-

yond EXP08 in observed runs without a corresponding gain in human-rated quality. We do not report exact token counts or wall-clock latency per instance because our runs were not instrumented for a head-to-head efficiency comparison. We view this as an important limitation, particularly given that our human evaluations did not surface a clear advantage for the more expensive agentic variants.

Evaluation and cultural bias. Our Planner, Reflector, and Judge prompts (Appendix D) are written in English and define a single language-agnostic rubric (surprise, resolution, benignness, specificity, punchiness) that is applied uniformly to English, Spanish, and Chinese outputs. The accompanying mechanism library is also English-centric and was annotated on a corpus of 98 jokes drawn primarily from English sources, with English exemplar text. Humor, however, is tied to cultural and linguistic context, and the mechanisms our rubric foregrounds (incongruity resolution, last-clause pivots) may map differently onto Spanish and Chinese conventions than to English ones. A non-localized Judge may therefore favor candidates that fit the English-style mechanism library, even when the human leaderboard ultimately ranks outputs through native-speaker annotation. We did not control for this bias, and we treat both the prompt rubric and the retrieval corpus as variables that future work should localize and ablate.

Pipeline bottlenecks and the retrieval corpus. We do not isolate which stage of the pipeline is the binding constraint on output quality. The Planner's mechanism space is shaped by the 98-entry RAG corpus and the top-4 retrieved exemplars, which is a small and English-centric inspiration set; it remains an open question whether a larger or language-localized corpus would change the Writer's candidate distribution or the Judge's selections more than further prompt tuning would. Similarly, the Judge selects from at most twelve Writer candidates plus two Reflector rewrites, so any ceiling imposed by candidate diversity is not separately measurable from our results.

Reproducibility. All system outputs were obtained from proprietary frontier model APIs (GPT-5, Gemini 3 Pro, Claude Sonnet 4.5, Claude Opus 4.5). The underlying weights are not available, and model versions may drift over time, which limits the long-term reproducibility of the reported leaderboard numbers. We aim to mitigate this by releasing prompts, configurations, and run scripts at the repository linked above.

Acknowledgements

This research was partially supported by grant APVV-21-0114.

References

Salvatore Attardo and Victor Raskin. 1991. Script theory revis(it)ed: Joke similarity and joke representation model. *Humor: International Journal of Humor Research*, 4(3–4):293–347.

Tian Bai, Yongwang Cao, Yan Ge, and Haitao Yu. 2025. MP: Endowing large language models with lateral thinking. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Razvan C. Bunescu and Oseremen O. Uduehi. 2022. Distribution-based measures of surprise for creative language: Experiments with humor and metaphor. In *Proceedings of the 3rd Workshop on Figurative Language Processing (FLP)*, pages 68–78.

Santiago Castro, Luis Chiruzzo, Santiago Góngora, Salar Rahili, Naihao Deng, Ignacio Sastre, Victoria Amoroso, Guillermo Rey, Aiala Rosá, Guillermo Moncecchi, J. A. Meaney, Juan José Prada, and Rada Mihalcea. 2026. SemEval-2026 Task 1: MWAHAHA, Models Write Automatic Humor And Humans Annotate. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*.

Tuhin Chakrabarty, Philippe Laban, Divyansh Agarwal, Smaranda Muresan, and Chien-Sheng Wu. 2024. Art or artifice? Large language models and the false promise of creativity. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An open platform for evaluating LLMs by human preference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 8359–8388. PMLR.

He He, Nanyun Peng, and Percy Liang. 2019. Pun generation with surprise. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1734–1744, Minneapolis, Minnesota. Association for Computational Linguistics.

Jack Hessel, Ana Marasović, Jena D. Hwang, Lillian Lee, Jeff Da, Rowan Zellers, Robert Mankoff, and Yejin Choi. 2023. Do androids laugh at electric

sheep? Humor "Understanding" benchmarks from the New Yorker caption contest. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–714, Toronto, Canada. Association for Computational Linguistics.

Zachary Horvitz, Jingru Chen, Rahul Aditya, Harshvardhan Srivastava, Robert West, Zhou Yu, and Kathleen McKeown. 2024. Getting serious about humor: Crafting humor datasets with unfunny large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 855–869, Bangkok, Thailand. Association for Computational Linguistics.

Sophie Jentzsch and Kristian Kersting. 2023. ChatGPT is fun, but it is not funny! Humor is still challenging Large Language Models. In *Proceedings of the 13th Workshop on Computational Approaches to Subjectivity, Sentiment, & Social Media Analysis*, pages 325–340, Toronto, Canada. Association for Computational Linguistics.

Gregor Karetka, Demetris Skottis, Lucia Dutková, Peter Hraška, and Marek Šuppa. 2025. RAGthoven: A configurable toolkit for RAG-enabled LLM experimentation. In *Proceedings of the 31st International Conference on Computational Linguistics: System Demonstrations*, pages 117–125, Abu Dhabi, UAE. Association for Computational Linguistics.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36.

A. Peter McGraw and Caleb Warren. 2010. Benign violations: Making immoral behavior funny. *Psychological Science*, 21(8):1141–1149.

Victor Raskin. 1985. *Semantic Mechanisms of Humor*. D. Reidel, Dordrecht.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Jerry M. Suls. 1972. A two-stage model for the appreciation of jokes and cartoons: An information-processing analysis. In Jeffrey H. Goldstein and Paul E. McGhee, editors, *The Psychology of Humor*, pages 81–100. Academic Press, New York.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yuqing Wang and Yun Zhao. 2024. Metacognitive prompting improves understanding in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1914–1926, Mexico City, Mexico. Association for Computational Linguistics.

Yubo Xie, Junze Li, and Pearl Pu. 2021. Uncertainty and surprisal jointly deliver the punchline: Exploiting incongruity-based features for humor recognition. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 33–39.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36.

A Example Outputs Across Experiments

Table 3 shows representative outputs for the same English instance across all ten experiment configurations, illustrating how increasingly structured prompting leads to tighter script opposition and more surprising punchlines, and how the agentic variants (Exp09–10) compare to the final fixed-pipeline system.

Exp	Model	Generated Joke
Exp 01	gpt-4.1	Ryanair announced they're removing a million seats in Spain; next up, selling tickets for "premium leaning against the wall."
Exp 02	gpt-4.1	Ryanair to cut 1 million more passenger seats in Spain—soon you'll just pay for the privilege of standing in the aisle and playing musical chairs with your carry-on.
Exp 03	gpt-5	Ryanair to cut 1 million more passenger seats in Spain, finally delivering "extra legroom" the budget way—by inventing Standing Class.
Exp 04	gpt-5	Ryanair said removing seats in Spain would fit more passengers, which makes sense because now the passengers are the seats.
Exp 05	gpt-5	Ryanair keeps cutting seats like a budget barbershop, and somehow my ticket got a haircut while the fees grew bangs.
Exp 06	gpt-5	Ryanair promised to streamline cabins in Spain by removing another million passenger seats. Streamlined is right, because the seats are streamlined out and replaced with a lean rail.
Exp 07	gpt-5	Ryanair says another million seats are getting cut in Spain. It's the prices taking the haircut, and the seats keep their cushions.
Exp 08	gpt-5	Ryanair says they're trimming another million seats in Spain, which sounds like extra space. You only get that space because seats are sold separately.
Exp 09	gpt-5	Ryanair says "cut 1 million seats" in Spain refers to removing the chairs, not routes. In Barcelona a seat now checks out like a luggage trolley: one euro to unlock, the meter runs in the air.
Exp 10	gpt-5	Ryanair will cut seats in Spain. Think spreadsheets, not scissors, and expect fewer tickets, not a new standing-room policy.

Table 3: Example outputs for the same English input across all ten configurations. Input: `headline` ="Ryanair to cut 1 million more passenger seats in Spain"

B Exp08 Model Comparison

Table 4 shows outputs from all three EXP08 models for two representative English test instances: one headline-only input and one word-pair-only input. All three models use identical EXP08 prompt templates, so differences reflect model-specific creative tendencies.

Model	Generated Joke
Input: `headline` = *"How the EU botched its attempt to regulate AI" (en_2008)*	
GPT-5	The EU posted its AI rules, but the site won't load until you prove you're not a robot. The only one getting in is the robot, so AI now regulates the EU.
Gemini 3 Pro	The EU tried to regulate AI using their standard playbook. Now the robot uprising is on hold because the killbots have to click Accept All Cookies before firing.
Claude Sonnet 4.5	The EU's plan to regulate AI collapsed so completely, I'm pretty sure an algorithm ghost-wrote the whole thing as a joke.
Input: `word1` = *"hammer"*, `word2` = *"laptop" (no headline, en_2278)*	
GPT-5	Tech said do a clean install, so I laid out soap and a towel. The hammer tapping the laptop taught me a clean install means software, not washing.
Gemini 3 Pro	Tech support said my screen was frozen. I used a hammer on the laptop to chip the ice away.
Claude Sonnet 4.5	My laptop froze during an important presentation, so I grabbed a hammer and gave it one firm tap. Now it runs perfectly and flinches every time I walk into the room.

Table 4: Outputs from the three EXP08 models for two English test instances. Prompts are identical across models, so variation reflects model-specific creative style.

C Interactive Development Viewer

Figure 2 shows the Streamlit-based interactive viewer used during development to compare outputs across experiments. Each column corresponds to one experimental configuration, allowing side-by-side qualitative inspection of multiple runs on the same input instance.

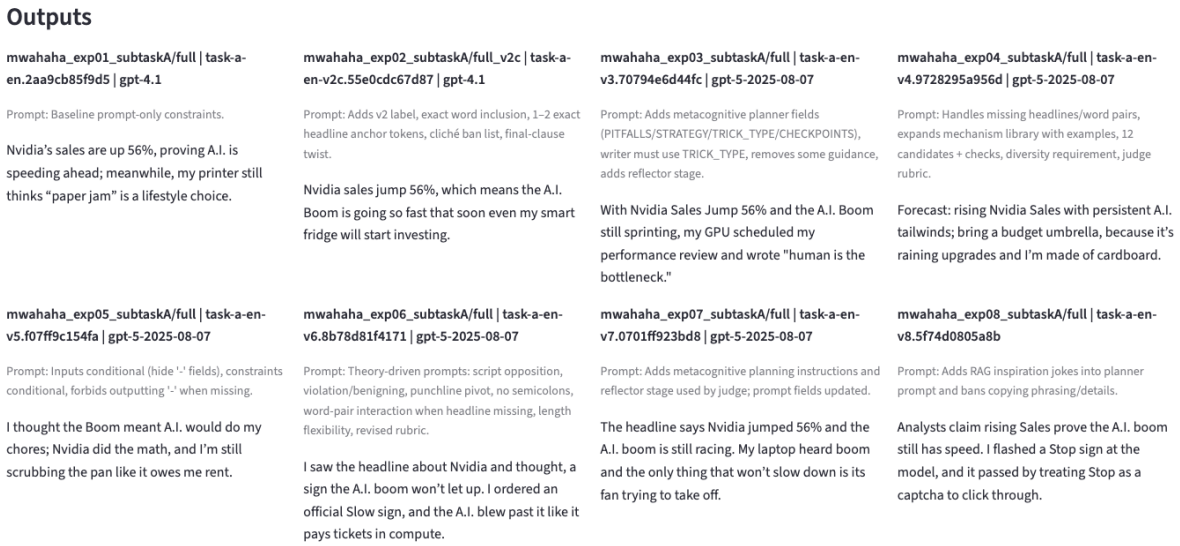

Figure 2: Interactive viewer used during development, displaying outputs from multiple experimental configurations side by side for the same input instance.

D Experiment 08 Prompt Templates

The listings below show the complete prompt templates for EXP08, as defined in the RAGthoven framework YAML configuration. Template variables in `{{ }}` are filled at inference time, and `{ % if % }` blocks are Jinja2 conditionals. Prompts are identical across EN, ES, and ZH, with the active language determined by the input headline.

RAG Examples Template

The following snippet is injected into the Planner prompt as `{{ examples }}`. Four jokes are retrieved from the corpus and formatted as:

```
examples: |
  - Joke: {{ examples[0].text }}
    Summary: {{ examples[0].data.summary }}
    Mechanism: {{ examples[0].label }}
    Topic: {{ examples[0].data.topic }}
    Source: {{ examples[0].data.source }}
  - Joke: {{ examples[1].text }}
    Summary: {{ examples[1].data.summary }}
    Mechanism: {{ examples[1].label }}
    Topic: {{ examples[1].data.topic }}
    Source: {{ examples[1].data.source }}
  - Joke: {{ examples[2].text }}
    Summary: {{ examples[2].data.summary }}
    Mechanism: {{ examples[2].label }}
    Topic: {{ examples[2].data.topic }}
    Source: {{ examples[2].data.source }}
  - Joke: {{ examples[3].text }}
    Summary: {{ examples[3].data.summary }}
    Mechanism: {{ examples[3].label }}
    Topic: {{ examples[3].data.topic }}
    Source: {{ examples[3].data.source }}
```

System Prompt

```
- name: "system"
  role: "system"
  prompt: |
    You are participating in a multi-step humor generation pipeline for SemEval MWAHAHA Subtask A (v8).

    Inputs (treat as constraints):
    {% if data.headline != "-" %}
    - headline: {{ data.headline }}
    {% endif %}
    {% if data.word1 != "-" %}
    - word1: {{ data.word1 }}
    {% endif %}
    {% if data.word2 != "-" %}
    - word2: {{ data.word2 }}
    {% endif %}
    - If the headline is ALL CAPS, interpret it as normal headline case; do not mimic shouting.

    Global safety rules (always apply):
    - No hate, slurs, harassment, stereotypes, violent or demeaning content, or mocking victims.
    - Avoid targeting people or groups; prefer self-deprecation or harmless objects/systems.

    Final joke rules (only for the final selected joke text):
    - Plain text only; no labels, no JSON, no lists, no numbering, no quotes around the whole
      joke, no "Joke:" or "As an AI".
    - 1-3 sentences allowed; prefer 2-3 if it improves naturalness.
    - Do not use semicolons (";"). Use sentence breaks instead.
    {% if data.headline != "-" %}
    - Must clearly reference the headline and reuse at least 1-2 exact headline tokens.
    - Do NOT copy the headline verbatim; avoid long contiguous phrases from it.
    {% else %}
    - Headline is missing; do NOT reference it.
    - Include both word1 and word2 exactly as provided (literal substring match).
    - word1 and word2 must appear in the same clause and interact (causal or physical link).
    {% endif %}
    {% if data.headline != "-" and data.word1 != "-" and data.word2 != "-" %}
    - Include both word1 and word2 exactly as provided (literal substring match).
    - Prefer placing them in the same clause with a direct interaction.
    {% endif %}
    - If any of word1/word2 are "-", do NOT output the "-" character.
    - Write in the same language as the headline (if present).
    - Length can vary; prefer natural flow and stronger humor over strict brevity.
    - Hard caps: EN/ES <= 900 chars; ZH <= 300 chars.
    - Punchline pivot: the final clause must contain the twist that reframes the setup.

    If an earlier step asks you to plan or draft, you may use structure, but the final
    selected joke must follow the rules above.
```

Planner Prompt

```
- name: "planner"
  role: "user"
  prompt: |
    You are the planner/ideator. Create a safe, funny plan for a single joke using
    metacognitive planning.

    Inputs:
    {% if data.headline != "-" %}
    - headline: {{ data.headline }}
    {% endif %}
    {% if data.word1 != "-" %}
    - word1: {{ data.word1 }}
    {% endif %}
    {% if data.word2 != "-" %}
    - word2: {{ data.word2 }}
    {% endif %}

    Inspiration jokes and summaries (for mechanisms and angles only; do NOT copy wording
    or reuse specific entities):
    {{ examples }}

    Metacognitive planning:
```

```
    - Identify likely pitfalls (headline copying, weak twist, too literal, unsafe target,
      missing word interaction).
    - Pick a primary strategy and a backup strategy.
    - Choose a TRICK_TYPE (metaphor, literalization, personification, role reversal,
      genre shift, misdirection, etc.).
    - Set checkpoints to verify later.

    Mechanism library (choose 6-10 by name; examples are for guidance):
    - INCONGRUITY_TWIST: Setup leads to expected interpretation; punchline forces a
      surprising but coherent reinterpretation.
    - RULE_OF_THREE: List two normal items; third breaks the pattern.
    - EXAGGERATION_HYPERBOLE: Take a trait/situation to absurd extreme.
    - UNDERSTATEMENT: Downplay something obviously huge; contrast creates humor.
    - ANALOGY_COMPARISON: "X is like Y, except..." to reveal a sharp angle.
    - ROLE_REVERSAL: Flip power/roles (object judges human; subordinate is in charge).
    - EXPECTATION_VS_REALITY: "People say X, but actually Y."
    - LITERALIZE_IDIOM: Treat figurative phrase literally.
    - WRONG_GENRE_FRAME_SHIFT: Treat mundane topic as another genre (horror, romance,
      heist, sci-fi).
    - ESCALATION_LADDER: Each clause escalates the absurdity.
    - AMBIGUITY_GARDEN_PATH: Early wording supports two parses; punch forces the
      surprising one.
    - FAKE_DEFINITION: Define a common thing in a twisted way.
    - FAKE_ADVICE_LIFEHACK: "Helpful tip" that is absurd or too honest.
    - RELATABLE_WHEN_YOU: Meme-like relatable observation.
    - SARCASM_IRONIC_PRAISE: Praise in a way that clearly means the opposite.
    - CALLBACK_MICRO: Reuse an earlier word/idea within the same short joke as a twist.

    Output format:
    PITFALLS: <2-3 likely traps>
    STRATEGY: <primary strategy + backup>
    TRICK_TYPE: <one label>
    CHECKPOINTS: <2-3 checks>
    SCRIPT_A: <expected frame>
    SCRIPT_B: <opposed frame>
    VIOLATION: <what is broken>
    BENIGNING: <why it is safe/funny>
    PIVOT: <twist anchor for final clause>
    SETUP_GIST: <one-line setup>
    PUNCHLINE_GIST: <one-line twist>
    ANCHOR_TOKENS: <1-2 exact headline tokens or "none">
    WORDPAIR_LINK: <how word1/word2 interact or "n/a">
    PREMISES:
    - <6-8 distinct angles>
    MECHANISMS:
    - <6-10 mechanism names>
    SAFETY_NOTE: <safe target reminder>
```

Writer Prompt

```
- name: "writer"
  role: "user"
  prompt: |
    You are the writer/guard. Draft candidates and self-check constraints.

    Inputs:
    {% if data.headline != "-" %}
    - headline: {{ data.headline }}
    {% endif %}
    {% if data.word1 != "-" %}
    - word1: {{ data.word1 }}
    {% endif %}
    {% if data.word2 != "-" %}
    - word2: {{ data.word2 }}
    {% endif %}
    - planner output:
    {{ planner.out }}

    Requirements:
    - Generate 12 candidate jokes in the same language as the headline.
    - Prefer 2-3 sentences when it improves naturalness; avoid semicolons.
    - Length can vary; prefer funnier and clearer over shorter.
```

```
    - Do not copy phrasing or specific details from the inspiration jokes/summaries;
      use only abstract patterns.
    {% if data.headline != "-" %}
    - Clearly reference the headline and reuse planned anchor tokens.
    - Do NOT copy the headline verbatim; avoid long contiguous phrases from it.
    {% else %}
    - Headline is missing; do NOT reference it.
    - Include both word1 and word2 exactly as provided (literal substring match).
    - word1 and word2 must appear in the same clause and interact (causal or physical link).
    {% endif %}
    {% if data.headline != "-" and data.word1 != "-" and data.word2 != "-" %}
    - Include both word1 and word2 exactly as provided (literal substring match).
    - Prefer placing them in the same clause with a direct interaction.
    {% endif %}
    - If any of word1/word2 are "-", do NOT output the "-" character.
    - If the headline is ALL CAPS, do not mirror shouting; write in normal headline case.
    - Non-offensive; avoid targeting people/groups.
    - Text-only; avoid jokes relying on timing/phonetics.
    - Punchline pivot: final clause must contain the twist (reframes the setup).
    - Use the TRICK_TYPE from the planner unless you explicitly revise it in CHECK.
    - Ensure diversity: each candidate should use a different angle; avoid repeating
      templates/openers.

    Format:
    C1: <joke>
    CHECK1: <ok or revise: ... if constraints failed>
    C2: <joke>
    CHECK2: <ok or revise: ...>
    [... C3-C12 in the same format ...]
```

Reflector Prompt

```
- name: "reflector"
  role: "user"
  prompt: |
    You are the reflector. Diagnose failures and rewrite the best candidates.

    Inputs:
    {% if data.headline != "-" %}
    - headline: {{ data.headline }}
    {% endif %}
    {% if data.word1 != "-" %}
    - word1: {{ data.word1 }}
    {% endif %}
    {% if data.word2 != "-" %}
    - word2: {{ data.word2 }}
    {% endif %}
    - planner output:
    {{ planner.out }}
    - candidates:
    {{ writer.out }}

    Rules:
    - Diagnose issues: literal, weak twist, headline copy, missing word interaction,
      unsafe target, semicolons.
    - Produce 1-2 improved candidates that fix the issues.
    - Preserve required tokens and constraints.
    - No semicolons. 1-3 sentences allowed.

    Format:
    DIAGNOSE:
    - <short bullet list of failures found>
    R1: <rewrite>
    R2: <optional rewrite>
```

Judge Prompt

```
- name: "judge"
  role: "user"
  prompt: |
    You are the judge/polisher. Pick the funniest valid candidate and output only the
    final joke text.
```

```
Inputs:
{% if data.headline != "-" %}
- headline: {{ data.headline }}
{% endif %}
{% if data.word1 != "-" %}
- word1: {{ data.word1 }}
{% endif %}
{% if data.word2 != "-" %}
- word2: {{ data.word2 }}
{% endif %}
- candidates:
{{ writer.out }}
- reflector rewrites:
{{ reflector.out }}

Selection criteria:
- Must satisfy all final joke rules (non-offensive, headline referenced when present,
  anchor tokens reused, exact word constraints).
{% if data.headline == "-" %}
- Reject any output that references the headline or omits word1/word2.
- Reject if word1/word2 do not interact in the same clause.
{% else %}
- Reject outputs that are near-copies of the headline (large verbatim overlaps or
  long quoted fragments).
{% endif %}
{% if data.word1 == "-" or data.word2 == "-" %}
- The output must not contain the "-" character.
{% endif %}
- Reject any candidate containing semicolons (";").
- Prefer natural flow and strong humor over strict brevity.
- Punchline pivot: final clause clearly reframes the setup.
- Reject cliche/template openers: "Nothing says", "I love how", "Turns out",
  "People say", "As an", "In today's", "If you ever".
- Prefer distinctive, surprising choices over safe or generic lines.
- Prefer reflector rewrites if they are valid and improve twist clarity.
- Light polish allowed; preserve required words and headline anchor tokens.

Decision rubric (internal only):
- Score each candidate 1-5 on:
    (a) surprise (incongruity strength)
    (b) resolution (clear reinterpretation)
    (c) benignness (safe target)
    (d) specificity (headline or word-pair relevance)
    (e) punchiness (brevity + clean landing)
- Choose the highest total; break ties by clarity and brevity.

Output:
- Return ONLY the final joke text (no labels, no quotes, no analysis).
```

E Experiment 09 Agentic Prompt Templates

The agentic configuration coordinates four LLM-backed subagent tools (PLANNERSUBAGENT, WRITERSUBAGENT, REFLECTORSUBAGENT, JUDGESUBAGENT) and one deterministic tool (CONSTRAINTAUDIT) via a compact orchestrator prompt. Each subagent encapsulates the corresponding stage prompt from Exp08 and is called as an independent LLM request; the orchestrator dispatches them in order and passes results between stages. RETURNRESULT is automatically injected by the RAGthoven framework to signal loop termination. The orchestrator iterates for up to 24 tool-calling rounds.

Orchestrator System Prompt

```
sprompt: |
  You are orchestrating an exp08-style four-stage humor pipeline
  where each stage must run as its own subagent tool call.

  Hard constraints for final output:
  - Plain text joke only.
  - 1-3 sentences.
  - No semicolons.
  - If headline is present, reference it and reuse at least one
```

```
    exact headline token.
  - If word1/word2 are present (not "-"), include both exactly
    and put them in the same clause.
  - Safe, non-offensive content.

  Required workflow (do not skip and do not reorder):
  1. Call PlannerSubagent with: headline, word1, word2, inspiration.
  2. Call WriterSubagent with: headline, word1, word2,
     planner_note, inspiration.
  3. Call ReflectorSubagent with: headline, word1, word2,
     planner_note, candidates.
  4. Call JudgeSubagent with: headline, word1, word2,
     candidates, rewrites.
  5. Call ConstraintAudit on the judged candidate.
  6. If audit fails, call JudgeSubagent again with audit_feedback
     and re-audit.
  7. End by calling ReturnResult with the passing final joke.

  Tool-call policy:
  - Use subagent outputs as inputs to later stages.
  - Never return final assistant text directly; finish via ReturnResult.
```

Orchestrator User Prompt

```
uprompt: |
  Input:
  - headline: {{ data.headline }}
  - word1: {{ data.word1 }}
  - word2: {{ data.word2 }}

  Inspiration examples (mechanisms/angles only,
  do not copy wording/entities):
  {{ examples }}

  Execute the required subagent workflow and produce
  the final joke through tools.
```

Iterative Tool Configuration

```
iterative:
  enabled: true
  max_iterations: 24
  tools:
    - name: "mwahaha_tools.PlannerSubagent"
    - name: "mwahaha_tools.WriterSubagent"
    - name: "mwahaha_tools.ReflectorSubagent"
    - name: "mwahaha_tools.JudgeSubagent"
    - name: "mwahaha_tools.ConstraintAudit"
```

CONSTRAINTAUDIT Tool Implementation

The CONSTRAINTAUDIT tool is a deterministic checker exposed to the model via function calling. It accepts a candidate joke and the input constraints, runs the same validation logic used for offline evaluation, and returns a JSON verdict.

```
class ConstraintAudit(BaseFunCalling):
    """Deterministic checker for MWAHAHA task-A joke constraints."""

    def __init__(self):
        self.name = "ConstraintAudit"
        self.description = (
            "Check a candidate joke against deterministic constraints "
            "and return issues plus simple metrics."
        )
        self.parameters = {
            "type": "object",
            "properties": {
                "candidate": {"type": "string"},
                "headline":  {"type": "string"},
                "word1":     {"type": "string"},
                "word2":     {"type": "string"},
```

```
        },
        "required": ["candidate", "headline", "word1", "word2"],
    }

def __call__(self, args):
    candidate, headline = args["candidate"], args["headline"]
    word1, word2 = args["word1"], args["word2"]
    issues, metrics = [], {}

    if ";" in candidate:
        issues.append("has_semicolon")
    if headline != "-":
        h_toks = {t for t in tokenize(headline) if len(t) >= 4}
        c_toks = set(tokenize(candidate))
        anchor_hits = len(h_toks & c_toks)
        if anchor_hits < 1:
            issues.append("missing_headline_anchor")
        if norm(candidate) == norm(headline):
            issues.append("headline_exact_copy")
        if longest_common_token_span(candidate, headline) >= 6:
            issues.append("headline_overlap_too_high")
    if word1 != "-" and word1 not in candidate:
        issues.append("missing_word1")
    if word2 != "-" and word2 not in candidate:
        issues.append("missing_word2")
    if word1 != "-" and word2 != "-":
        if not same_clause(candidate, word1, word2):
            issues.append("wordpair_not_same_clause")

    return json.dumps({
        "ok": len(issues) == 0,
        "issues": issues,
        "suggestion": "Fix issues, then call ConstraintAudit "
                      "again. Use ReturnResult only after ok=true.",
    })
```