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

Anonymous ACL submission

Abstract

We present RAGTHOVEN, our system for SemEval-2026 Task 1 (MuWaHaHa), 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*, *Writer*, *Reflector*, *Judge*) grounded in computational humor theories (Benign Violation Theory, Script-based Semantic Theory of Humor) and iteratively refined through prompt engineering 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 additionally explore an agentic variant that exposes the same four pipeline stages as tool-calling agents orchestrated by a model loop with a CONSTRAINTAUDIT checker; while it achieves full constraint compliance, human pairwise evaluation did not reveal a significant quality advantage over the simpler non-agentic baseline. RAGTHOVEN achieves Rank 1 in all three languages, with the strongest result in Spanish (Elo 1182, 42 points above the Gemini 2.5 Flash baseline). However, while the system leads in raw Elo in Spanish, it shares Rank 1 with the baseline in all three languages due to overlapping confidence intervals; in English and Chinese the gap narrows further, suggesting that elaborate multi-stage prompt engineering may offer diminishing returns once a strong frontier model is in the loop.

1 Introduction

Humor is considered among the most human cognitive capacities, yet it remains one of the hardest targets for natural language generation. The SemEval-2026 Task 1, *MuWaHaHa* (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 participating in this task, which we call RAGTHOVEN, treats humor generation as a *structured creative process* decomposed into four stages: ideation (*Planner*), candidate generation (*Writer*), self-critique (*Reflector*), and selection (*Judge*). Each stage receives carefully engineered prompts grounded in computational humor theory. 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., 2023).

RAGTHOVEN achieves Rank 1 in all three languages: Elo 1045 in English (among 9 tied top systems out of 33), Elo 1182 in Spanish (Rank 1, tied with the organizers' Gemini 2.5 Flash baseline), and Elo 1045 in Chinese (among 8 tied top systems out of 21). The Spanish result is particularly strong, with a 42-point margin over the next system (the Gemini 2.5 Flash baseline at 1140).

Beyond the shared task, this work contributes a case study of RAG-augmented multi-stage prompt engineering and agentic tool-calling for constrained creative text generation, with empirical findings on when increased scaffolding yields diminishing returns over strong frontier models.

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

2 Background

Task setup. Each Subtask A instance provides an `id`, two constraint words (`word1`, `word2`), and optionally a `headline` (the field is "-" when absent). Systems must return free-form text in the target language. Hard constraints imposed by the organizers include: both words must appear verbatim in the output; the text must reference the headline 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

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3343–3356
July 3-4, 2026

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; Exp09–10 are evaluated on a held-out English sample. GPT-5 refers to `gpt-5-2025-08-07`; Gemini to Gemini 3 Pro; Sonnet to `claude-sonnet-4-5-20250929`; Opus to `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).

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) builds on the incongruity-resolution model (Suls, 1972): a joke works by overlaying two partially compatible *scripts* that are suddenly revealed as incompatible; the incongruity is then resolved by a punchline pivot. The *Benign Violation Theory* (BVT) (McGraw and Warren, 2010) frames humor as arising when something simultaneously constitutes a *violation* of expectations or norms *and* is perceived as benign. Together these theories give a generative account of why jokes work, which we encode directly as prompting instructions; empirical work on incongruity-based features further supports this approach (Xie et al., 2021; Bunescu and Uduehi, 2022). The *General Theory of Verbal Humor* (GTVH) (Attardo and Raskin, 1991) extends SSTH with six knowledge resources including logical mechanism and narrative strategy, providing finer-grained categories we use to annotate our joke retrieval corpus.

LLMs and creative generation. Despite impressive general language abilities, LLMs struggle with genuinely creative output. Chakrabarty et al. (2024) find that LLM-generated text, while fluent, scores poorly on novelty compared to professional writers. In humor specifically, Jentzsch and Kersting (2023) show that ChatGPT generates fewer than 25 distinct jokes across over 1,000 prompts, recycling a small set of templates, and Horvitz et al. (2024) find LLMs are more reliable at removing humor than generating new instances. These findings motivate our design: we decompose humor generation into sub-tasks handled by specialized prompts (Khot et al., 2023), grounding each in humor theory rather than relying on the model's unconstrained generative capacity.

Metacognitive prompting and RAG. Metacognitive prompting (Wang and Zhao, 2024; Bai et al., 2025), in which a model reflects on its own reasoning, inspires our Reflector stage. Retrieval-augmented generation (Lewis et al., 2020) inspires our use of a curated joke corpus at the ideation stage.

3 System Overview

RAGTHOVEN is a configuration-driven pipeline built on the RAGthoven framework (Karetka et al., 2025).[1] All stages are implemented as prompted LLM calls; 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

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

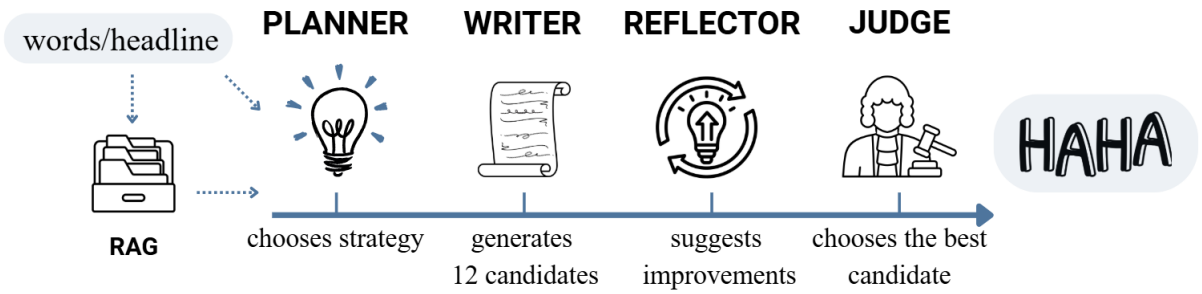

Figure 1: Full pipeline for EXP08.

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 (Exp07–Exp08). Inspired by metacognitive prompting (Wang and Zhao, 2024; Bai et al., 2025), the Reflector receives the top-ranked draft jokes and produces a diagnosis for each (e.g., "punchline is predictable," "word inclusion feels forced") together with a revised version. The revised candidates are passed back to the Judge for final selection.

Judge. 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.

3.2 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.

3.3 Agentic Pipelines (Exp09–10)

EXP09 re-implements the same four stages as tool-calling agents (Yao et al., 2023): a compact orchestrator dispatches PlannerSubagent, WriterSubagent, ReflectorSubagent, and JudgeSubagent in sequence, passing outputs between them as tool results. A fifth tool, CONSTRAINTAUDIT, deterministically checks the Judge's output and, on failure, triggers a targeted re-call of JudgeSubagent with structured feedback (up to 24 iterations total). We evaluate five variants: agentic GPT-5, Gemini 3 Pro, Claude Sonnet 4.5, Claude Opus 4.5, and a non-agentic GPT-5 baseline.

EXP10 extends this with autonomous multi-branch exploration: the orchestrator may open 2–4 independent branches, each running its own Planner and Writer calls, build a cross-branch shortlist, and converge through JudgeSubagent and CONSTRAINTAUDIT. Dynamic tool ordering and parallel tool calls are enabled, with up to 36 iterations. We evaluate four model variants (GPT-5, Gemini 3 Pro, Claude Sonnet 4.5, Claude Opus 4.5); no non-agentic baseline is included.

3.4 Experiment Progression

Each experiment in the 01–08 sequence is additive in its prompt constraints: we never remove a constraint found to help, only add or refine (though the underlying model changes at Exp03 and Exp08). Exp09–10 are orthogonal: they test the same pipeline via tool-calling agents rather than a fixed multi-turn prompt sequence (see Table 1).

4 Experimental Setup

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 in-

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.

stances 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` via the OpenAI API. Experiments 03–07 use `gpt-5-2025-08-07` (temperature 1.0). For EXP08 we evaluate three frontier models: `gpt-5-2025-08-07` (GPT-5.2), Gemini 3 Pro, and `claude-sonnet-4-5-20250929` (Claude Sonnet 4.5). To select the final submission model, we generated outputs for all three models on the test split and manually inspected a random sample across all three languages (see Table 4 in the appendix for representative examples). Claude Sonnet 4.5 was judged to be the strongest overall, consistently producing outputs with greater emotional resonance and more effective punchline delivery. We therefore submit Claude Sonnet 4.5 outputs as the final competition entry for all three languages. 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 multi-turn pipeline.

Retrieval. Sentence embeddings use `sentence-transformers/all-MiniLM-L6-v2`; cross-encoder re-ranking uses `ms-marco-MiniLM-L-12-v2` (Reimers and Gurevych, 2019), 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.

Development tooling. Iterative prompt refinement was supported by a Streamlit-based interactive viewer (Figure 2 in Appendix C) that rendered outputs from multiple experiments side by side, enabling rapid qualitative comparison across configurations during development.

5 Results

Competition results. Table 2 reports the official Elo ratings and ranks for all competing systems across the three languages. Systems within the same rank group are statistically indistinguishable at the 95% confidence level (i.e., their confidence intervals overlap).

RAGTHOVEN achieves Rank 1 in all three languages. The strongest result is in Spanish, where RAGTHOVEN achieves an Elo of 1182, the highest of any system. While it shares Rank 1 with the or-

ganizers' Gemini 2.5 Flash baseline (Elo 1140) due to overlapping confidence intervals, RAGTHOVEN leads by 42 Elo points. In English and Chinese, RAGTHOVEN places within the top rank group (9 and 8 tied systems, respectively), with Elo 1045 in both cases.

Qualitative ablation. A manual review of outputs across all experiments (see Table 3 in the Appendix A) reveals a clear progression in joke quality. EXP01 outputs are safe and predictable, frequently repeating the headline verbatim with no wordplay or surprise. Adding the last-clause twist and cliché ban in EXP02 introduces an element of unexpectedness, though jokes remain straightforward with no double meanings or wordplay. The introduction of TRICK_TYPE planning fields in EXP03 is where wordplay first emerges as a deliberate product of the structured ideation, and outputs begin to show creativity. Expanding to best-of-12 candidates in EXP04 is a turning point: jokes become noticeably more solid, no longer merely echoing the headline but instead steering creatively from it while staying on-topic, with word play and double meanings appearing regularly. Conditional prompting in EXP05 removes typographic artifacts of machine-generated text (dashes, semicolons), giving outputs a more natural feel. Theory-driven prompting in EXP06 yields longer, more structurally complex jokes that occasionally attempt sarcasm or self-deprecation; however, when the humor relies heavily on tone rather than linguistic surprise, the punchlines tend not to land as well. The Reflector stage in EXP07 did not produce a large qualitative leap on its own, though it was effective when the Writer produced jokes with "forced" word inclusion: the Reflector diagnosed this and rewired the sentence structure to embed the word more naturally. The largest qualitative gain comes from RAG in EXP08: outputs demonstrate strong command of wordplay, double meanings, and unexpected pivots. Retrieved joke mechanisms provide a bridging frame that the Planner would otherwise have to invent from scratch, and this is most visible on instances where the two required words have no obvious semantic relationship.

Agentic experiments (Exp09–10). Qualitatively, neither agentic architecture produces outputs that are clearly superior to the non-agentic baseline. The non-agentic GPT-5 baseline delivers a punchy two-liner (*"Relax, it's a new upgrade called Vertical Class where you pay to use your own legs"*), while the agentic GPT-5 in Exp09 constructs a longer literalisation frame (*"Ryanair says 'cut 1 million seats' refers to removing the chairs, not routes"*) and Exp10 yields a similarly adequate but unremarkable result (*"Ryanair will cut seats in Spain. Think spreadsheets, not scissors"*). The autonomous branching in Exp10 also required substantially more tool calls per example than the fixed-sequence Exp09, with models frequently opening branches that were later discarded, without a corresponding improvement in output quality. Human pairwise evaluation confirmed no significant quality advantage for either agentic variant over the simpler baseline, and the 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 (MuWaHaHa) Subtask A. Through ten experiments we showed 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 value by diversifying the space of joke mechanisms considered before writing. Our agentic experiments (EXP09–EXP10) show that the pipeline stages can be implemented as tool-calling agents with a CONSTRAINTAUDIT feedback loop; yet neither variant produced measurably better outputs than the simpler non-agentic baseline, and the autonomous branching in EXP10 proved less robust while requiring substantially more tool calls, suggesting that increased agentic complexity is difficult to justify for this task. At the same time, RAGTHOVEN only clearly outperforms the single-prompt Gemini 2.5 Flash baseline in Spanish; in English and Chinese the two are statistically tied, raising the question of whether elaborate multi-stage prompting offers 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.

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 N. Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. 2024. Chatbot Arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning*.

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 Suppa. 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.

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*, pages 3982–3992.

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

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; 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; 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.

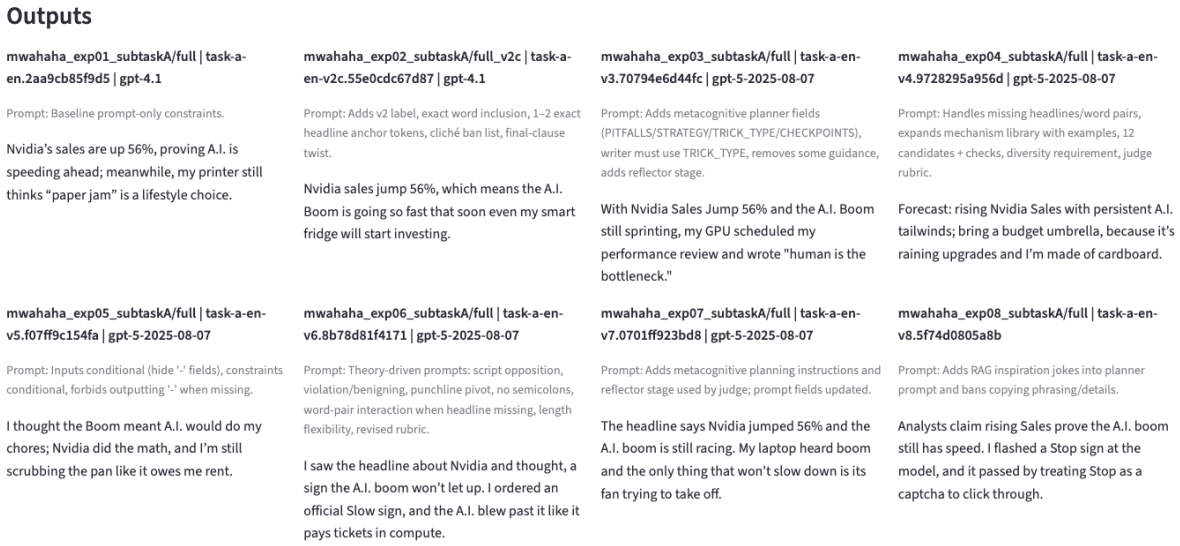

Figure 2: Interactive viewer used during development, displaying outputs from multiple experimental configurations side by side for the same input instance.

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; `{ % if % }` blocks are Jinja2 conditionals. Prompts are identical across EN, ES, and ZH; the active language is 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.

```python
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.",
})
```