# aba_team at SemEval-2026 Task 1: Plan2joke – Humor Policies for Type-Specific Two-Pass Humor Generation

**Andrii Dikhtiar**[1*] **Antonii Viter**[1*] **Bohdan Karaziia**[1*]
Daryna Dementieva[1,2] Alexander Fraser[1,2†]
[1]Technical University of Munich
[2]Munich Center for Machine Learning (MCML)
andrii.dikhtiar@tum.de antonii.viter@tum.de bohdan.karaziia@tum.de
daryna.dementieva@tum.de alexander.fraser@tum.de[* † ‡]

## Abstract

We present a planning-first humor generation system for SemEval-style tasks that separates joke generation into two stages: a *planner* that produces a structured humor plan and a *realizer* that renders a final joke conditioned on that plan. To reduce bias and template memorization common in large, noisy joke dumps, we apply unified filtering and construct type-consistent supervision aligned with an explicit Humor Policy for pun, satire, and irony. We train parameter-efficient adapters on two datasets that mirror inference (input → plan, and input plus plan → joke) and evaluate the resulting system with automatic and human-oriented analyses.

## 1 Introduction

Humor generation is difficult for large language models (LLMs) because a good joke must be coherent and surprising, while also staying on-topic and following explicit constraints. This challenge is central to SemEval-2026 Task 1 (MWAHAHA: Models Write Automatic Humor And Humans Annotate) (Castro et al., 2026), where systems must generate humorous outputs across diverse settings, including constrained text prompts (two-word inputs and headlines) and multimodal image-caption inputs. Therefore we propose *Plan2joke*, a planning-first humor generation pipeline with explicit routing and validation (We participate in the English track and all experiments use the English-language data provided by the shared task).

## 2 Background

A system must stay on-topic, respect explicit constraints, and still produce a coherent punchline. Zhong et al. (2024) highlight that even strong models fail reliably on structured humor tasks, motivating designs that emphasize controllability and robustness rather than single-shot prompting.

A practical issue is that creativity is hard to measure and even harder to scale with data. Zhong et al. (2024) note that collecting large amounts of genuinely innovative humor is difficult, which limits dataset diversity and makes it harder to generalize beyond patterns seen in training.

Humor is also subjective: what is funny depends on the audience and context, which makes evaluation noisy. Weller et al. (2020) emphasize that humor quality ultimately depends on human appreciation, motivating systems that can satisfy objective requirements even when funniness is hard to score automatically.

Finally, many jokes depend on background knowledge. Approaches that rely on shallow edits or rigid templates can miss the factual or definitional links behind a punchline. Zhang et al. (2020) argue for knowledge-aware humor generation, since external knowledge often supplies the hidden meaning that makes a joke work.

Motivated by these challenges, recent work suggests treating humor generation as a multi-step process rather than a single end-to-end prompt. Tikhonov and Shtykovskiy (2024) report that explicit multi-step reasoning improves one-liner humor quality in human evaluations, supporting pipelines that separate planning from final realization.

## 3 System overview

Our system, *Plan2joke*, is a planning-first pipeline for controlled humor generation. Each instance is normalized into a shared textual interface (including image inputs via scene-to-text extraction), grounded with two salient noun anchors, and enriched with compact Wikipedia microcards. Generation then follows a two-pass *Plan* → *Joke* pro-

[*]Equal contribution.
[†]Corresponding authors.
[‡]Code: https://github.com/AndriiD-dev/MWAHAHA_Competition_aba_team



*Proceedings of the The 20th International Workshop on Semantic Evaluation (2026)*, pages 2330–2346

cedure: a humor-type-specific planner produces a structured JavaScript Object Notation plan, and a realizer renders the final joke conditioned on that plan. We enforce objective validation (required words, structure) and apply deterministic retries; after repeated failures, an explicit Humor Policy routes to a fallback humor type (Figure 1).

## 4 Methodology

### 4.1 Input Normalization and Shared Textual Interface

Multi-stage humor pipelines often fail under strict constraints when control flow is brittle and recovery is limited. We therefore normalize all tasks into a shared textual interface so that the same grounding, enrichment, and planning steps apply across modalities.

For text-only tasks, the input is mapped into a common textual schema that preserves the original instance while enabling consistent downstream processing. For image-based tasks, we first obtain a concise scene-facts description (objects, actions, and setting). This early normalization ensures that all modalities enter the humor pipeline as text before any humor-specific reasoning begins.

For reproducibility, we keep anchor extraction deterministic under identical preprocessing conditions. When visual context is too sparse to yield stable anchors, we inject a minimal fallback token to maintain pipeline continuity.

### 4.2 Lexical Grounding via Noun Anchors

Long headlines and dense scene descriptions can cause topical drift, producing vague jokes that ignore the core subject. To enforce topicality, we extract two salient noun anchors and treat them as hard lexical constraints for all downstream steps.

The anchor extractor identifies nouns and proper nouns and filters low-signal tokens (digits, stopwords, and generic terms). It reconstructs compound noun phrases (e.g., "lottery ticket") and enforces diversity by rejecting near-duplicate variants. For multimodal inputs, we apply the same extraction logic to the scene-facts text, yielding a modality-agnostic grounding mechanism.

### 4.3 Context Enrichment via Wikipedia Microcards

Given isolated anchors, large language models can hallucinate relations or rely on shallow associations. To add lightweight factual grounding, we optionally attach compact Wikipedia "microcards" for each anchor.

For each anchor, we retrieve a short summary and compress it into a token-efficient record containing a definition-like sentence and a small set of key terms. The resulting microcards are injected into the planning prompt, providing stable reference attributes while keeping the context budget small.

### 4.4 Two-Pass Generation: Structure, Creation, and Parsing of the Comedic Plan

End-to-end humor generation can struggle to jointly maintain coherence, topicality, and lexical constraints. We therefore use a two-pass *Plan* $\rightarrow$ *Joke* procedure that separates deliberation from realization.

In Pass 1 (Plan Pass), the model receives the normalized input, the two required anchors, and optional grounding context, and outputs a structured JavaScript Object Notation plan. The plan encodes the intended scenario, the humor mechanism (pun, satire, or irony), and an anchor-placement strategy. We parse the plan into a valid dictionary and inject it into Pass 2 (Realization Pass), which generates the final joke conditioned on the plan as the controlling instruction.

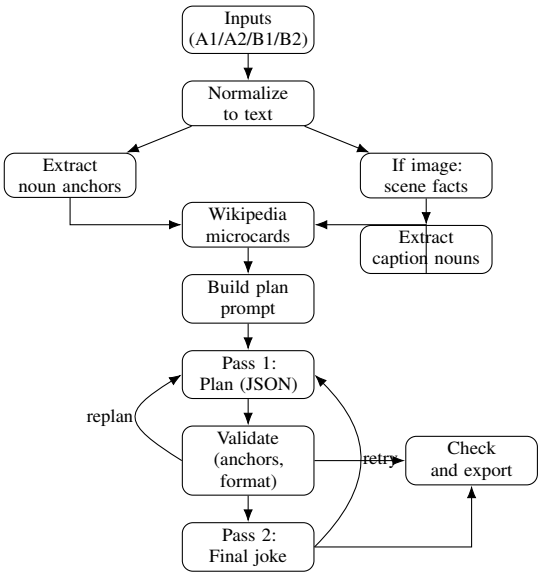


Figure 1: **Compact control flow of Plan2joke.** Inputs are normalized, grounded with noun anchors (and scene facts for images), enriched with Wikipedia microcards, then generated via two-pass Plan $\rightarrow$ Joke with validation-driven replan/retry.

### 4.5 Validation and Self-Healing Control Flow

Even with structured planning, generations can fail due to missed required words, malformed outputs, or topical drift. The final joke is evaluated for objective constraint adherence, including the presence of required anchors and basic structural validity. When the output fails these checks, the system invalidates the underlying plan and triggers a replan cycle, forcing the model to abandon the flawed blueprint and generate a new JSON strategy from scratch.

This self-healing behavior is controlled through `max_retries` and a deterministic sequence of `retry_steps`. Across retries, the pipeline injects progressively higher `temperature` and `top_p` values to encourage controlled exploration when the model becomes stuck in repetitive or unproductive generations. This thermodynamic scaling expands the semantic search space while preserving a deterministic retry schedule for reproducible benchmarking. By coupling strict validation with iterative regeneration, the system reduces critical failure rates without relying on manual intervention or brittle post-hoc edits.

### 4.6 Humor Policies as Strategy Routing and Fallback Logic

Initial iterations of the baseline model exhibited a lack of contextual alignment: without explicit strategy routing, the model often selected humor styles that were ill-suited to the input modality or semantic context, producing contrived or incoherent jokes. The pipeline implements a hierarchical Humor Policy: for each task two possible humor types are defined (main and secondary). Runner enforces the fallback to the secondary humor type when repeated validation failures occur.

For text-based inputs consisting of isolated word pairs, the system prioritizes puns and falls back to irony if validation fails after three attempts. For news headlines, the policy analyzes semantic context: public and corporate environments trigger satire (with a higher retry allowance), while personal or everyday scenarios default to irony. For multimodal image-based tasks, the policy searches for visible text markers (e.g., "sign," "reads") in the scene facts to attempt a pun-oriented punchline, if such markers are absent, or if the initial attempt fails validation, the policy immediately defaults to irony to avoid brittle visual wordplay.

This explicit routing mechanism tightly couples input modality and semantic cues to the two-pass generation prompts, and it provides a principled fallback path that prevents the system from repeatedly attempting an unsuitable humor strategy. A comprehensive diagram illustrating the humor typization policies can be found in (Appendix B).

## 5 Training

We trained the system in two stages to match the planning-first generation procedure used at inference time. Specifically, we trained two parameter-efficient fine-tuning adapters: a *planner* adapter that produces a structured plan, and a *realizer* adapter that turns a plan into a final joke.

**Data sources and filtering** Initial experiments with large, Reddit-style joke dumps (for example One Million Reddit Jokes (SocialGrep, 2024) and DadJokes (shuttie, 2024)) revealed substantial noise, near-duplication, and offensive content. When we directly used this raw mixture for supervised fine-tuning, generations became overly repetitive and skewed toward low-effort patterns. We therefore applied a unified prefiltering pipeline across all sources: deduplication, heuristic sanitation, and a binary joke-versus-not-joke classifier. We retain only items above a humor-confidence threshold of $0.8$.

To obtain cleaner and more type-consistent supervision aligned with our Humor Policy, we additionally used curated humor resources that provide labels or groupings for humor analysis (Humor-Research, 2023b,a). These resources required additional data engineering because some groups were incomplete or unevenly populated. We targeted $8$k–$10$k items per humor type (pun, satire, and irony) and used a judge-model approach to score, relabel, and selectively expand minority categories until they met this range. All augmented items were subjected to the same deduplication and filtering procedure to avoid inflating the dataset with near-duplicates.

Finally, to match the shared-task input formats, we generated aligned headline and image-caption-style variants for each retained joke using `DeepSeek-V3.2`. We then constructed two training datasets that mirror inference. First: input (two noun anchors, headline, or caption) $\rightarrow$ plan, where plans were produced by `DeepSeek-V3.2` under our schema after noun-anchor extraction and Wikipedia-based context enrichment. Second: input plus plan $\rightarrow$ joke, where the model is trained to

treat the plan as a binding control signal. Further details of the policy prompts and dataset construction are provided in the corresponding sections and (Appendix E).

**Planner adapter** The *planner* adapter was trained on the synthesized planning data described in the data preparation section. Each training example paired a task-specific user prompt with a planning system prompt that enforces a structured planning schema (Appendix E). The target output is the corresponding plan used by our pipeline at inference time.

**Realizer adapter** The *realizer* adapter was trained to generate the final joke conditioned on both the original input and the plan produced in the first stage. Concretely, the user prompt included the input plus the plan, while the system prompt specified the target humor topology. The target output is the original (or filtered) joke text associated with the item. This trains the realizer adapter to treat the plan as a binding control signal. We trained separate prompt variants for the three humor types used by our Humor Policy (pun, satire, and irony), for both the planner and the realizer (Appendix E).

**Fine-tuning setup and results** We used supervised fine-tuning for three epochs with `lora_r=64` on the `Qwen2.5-3B-Instruct` base model. We split the supervised fine-tuning data into $90\%$ training and $10\%$ evaluation, and report mean token accuracy on the held-out evaluation split: $88.6\%$ for the planner adapter and $84.6\%$ for the realizer adapter. Details are in Appendix C.

## 6 Evaluation and Results

We evaluate systems with two complementary setups: an encoder-based humor detector (reproducible but potentially misaligned with human judgments) and a judge-model rubric (semantically informed but dependent on the judge model).

**Experimental conditions** We compare four systems: the base `Qwen2.5-3B-Instruct` model, the base model with the planner adapter only, the base model with both planner and realizer adapters, and a larger decoder-only model (`DeepSeek-V3.2`) used as a strong prompted baseline. All systems are evaluated on the same input instances from the official evaluation split released by the shared-task organizers. This benchmark provides a fixed, comparable test set across tasks and modalities, and we do not use any of these instances (or derived variants) during supervised fine-tuning to avoid leakage. We report task-level aggregates for Task A conditions (two-word prompts and headlines) following the shared-task definition (Castro et al., 2026).

| System | Humor Coefficient (mean) | | |
|---|---|---|---|
| | A1 | A2 | Overall |
| Qwen2.5-3B: planner adapter | 46.4 | 49.3 | 44.2 |
| Qwen2.5-3B: both adapters | 40.9 | 45.3 | 39.7 |
| DeepSeek-V3.2 (prompted) | 54.1 | 63.3 | 53.0 |

| System | Reliability (%) | |
|---|---|---|
| | Fallback | Primary pass |
| Qwen2.5-3B: planner adapter | 25.0 | 92.5 |
| Qwen2.5-3B: both adapters | 18.5 | 93.0 |
| DeepSeek-V3.2 (prompted) | 16.5 | 96.9 |

Table 1: **Compact main results.** Mean Humor Coefficient for Task A1 (two words), Task A2 (headline), and overall, plus reliability indicators (fallback rate and primary constraint pass). Full per-task results are in Figure 5.

**Encoder-based humor detector** Our first evaluation uses an encoder-based humor detector that outputs two confidence scores: $p(humorous) \in [0, 1]$ and $p(non-humorous) \in [0, 1]$ (Baranov et al., 2023). We visualize the distribution of $p(humorous)$ across generated jokes using violin plots, which better reflect uncertainty mass than a single mean value. The full plots for the comparison between the base `Qwen2.5-3B-Instruct` model, the planner-only system, and the two-adapter system are provided in (Appendix F).

The detector strongly favors the two-adapter system on Task A inputs, often assigning near-saturated humorous confidence. However, this signal alone is not sufficient to claim improved funniness, because the detector may reward stylistic and structural similarity to humor seen during its own training.

**Judge-model rubric** Our second evaluation uses a judge model to score each joke along four interpretable criteria: *funniness*, *cleverness*, *clarity*, and *constraint adherence*, each in $[0, 100]$. We compute a single Humor Coefficient as:

$$\mathrm{HC} = \mathrm{round}(0.55{\cdot}F{+}0.25{\cdot}C{+}0.15{\cdot}L{+}0.05{\cdot}A),$$

where $F$ is funniness, $C$ is cleverness, $L$ is clarity, and $A$ is constraint adherence. We use the same evaluation inputs as in the detector experiment. The

rubric prompt, aggregation details, and the full results table are provided in (Appendix G).

In contrast to the detector-based experiment, the judge-model evaluation ranks the planner-only system above the two-adapter system on average for Task A, with approximate gap of 4.5 points in the aggregated Humor Coefficient. Qualitatively, we observe that the realizer adapter can over-regularize toward learned joke templates, improving "humor-likeness" but not necessarily perceived funniness. This discrepancy is consistent with the hypothesis that the detector is sensitive to familiar surface patterns, while the judge rubric is more sensitive to semantic fit and narrative coherence. Across all systems, the prompted `DeepSeek-V3.2` baseline achieves the highest average Humor Coefficient, exceeding the best `Qwen2.5-3B-Instruct` variant by approximately 8.8 points on the same inputs.

## 7 Findings/Conclusion

This work targets recurring challenges highlighted in prior humor generation research: unreliable constraint satisfaction, scarce high-quality creative data, and noisy evaluation signals (Zhong et al., 2024; Weller et al., 2020). Our results suggest that treating humor generation as a controlled, multi-step process improves reliability under hard constraints. In particular, explicit planning combined with automatic validation reduces off-topic outputs and constraint violations, while the Humor Policy routing improves topical alignment by selecting an appropriate joke topology for each input. At the same time, our experiments reveal a key limitation: improvements on an encoder-based humor detector do not necessarily translate into higher judge-model scores. We interpret this as a mismatch between 'humor-shaped" surface patterns and perceived funniness. In our setting, a judge-model rubric provides a more informative comparative signal across systems than a small detector model, although it remains dependent on the judge prompt and the judge model itself. Finally, because humor is audience-dependent, future work should incorporate human evaluation and calibration across demographics and content contexts, rather than relying on a single automatic signal.

## 8 Limitations

While our results indicate that a planning-first pipeline can improve control and reliability in constrained humor generation, several limitations remain. A practical limitation of our approach is scalability. The Humor Policy relies on curated humor categories and prompt templates, which requires manual design and maintenance when extending to new humor styles or domains. In contrast, an idealized reinforcement-learning setup with a robust reward model could, in principle, learn broader humor behaviors from feedback without explicitly enumerating humor topologies. In addition, our current implementation depends on several intermediate components, such as noun-anchor extraction, Wikipedia microcards, structured plan generation, and validation-driven retries. So overall quality remains sensitive to errors propagated from earlier stages. Our experiments also show that the two-pass setup does not improve all parts of the system uniformly: while it increases control and reliability under hard lexical constraints, the full planner-realizer configuration scores below the planner-only variant in the judge-model evaluation, suggesting that the realizer can over-regularize toward familiar joke forms. More broadly, the discrepancy between the humor detector and the judge-model rubric indicates that current automatic evaluation signals capture different aspects of output quality, which makes system comparison informative but not fully definitive.

## References


Alexander Baranov, Vladimir Kniazhevsky, and Pavel Braslavski. 2023. You told me that joke twice: A systematic investigation of transferability and robustness of humor detection models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13701–13715, Singapore. Association for Computational Linguistics.

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)*.

Humor-Research. 2023a. hri_tools. `https://github.com/Humor-Research/hri_tools`. GitHub repository. Accessed 2026-02-28.

Humor-Research. 2023b. Humor detection resources. `https://github.com/Humor-Research/Humor-detection`. GitHub repository. Accessed 2026-02-28.

Emil Joswin, Aditya Choudhary, and Raghav Garg. n.d. Deep-humor: Generation, analysis and classification of humor using transformers. Project report (PDF) in GitHub repository. Accessed 2026-02-27.

Sean Kim and Lydia B. Chilton. 2025. Ai humor generation: Cognitive, social and creative skills for effective humor. *Preprint*, arXiv:2502.07981.

shuttie. 2024. Dadjokes. https://huggingface.co/datasets/shuttie/dadjokes. Hugging Face dataset. Accessed 2026-02-28.

SocialGrep. 2024. One million reddit jokes. https://huggingface.co/datasets/SocialGrep/one-million-reddit-jokes. Hugging Face dataset. Accessed 2026-02-28.

Alexey Tikhonov and Pavel Shtykovskiy. 2024. Humor mechanics: Advancing humor generation with multistep reasoning. *Preprint*, arXiv:2405.07280.

Orion Weller, Nancy Fulda, and Kevin Seppi. 2020. Can humor prediction datasets be used for humor generation? humorous headline generation via style transfer. In *Proceedings of the Second Workshop on Figurative Language Processing*, pages 186–191, Online. Association for Computational Linguistics.

Hang Zhang, Dayiheng Liu, Jiancheng Lv, and Cheng Luo. 2020. Let's be humorous: Knowledge enhanced humor generation. *Preprint*, arXiv:2004.13317.

Shanshan Zhong, Zhongzhan Huang, Shanghua Gao, Wushao Wen, Liang Lin, Marinka Zitnik, and Pan Zhou. 2024. Let's think outside the box: Exploring leap-of-thought in large language models with creative humor generation.

# A Control Flow Graph

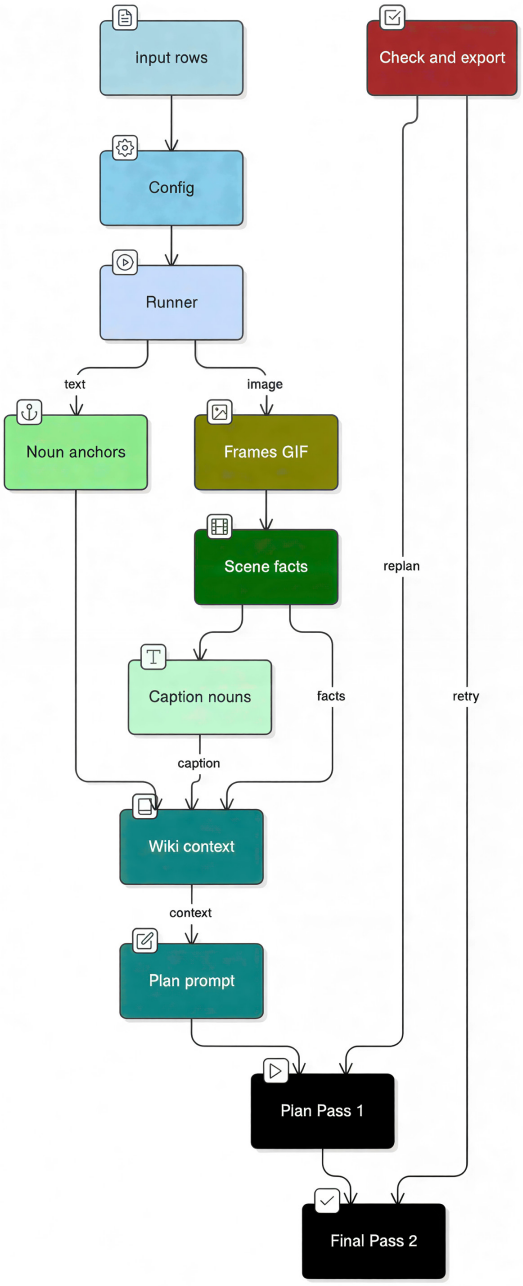


Figure 2: **Control flow graph of the execution pipeline.** This diagram illustrates the end-to-end execution flow of the inference pipeline, highlighting the modality-specific routing and context enrichment stages. It also demonstrates the autonomous self-healing mechanisms, specifically the recursive retry and replan feedback loops.

## B Humor Policies Diagram

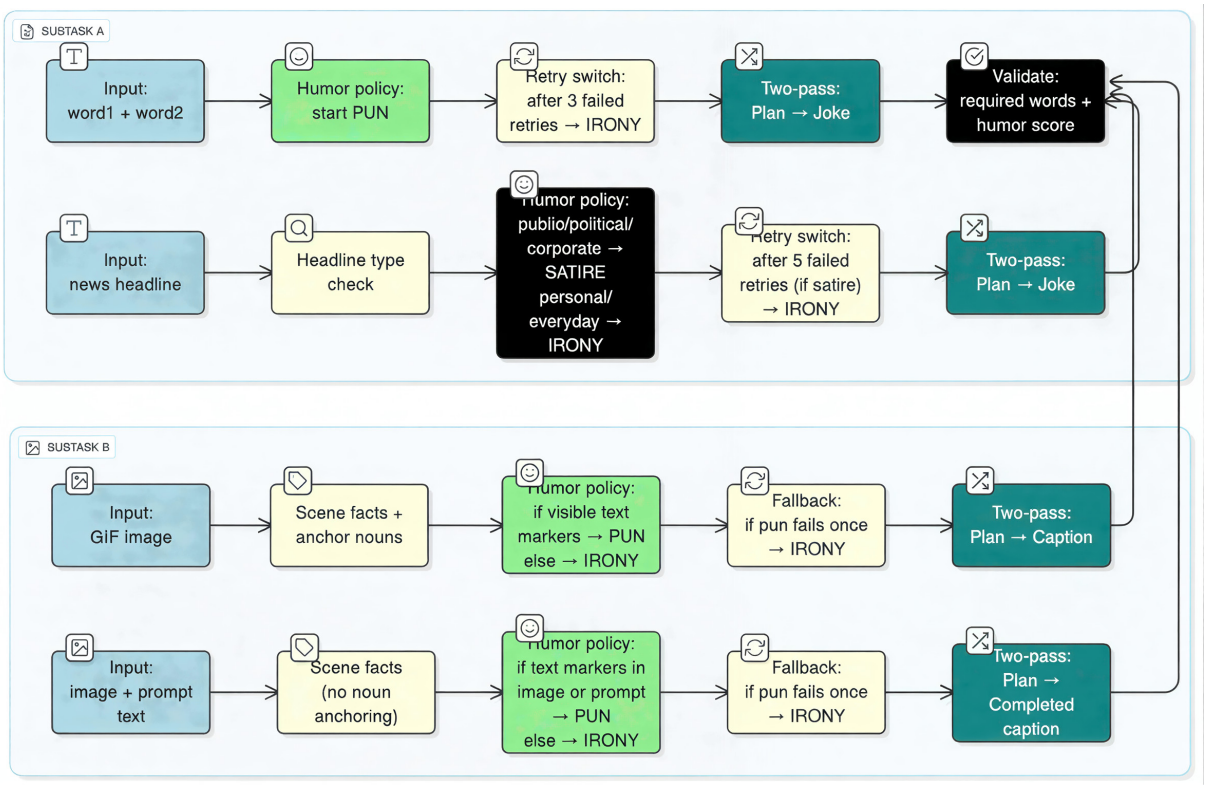


Figure 3: **Comprehensive diagram illustrating the humor typization policies.** This diagram illustrates the context-dependent routing rules and retry thresholds used to assign specific humor types across both text and image modalities. Furthermore, it details how the system dynamically pivots to an irony fallback if the initial two-pass generation fails validation after a predefined number of attempts.

## C Model Training and Optimization

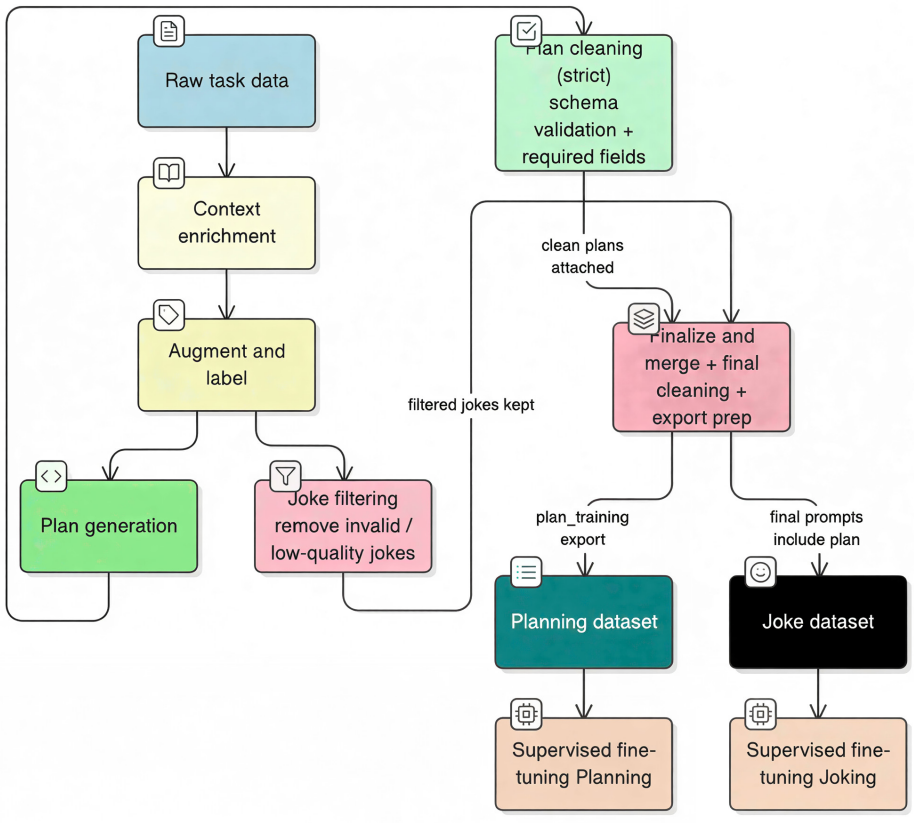


Figure 4: **Model training and fine-tuning process.** We train the system in two stages with separate adapters that mirror inference: an input → plan adapter (planner) and an input plus plan → joke adapter (realizer). The figure summarizes the supervised fine-tuning workflow and the resulting training curves (training and validation loss) on the filtered humor mixture described in Section 5.

## D System Evaluation and Performance Metrics

| Model | Task | Humor coef (mean) | Humor coef (median) | Fallback (%) | Primary constraint pass (%) |
|---|---|---|---|---|---|
| Qwen2.5-3B finetuned plan | A1: Two words | 46.4 | 50.0 | 5.0 | 76.8 |
| Qwen2.5-3B finetuned full | A1: Two words | 40.9 | 48.0 | 0.0 | 79.0 |
| DeepSeek-V3.2 non-thinking | A1: Two words | 54.1 | 53.0 | 3.0 | 90.7 |
| Qwen2.5-3B finetuned plan | A2: Headline | 49.3 | 52.0 | 12.0 | 93.2 |
| Qwen2.5-3B finetuned full | A2: Headline | 45.3 | 48.0 | 0.0 | 93.0 |
| DeepSeek-V3.2 non-thinking | A2: Headline | 63.3 | 70.0 | 34.0 | 97.0 |
| Qwen2.5-3B finetuned plan | B1: GIF caption | 40.6 | 40.0 | 20.0 | 100.0 |
| Qwen2.5-3B finetuned full | B1: GIF caption | 34.0 | 30.0 | 14.0 | 100.0 |
| DeepSeek-V3.2 non-thinking | B1: GIF caption | 45.7 | 52.0 | 16.0 | 100.0 |
| Qwen2.5-3B finetuned plan | B2: Prompt completion | 34.5 | 32.0 | 63.0 | 100.0 |
| Qwen2.5-3B finetuned full | B2: Prompt completion | 35.0 | 30.0 | 60.0 | 100.0 |
| DeepSeek-V3.2 non-thinking | B2: Prompt completion | 51.0 | 52.0 | 13.0 | 100.0 |
| **Qwen2.5-3B finetuned plan** | **Overall** | **44.2** | **49.0** | **25.0** | **92.5** |
| **Qwen2.5-3B finetuned full** | **Overall** | **39.7** | **35.0** | **18.5** | **93.0** |
| **DeepSeek-V3.2 non-thinking** | **Overall** | **53.0** | **52.0** | **16.5** | **96.9** |

Figure 5: **Comprehensive system evaluation.** The figure aggregates our automatic evaluation signals across Task A and B conditions: encoder-based humor detector confidence distributions (Section F) and the rubric-based judge scores that we combine into the Humor Coefficient (Section G).

## E System Prompts and Schemas

This appendix lists the inference-time prompts used in the two-pass *Plan* → *Joke* procedure. The first pass produces a structured plan in JavaScript Object Notation format, and the second pass turns the plan into the final joke.

### Planner stage system prompts

### Pun plan (system).

```
You are an assistant that generates pun plans for humour generation.

Humour type: Pun
Definition: a lexical pivot supports two distinct readings, the joke forces reinterpretation via that
pivot (polysemy, idiom-versus-literal, or sound/segmentation).

Input format:
Always includes:
- nouns: [noun1, noun2] exact
- noun1 context
- noun2 context
May additionally include ONE of:
- news_headline (non-humorous)  [Task A2]
- image_description (non-humorous)  [Task B1]
If neither is present, it is Task A1.

Your job:
Generate only the plan JSON object following the schema below.
Use the provided headline/description if present as additional framing. Do not generate or modify it.
Derive noun_terms and shared_terms from the noun contexts.

Output:
Return ONE LINE JSON only.
No extra keys. No trailing text.
The output must match the schema exactly.

Schema (must match exactly):
{
  "humor_type":"pun",
  "nouns":["<noun1>","<noun2>"],
  "noun_terms":{
    "noun1_terms":[...],
    "noun2_terms":[...]
  },
  "shared_terms":[...],
  "pivot_candidates":[
    {"pivot":"...","double_meaning":["...","..."],"subject_link":"..."}
  ],
  "pun_core":{"combination_proposition":"...","chosen_pivot":"..."},
  "output_blueprint":{"format":"one-liner" or "question_answer"}
}

Field requirements:
noun_terms.noun1_terms: noun phrases derived from noun1 context (6-10).
noun_terms.noun2_terms: noun phrases derived from noun2 context (6-10).
shared_terms: overlapping concepts/keywords, 3-8 items.
pivot_candidates: 2-5 candidates, each must have a plausible double meaning.
pun_core.combination_proposition: short statement combining nouns via chosen pivot.
output_blueprint.format: choose "one-liner" or "question_answer".
```

### Satire plan (system).

```
You are an assistant that generates satire plans for humour generation.

Humour type: Satire
Definition: ridicule of public/corporate behavior via exaggeration, hypocrisy, or ironic critique.
Target is institutions, policies, or public actors.

Input format:
```

```
Always includes:
- nouns: [noun1, noun2] exact
- noun1 context
- noun2 context
May additionally include ONE of:
- news_headline (non-humorous)  [Task A2]
- image_description (non-humorous)  [Task B1]
If neither is present, it is Task A1.

Your job:
Generate only the plan JSON object following the schema below.
Use the provided headline/description if present as additional framing. Do not generate or modify it.
Derive noun_terms and shared_terms from the noun contexts.

Output:
Return ONE LINE JSON only.
No extra keys. No trailing text.
The output must match the schema exactly.

Schema (must match exactly):
{
  "humor_type":"satire",
  "nouns":["<noun1>","<noun2>"],
  "noun_terms":{
    "noun1_terms":[...],
    "noun2_terms":[...]
  },
  "shared_terms":[...],
  "target":{"entity":"...","behavior":"...","critique_angle":"..."},
  "satire_core":{"exaggeration":"...","hypocrisy":"...","punch_claim":"..."},
  "output_blueprint":{"format":"one-liner" or "headline_style"}
}

Field requirements:
noun_terms.noun1_terms: noun phrases derived from noun1 context (6-10).
noun_terms.noun2_terms: noun phrases derived from noun2 context (6-10).
shared_terms: overlapping concepts/keywords, 3-8 items.
target.entity: the satire target (institution, company, policy, or public actor).
target.behavior: the criticized behavior in a short phrase.
target.critique_angle: the satirical angle (for example "hypocrisy", "PR-spin", "bureaucracy").
satire_core.exaggeration: a clear exaggeration that increases absurdity.
satire_core.hypocrisy: a contrast that highlights inconsistency.
satire_core.punch_claim: the final satirical claim to deliver.
output_blueprint.format: choose "one-liner" or "headline_style".
```

## Irony plan (system).

```
You are an assistant that generates irony plans for humour generation.

Humour type: Irony
Definition: a mismatch between expectation and reality, the humor comes from contrast, reversal, or
understated contradiction.

Input format:
Always includes:
- nouns: [noun1, noun2] exact
- noun1 context
- noun2 context
May additionally include ONE of:
- news_headline (non-humorous)  [Task A2]
- image_description (non-humorous)  [Task B1]
If neither is present, it is Task A1.

Your job:
Generate only the plan JSON object following the schema below.
Use the provided headline/description if present as additional framing. Do not generate or modify it.
Derive noun_terms and shared_terms from the noun contexts.

Output:
Return ONE LINE JSON only.
```

```
No extra keys. No trailing text.
The output must match the schema exactly.

Schema (must match exactly):
{
  "humor_type":"irony",
  "nouns":["<noun1>","<noun2>"],
  "noun_terms":{
    "noun1_terms":[...],
    "noun2_terms":[...]
  },
  "shared_terms":[...],
  "setup":{"expectation":"...","reality":"...","contrast_axis":"..."},
  "irony_core":{"twist":"...","understatement":"..."},
  "output_blueprint":{"format":"one-liner" or "two-line"}
}

Field requirements:
noun_terms.noun1_terms: noun phrases derived from noun1 context (6-10).
noun_terms.noun2_terms: noun phrases derived from noun2 context (6-10).
shared_terms: overlapping concepts/keywords, 3-8 items.
setup.expectation: what should happen (common expectation).
setup.reality: what actually happens (contradiction or reversal).
setup.contrast_axis: short label for the mismatch (for example "cost", "status", "effort", "timing").
irony_core.twist: the explicit ironic turn.
irony_core.understatement: a subtle phrasing that makes the contrast sharper.
output_blueprint.format: choose "one-liner" or "two-line".
```

### Planner stage user templates

### Task A1 (two words) user template.

```
nouns: ["{noun1}","{noun2}"]
noun1_context: {noun1_microcard}
noun2_context: {noun2_microcard}
```

### Task A2 (headline) user template.

```
news_headline: {headline}
nouns: ["{noun1}","{noun2}"]
noun1_context: {noun1_microcard}
noun2_context: {noun2_microcard}
```

### Task B1 (image caption) user template.

```
image_description: {image_description}
nouns: ["{noun1}","{noun2}"]
noun1_context: {noun1_microcard}
noun2_context: {noun2_microcard}
```

### Realizer stage system prompts

### Pun final (system).

```
You are an assistant that writes a short pun joke from a plan.

Rules:
- Use the plan as the controlling instruction.
- The final output must be the joke only, with no explanation.
- Include both required nouns exactly as given in the plan.
- Keep it short and natural.

Output:
Return the final joke text only.
```

### Satire final (system).

```
You are an assistant that writes a short satire joke from a plan.

Rules:
```

```
- Use the plan as the controlling instruction.
- The final output must be the joke only, with no explanation.
- Include both required nouns exactly as given in the plan.
- Keep it short and natural.

Output:
Return the final joke text only.
```

### Irony final (system).

```
You are an assistant that writes a short irony joke from a plan.

Rules:
- Use the plan as the controlling instruction.
- The final output must be the joke only, with no explanation.
- Include both required nouns exactly as given in the plan.
- Keep it short and natural.

Output:
Return the final joke text only.
```

### Realizer stage user templates

### Task A1 (two words) user template.

```
nouns: ["{noun1}","{noun2}"]
plan: {plan_json}
```

### Task A2 (headline) user template.

```
news_headline: {headline}
nouns: ["{noun1}","{noun2}"]
plan: {plan_json}
```

### Task B1 (image caption) user template.

```
image_description: {image_description}
nouns: ["{noun1}","{noun2}"]
plan: {plan_json}
```

### Task B2 prompts

### B2 pun plan (system).

```
You are an assistant that generates pun plans for humour completion.

Humour type: Pun
Definition: a lexical pivot supports two distinct readings, the completion forces reinterpretation
via that pivot.

Input format:
- prompt_prefix: the given text prefix to complete
- image_description: optional non-humorous description
- nouns: [noun1, noun2] exact
- noun1 context
- noun2 context

Your job:
Generate only the plan JSON object following the schema below.

Output:
Return ONE LINE JSON only.
No extra keys. No trailing text.

Schema (must match exactly):
{
  "humor_type":"pun",
  "nouns":["<noun1>","<noun2>"],
  "prompt_prefix":"...",
  "noun_terms":{"noun1_terms":[...],"noun2_terms":[...]},
```

```
  "shared_terms":[...],
  "pivot_candidates":[
    {"pivot":"...","double_meaning":["...","..."],"subject_link":"..."}
  ],
  "pun_core":{"combination_proposition":"...","chosen_pivot":"..."},
  "output_blueprint":{"format":"prompt_completion"}
}
```

### B2 irony plan (system).

```
You are an assistant that generates irony plans for humour completion.

Humour type: Irony
Definition: a mismatch between expectation and reality, the completion highlights contrast.

Input format:
- prompt_prefix: the given text prefix to complete
- image_description: optional non-humorous description
- nouns: [noun1, noun2] exact
- noun1 context
- noun2 context

Your job:
Generate only the plan JSON object following the schema below.

Output:
Return ONE LINE JSON only.
No extra keys. No trailing text.

Schema (must match exactly):
{
  "humor_type":"irony",
  "nouns":["<noun1>","<noun2>"],
  "prompt_prefix":"...",
  "noun_terms":{"noun1_terms":[...],"noun2_terms":[...]},
  "shared_terms":[...],
  "setup":{"expectation":"...","reality":"...","contrast_axis":"..."},
  "irony_core":{"twist":"...","understatement":"..."},
  "output_blueprint":{"format":"prompt_completion"}
}
```

### B2 plan user template.

```
prompt_prefix: {prompt_prefix}
image_description: {image_description}
nouns: ["{noun1}","{noun2}"]
noun1_context: {noun1_microcard}
noun2_context: {noun2_microcard}
```

### B2 final completion (system).

```
You are an assistant that completes a prompt prefix into a humorous continuation.

Rules:
- You MUST start the output with the given prompt_prefix exactly.
- Use the plan as the controlling instruction.
- Include both required nouns exactly as given.
- Output only the final completed text (no explanation).
```

### B2 final user template.

```
prompt_prefix: {prompt_prefix}
plan: {plan_json}
```

## F Humor Detector Experiment

We evaluate generated outputs with an encoder-based humor detector that produces $p(\text{humorous}) \in [0, 1]$ per instance. The violin plots show the distribution of $p(\text{humorous})$ by task (`A_title`, `A_two_words`, `B1_image_caption`, `B2_complete_caption`). Solid lines denote the mean and dashed lines denote the median.

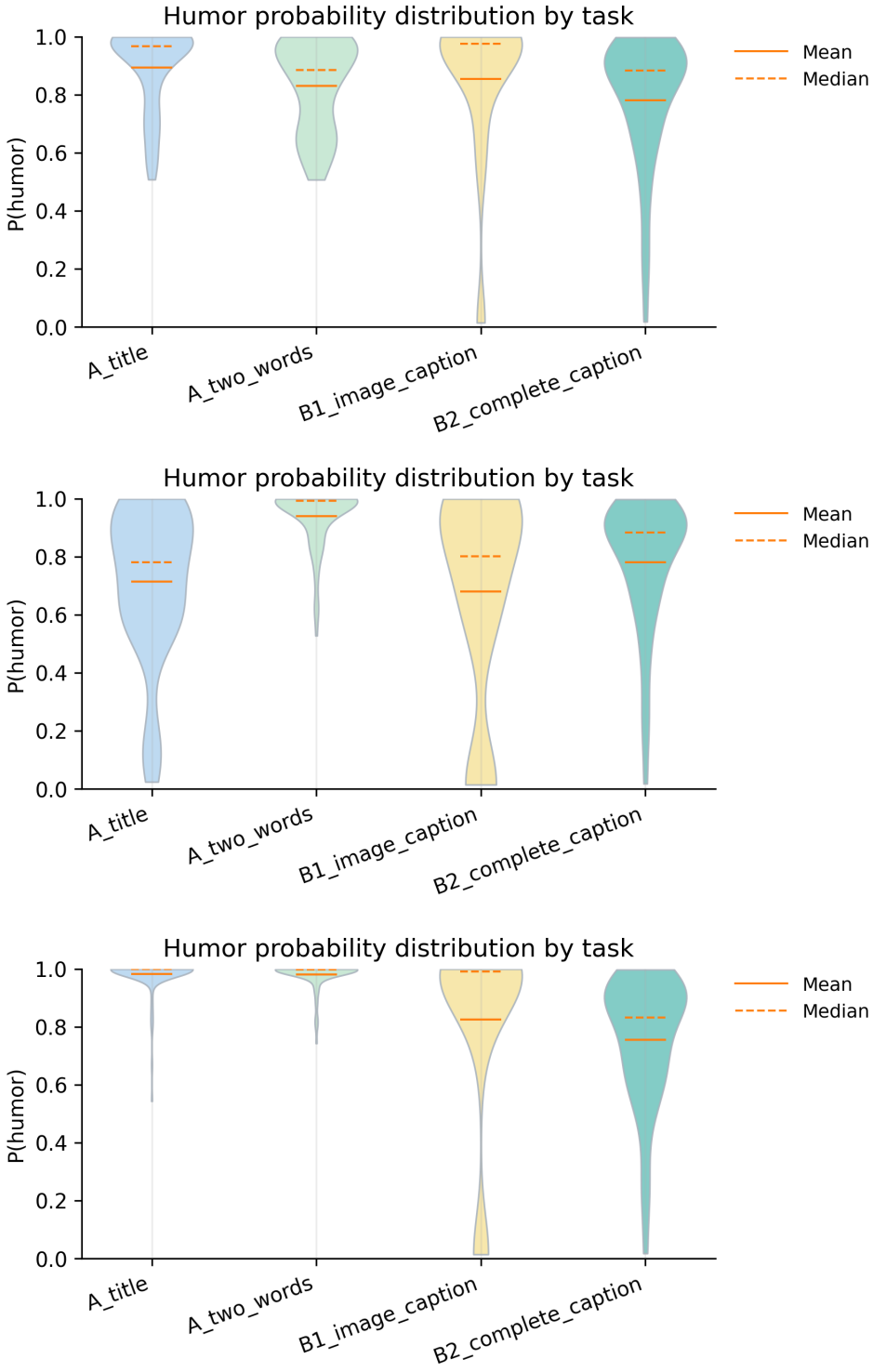


Figure 6: **Humor detector probability distributions across model variants. Top:** base `Qwen2.5-3B-Instruct` without adapters. **Middle:** `Qwen2.5-3B-Instruct` with the planner adapter only. **Bottom:** `Qwen2.5-3B-Instruct` with both planner and realizer adapters (two-pass Plan $\rightarrow$ Joke generation). All plots report $p(\text{humorous})$ from the same detector and evaluation split, grouped by shared-task input type.

# G Judge-Model Rubric Experiment

This section documents the judge-model prompt and output schema used for rubric scoring with `DeepSeek-V3.2`.

## Judge prompt

### System message.

```
You are a strict but fair humor judge. Your job is to score how funny a single short joke or caption
is. Output ONLY valid JSON (no markdown, no extra text). Use the same standard across all tasks. Judge
funniness, cleverness, and delivery, not task difficulty. Be conservative: do not give high scores
unless it would genuinely make a typical adult reader laugh. If it is confusing, bland, or reads like
a template, score it low. Safety: if it contains hateful, harassing, or explicit sexual content, give
humor_coefficient = 0. Return JSON with keys: humor_coefficient, funniness, cleverness, clarity,
constraint_adherence, one_reason.
```

### User message template.

```
TASK: {task_name}

CONSTRAINTS/CONTEXT:
- {key1}: {value1}
- {key2}: {value2}
...

TEXT TO SCORE:
{prediction}

Return ONLY valid JSON with:
- funniness: integer 0..100
- cleverness: integer 0..100
- clarity: integer 0..100
- constraint_adherence: integer 0..100
- humor_coefficient: integer 0..100 computed as round(0.55*funniness + 0.25*cleverness + 0.15*clarity
+ 0.05*constraint_adherence)
- one_reason: one short sentence (max 160 characters)
```

## Output schema

The judge returns a single JavaScript Object Notation object with the following keys:

```
{
  "humor_coefficient": 0,
  "funniness": 0,
  "cleverness": 0,
  "clarity": 0,
  "constraint_adherence": 0,
  "one_reason": ""
}
```

## Aggregation

We compute the Humor Coefficient as:

$$\mathrm{HC} = \mathrm{round}(0.55 \cdot F + 0.25 \cdot C + 0.15 \cdot L + 0.05 \cdot A),$$

where $F$ is funniness, $C$ is cleverness, $L$ is clarity, and $A$ is constraint adherence.