FunnyBorg at SemEval-2026 Task 1: Humor Generation

Stefan Oprea and Lacrimioara Toma Oprea and Maria-Teodora Paval-Istrate
Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iasi
{stefan.oprea, lacrimioara.toma, maria.istrate}@info.uaic.ro

Diana Trandabat
Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iasi
diana.trandabat@info.uaic.ro

Daniela Gifu
Faculty of Computer Science, "Alexandru Ioan Cuza" University of Iasi
Institute of Computer Science, Romanian Academy - Iasi Branch
daniela.gifu@info.uaic.ro

Abstract

Our team competed in the SemEval-2026 Task 1: MWAHAHA: Humor Generation. This is a task for generation of computational humor. The generated jokes are text-based, but also include memes, for captioning an image. Our approach involved prompt engineering using a voting system. We obtained rank 1 in one of the subtasks, and rank 2 in three other subtasks.

1 Introduction

LLMs researches focus on STEM, not humor, thus the field of computational humor is challenging. LLMs lack phonetic understading and pragmatic inference poses even more challenges.

While the field of humor evaluation is very strong, and it was present in SemEval competitions in the last years, humor generation is in development. In the past, there have been some template-based approaches, which attempted to create expert systems for creating humor. Better, more successful approaches involve using LLMs.

2 Background

For this SemEval task, we had to solve several subtasks:

- Subtask A English
  - Headline based - Given a newspaper headline, generate a funny joke.
  - Words based - Giver two words, generate a joke which contains both words.
- Subtask A Spanish - Same, but in Spanish.
- Subtask A Chinese - Same, but in Chinese.
- Subtask B1 - We are provided with a GIF, and we must generate a funny caption.
- Subtask B2 - We are provided with a GIF, as well as a caption with blanks. We have to create a funny joke filling in the blanks.

3 System overview

3.1 System architecture - microservices based

Our first approach was a system architecture using microservices approach with a REST API. This has several benefits: On the one hand, it facilitates collaboration between team members, as each team member can work on his own microservices, without having to deploy a common codebase to a shared compute environment. Every team member has his own codebase and his own compute environment, where the takes care of deploying and operating his services.

On the other hand, this could improve security, as each service runs in an isolated environment, and they communicate over HTTP. But there is a major caveat, as this poses the risk of prompt injection.

In practice, there are several approaches to mitigate prompt injection. Firstly, providing secure authentication between the microservices guards against malicious requests coming from the internet. This way, we ensure that the requests are coming from our microservices. But this does not guard against malicious prompts generated by the microservice itself, for example by virtue of a library which is affected by a supply chain attack. Therefore we should review the code deployed in each microservice and provide a security review of

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3446–3448

the microservice. This defeats the purpose of the microservice architecture.

Secondly, it is possible to guard against prompt injection by other means, which don't imply authentication: attempting to sanitize the incoming prompts, hardening the prompter or limiting the capabilities of the LLM client.

In this architecture, we devised several modules, where each module is a web service with its own repository and endpoints.

3.1.1 Runner

The Runner is the orchestrator, which calls the other services. It calls into our internal services, as well as the LLMs for humor generation. The endpoints and credentials are stored in a configuration file, and we make sure not to commit it to the public repo.

3.1.2 Validator submodule

The Validator is a submodule, which is used by the runner. It checks if the generated jokes respects joke size, include mandatory words etc. It also strips headline and $\text{fill}_i n_t he_b lank text, if present.$

3.1.3 Translator service

The translator service provides natural language translation and facilitates the subtask A, translating between English, Spanish and Chinese.

3.1.4 Image recognition service

The Image recognition service allows you to submit a gif (image or short clip). It extracts and returns a list of hints.

3.1.5 Prompt engineerers

The Prompt engineerers are services which create prompts. They take a batch of joke commands, and return prompts.

Each joke command can contain:

- headline

  The generated joke must be related to a given news article headline (it could be a punchline, or a joke inspired by the headline). The generated joke need not contain the headline, as it is added by the battle arena and displayed to the annotators.

- keywords

  The generated joke MUST include these keywords.

- $\text{fill}_i n_t he_b lank$

  The generated joke must fill in the blanks. The generated text need not contain the blanks, as they are added by the battle arena.

- hints

  These are hints generated by the image recognition service. They are extracted from the GIF. They need not be included mot-a-mot in the generated joke.

For each joke command, the service returns one or more LLM prompts.

We use several Prompt engineerer microservices. Each prompt engineerer can optionally use a joke classes database of its own.

3.1.6 Prompter services

The Prompters takes some jokes prompts, and call LLMs to generate jokes. It cleans up the generated jokes and returns just the joke text, without the LLM's chit-chat.

3.1.7 Humor evaluator services

The Evaluator service performs humor recognition. We send it a list of jokes, and it selects and returns the funniest one.

3.2 System architecture - pipeline based

Our second architecture uses a pipeline approach, which is widely used. There are many commercially available PaaS solutions for ML pipelines, but they require registration, and the free plan offers limited capabilities.

For these reasons, we decided to use a manual approach for pipelines, where each pipeline step takes as input one or more TSV files, and outputs a single TSV file.

3.2.1 Error handling

We ran the pipeline steps in Google Colab. This required some error handling. For this, we created a loop to perform retries of the pipeline step. Also, we opened the output files in append mode, and implemented a deduplication strategy to skip the lines which have been already generated.

3.2.2 Image based tasks

As regards the image-based tasks, this presented a challenge, given that the image provided by the organizers is in GIF format. The machine learning model used supports single images, but not GIFs.

We processed the GIF and extracted only one frame as a PNG image, to submit to the model.

For the image based tasks, it was necessary to use an image. The model used permitted the usage of images, but to improve performance, we decided to use memoization. For this, we created TSV files with hints for the images. This involves splitting the input file, which is a GIF, into frames, and using a single frame, as the used LLM only allows to upload a single frame, not a GIF. Then we generated a description of the image and stored it in the hints file.

3.2.3 Foreign language tasks

For the tasks which are not in English, we considered two approaches. Firstly, it's possible to translate the instructions into English, generate the joke in English, then translate the joke into the foreign language. Secondly, it's possible to prompt the LLM to generate the joke directly in the foreign language. We went with the second approach, as it performs really well.

4 Experimental setup

In search for LLMs to use for this task, we only considered the free plan. An approach that did not work was to use major LLMs, as their free plan imposes a strict RPD (requests per day) limit, didn't make it feasible.

We decided to use the Gemma model from Google, which offers both an API with a good free plan, as well as the possibility to run the model locally and fine-tune it. This model is designed to be run on client hardware, such as laptops, mobile phones and wearables, therefore it is lightweight enough to be run in a free environment with reasonable system resources.

5 Results

The running time for a normal prompt was about 10 seconds. The Requests per Minute allowed in the free plan permitted us to run the prompts continuously, without the need to throttle them. We ran the prompts in series, without the need to parallelize, as the running time for a single task is just a couple of hours.

For the text-based subtasks (A), our team obtained 2nd rank. For the text-plus-image subtask (B2), we obtained 1st rank. In the image-based subtask (B1) however, we obtained last rank.

Conclusion

In conclusion, our prompt engineering approach posed a number of challenges. By using good prompt engineering, we obtained satisfactory results. The used Gemma model performed well, even though it is not as powerful as, for example, the bigger Gemini models it is based on.

References