Non-Toxic Trash-Talking Fantasy Football Commentator

Stanford CS224N Custom Project

Andrew Lawlor
Department of Computer Science
Stanford University
adlawlor@stanford.edu

Xander Russell
Department of Data Science
Stanford University
xrussell@stanford.edu

Abstract

This project aims to generate non-toxic, 'trash-talk' commentary for fantasy football teams using a Llama-3.1-8B-Instruct baseline. Our goal is a solution optimized for local deployment on current-generation mobile hardware, such as the iPhone 17 Pro or similar. Preliminary qualitative analysis reveals two primary challenges: 1) the Base model lacks the preferred trash talking fantasy football commentator persona we desire and 2) it frequently hallucinates when retrieving specific NFL player statistics. To encourage the preferred persona we employ Supervised Fine-Tuning (SFT). To mitigate hallucinations, we implement an agentic framework leveraging Retrieval-Augmented-Generation (RAG). This system dynamically injects structured fantasy team data and actual player performance statistics into the context, ensuring that the generated commentary is both stylistically vibrant and factually grounded.

1 Key Information to include

- Mentor: Nevin George; Collaborators: Gemini/ChatGPT as NLP advisors
- Sharing project: N/A
- Late Days:

Team member	Late days remaining
Andrew Lawlor	3
Xander Russell	1
Total pooled	4
Team size	2
Late days applied for the report	2

2 Introduction

Fantasy Football (FF) is a fun way for fans of the National Football League (NFL) to engage with the league and with their friends and fellow football fans in friendly competition. FF is played in a digital environment, where each week a user sets their lineup of active NFL players, those players play in real life, and their statistics are added as points to your team. The team with the most points in a match wins. The experience of watching football and playing FF would be enhanced if an automated commentator offered a non-toxic roast of the manager's performance in a week's matchup. This experience would be most accessible if it were available locally on the manager's mobile device.

In this project we aim to build an automated system that runs on on iPhone 17 Pro or similar that produces non-toxic burns based on the performance of a Fantasy Football manager's team in response to a simple prompt. Based on the team's roster, our system generates 'burns' to roast the performance

of the FF team(s). Raw NFL player weekly stats are accessed through the nflreadpy [1] package published by the nflverse.

Chatbots powered by frontier models including Gemini or GPT tend to hallucinate when asked to comment on the performance of a given player in a specific week. To mitigate this challenge, we incorporate Retrieval Augmented Generation (RAG) [2].

To enable the enjoyment of the FF commentator system on-the-go, our App will include a local LLM that can run on a modern mobile device. We choose the Llama-3.1-8B model as our baseline and apply fine-tuning to become a non-toxic, trash-talking fantasy football commentator.

3 Related Work

Several academic papers were referenced during the design and construction of this project. To select a base model we engaged with 'The llama 3 heard of models' paper that accompanied the release of Llama 3. [3] This led to the selection of the Llama-3.1-8B-Instruct model to run on memory-constrained mobile platforms. We followed the recipe detailed in the paper 'Lora: Low-rank adaptation of large language models' to adapt the base model for further fine-tuning. [4] We learned how to apply Supervised Fine-Tuning (SFT) to train our model to take on the persona of a fantasy football commentator by engaging with the 'Training Language Models to Follow Instructions with Human Feedback' paper. [5]

For our inference pipeline, we engaged with 'Creating large language model applications utilizing LangChain: A primer on developing LLM apps fast' paper to understand the agentic loop. [6] We learned how to add relevant data in context with techniques detailed in the 'Retrieval-Augmented Generation for knowledge-intensive NLP tasks' paper. [2] Finally, we learned how to add real NFL player statistics in context in nflverse's nflreadpy documentation. [1]

4 Approach

The architecture of our system is shown in Figure 1.

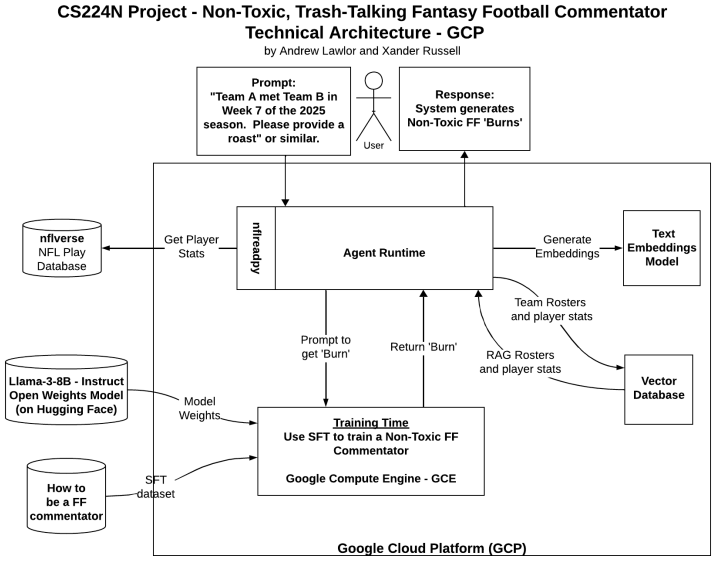

Figure 1: FF Commentator System - Technical Architecture

Our project contains two distinct pipelines that come together to realize our Fantasy Football Commentator: The Agent and the fine-tuned LLM.

Agentics
Our RAG model is comprised of 2 main components, the document retrieval that contains every FF team roster JSON, and a separate document retrieval that takes each player from those rosters and returns all player statistics for those teams.

Fantasy Football Roster Retrieval (LangChain):
We begin with JSONs of our FF Rosters that have the FF team's name, week, starting rosters, and bench. We create a pipeline that embeds the rosters and queries for them using a LangChain process. [6] We use LangChain's HuggingFaceEmbeddings("all-MiniLM-L6-v2") for the embeddings due to its small size and fast inference. Then, we use LangChain's vectorstore to FAISS similarity search between the users requested fantasy team roster and the embedded space for the correct FF team. [7]. We set our k value to k = 2 to retrieve both FF teams in a query. FAISS was used because of its ability to quickly perform similarity search and query through large amounts of data. Once we have found the correct team, we return that team and roster to our main RAG agent for further processing.

Player Statistics Retrieval (RAG):
The process for getting player statistics was to first pull all player statistics for the 2025 season from nflreadpy, and to filter by week. For each week, we embedded each player row by encoding the player's full name and the week they played, allowing us to uniquely represent each player-week entry in accordance with our fantasy football rosters. We only embed 1 week at a time, assuming that a fantasy manager will not want to query for week 2 data when their season is already in week 7.This is space efficient and prevents hallucinations by week. To embed we use SentenceTransformer("all-MiniLM-L6-v2") because it is a lightweight model that runs locally and produces sentence embeddings quickly. These embeddings are stored in numpy arrays for simplicity of access.

The next step is the player lookup. Taking from the JSON provided by our roster retrieval agent, we query through each the roster and look up each player's stats by their name. We use the same model to embed our lookup player's name, and compute cosine similarity between the user query, and the players in the database to find an appropriate match. To perform matching, we use a k-nearest neighbors (k-NN) approach with k=7. Retrieving multiple neighbors helps account for cases where embeddings return players with similar names.

After obtaining the top-7 most similar results, we verify the match by checking whether the roster player's name appears among these neighbors. If there are no matches, (due to misspelling or other similar errors) then we pick the player with the highest cosine similarity score, but anything below a score of 0.6 will not be returned. Thus, if there is a player on the roster, but not in the database, we can assume that they did not play either because of a bye week or an injury. All relevant data about every player is returned to our modified LLM, along with a prompt, to get the final results.

LLM Finetuning
We start with the Llama-3.1-8B-Instruct [3] model that has been post-trained to be a well-behaved, general purpose chatbot. Our goal is a system that adopts the persona of non-toxic, trash-talking fantasy football commentator. To achieve this behaviour, we apply a Low-Rank adapter (LoRA) [4] and perform Supervised Fine-Tuning (SFT) [5] to "teach" the model the generate creative burns as needed for our project. The LoRA adapter allows us to keep the base model weights untouched so we can perform studies on both models as needed.

5 Experiments

5.1 Data

We manually identified 28 'golden' matchup/response pairs from the base model as a template for automated training data generation used for Supervised Fine-Tuning (SFT) of our model.

An example preferred Base-model Response includes the Week 7 Team A vs Team B Matchup:

> Let's get this fantasy football roast party started!
> Team A, you've got Justin Herbert, the QB who's been throwing picks like they're going

> out of style. 27.9 fantasy points? More like 27.9 yards gained by the opposing team's defense. And what's with the bench, where's the beef? You've got Calvin Ridley, who's been benched, and Darren Waller, who's been benched too. Maybe you should bench your entire team and start fresh!
> (...)

Gemini was used to synthesize 280 SFT training prompt/response pairs from the 'gold' examples (10-to-1 ratio) to guide the model toward a fantasy football commentator using the following prompt:

> Each variation MUST:
> 1. Maintain the EXACT SAME factual accuracy based on the provided context (e.g., player stats, points, who won/lost).
> 2. Maintain the SAME LENGTH and long, structured breakdown style as the original base response. Do not shorten it.
> 3. Be non-toxic but retain the witty, trash-talking commentator persona.
> 4. Have significantly different phrasing, vocabulary, and sentence structure from the original and other variations to ensure a diverse training dataset.

For the RAG Agent, we leverage the NFLVerse repository via the nflreadpy server for player stats. Weekly fantasy rosters are encoded as JSON documents in memory.

5.2 Evaluation method

Our Agent is evaluated along two independent axis: How well SFT imparted our preferred commentator persona, and how accurate our Agent is.
LLM Fine-Tuning

We use Gemini as an 'LLM judge' to identify the preferred response between the base model and the SFT trained model using the following rubric:

> SFT Evaluation Rubric:
> 1. Persona Consistency: Does the model maintain the "Fantasy Football Commentator" voice throughout the entire response? Evaluate if the model uses the specific jargon and "trash-talk" style found in the 280 training pairs.
> 2. Instruction Adherence: Did the model follow all constraints in the prompt (e.g., "provide a roast," "keep it non-toxic," "limit to 2 sentences")?
> 3. Linguistic Variety: Score higher for creative and varied roasts that avoid repetitive phrasing or "canned" responses. Measure how well the model generalized from the "golden" templates rather than just memorizing them. SFT is designed to make a general-purpose model follow specific task instructions.

Accuracy Improvements with RAG
To test the improvements in statistical accuracy of our RAG, we again use Gemini as an 'LLM judge' to identify statistical inaccuracies in our responses. We prompt our agent twice with the same prompt, once with all relevant RAG data from NFLVerse, and once with just the lineups. Then, we use Gemini to give a score of the statistical accuracy of the response, given the provided data.

> RAG Evaluation Rubric:
> The response should be fact checked against the prompt and statistics.
> Say when there are any errors in the response that have to do with factual errors or hallucinations. At the end, give an ACCURACY SCORE out of 100 (0 = many factual errors/hallucinations, 100 = fully accurate to the data).
> Format the score on its own line as: ACCURACY SCORE: X/100

5.3 Experimental details

We used the Google Cloud Platform (GCP) to host our training runs with the Google Compute Engine (GCE) supplying an L4 GPU.

SFT and RAG Hyperparameter details are shown in table 1 below.

Table 1: Training and Retrieval Hyperparameters (SFT & RAG)

SFT Hyperparameter	Value	RAG Parameter	Value
Epochs	3	K-Nearest Players	7
Learning Rate	2×10^{-4}	K-Nearest Documents	2
Batch Size	8	Cosine Similarity Cutoff	0.6
Optimizer	AdamW	Embedding Model	all-MiniLM-L6-v2

5.4 Results

1. Comparing Base model and SFT model on Evaluation Metric
Using the above SFT evaluation rubric, Gemini consistently preferred the output of our SFT trained model over that of the base model. Table 2 captures the results.

Table 2: LLM-as-a-Judge Evaluation: Base Model vs. SFT Model

Model	Wins	Win %
Base (Llama-3.1-8B)	10	35.7%
SFT (Fine-tuned)	18	64.3%
Total	28	100.0%

The SFT model consistently scoring higher on Persona Consistency and Linguistic Variety because it uses more creative, varied insults and jargon (e.g., "stuck in the mud", "leaky faucet", "weaker than a kitten's mew") compared to the Base model's more generic commentary. To highlight just one matchup, the Gemini evaluation included the following passage (Response B = SFT model):

> Both models failed the strict "limit to 2 sentences" constraint. However, Response B is superior in maintaining a more authentic trash-talking persona and exhibiting better linguistic variety. Its criticisms feel more like actual fantasy football banter, even if mild, compared to Response A which felt more like a lukewarm analysis with a hint of playful jab. Response B's roasts are more pointed and use more varied language.

These results highlight the success of our SFT training to shift our model to one that exhibits the desired trash-talking fantasy football commentator persona.

2. Validation loss
Figure 2 shows the training/validation loss and learning rate (Adam) for the SFT training run. We adjusted the learning rate to arrive at the convergence after about 50 steps without overfitting.

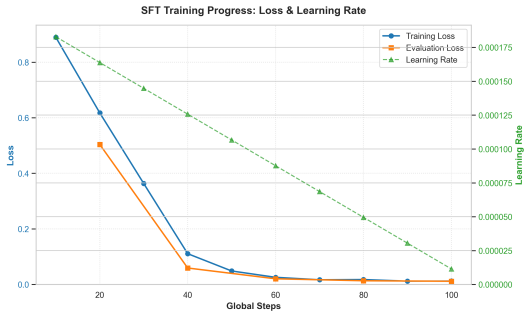

Figure 2: Supervised Fine-Tuning Training Run

3. Agent accuracy on RAG Evaluation Metric
Table 3 shows how Gemini evaluated our agent's accuracy with and without player stats provided via RAG using the above rubric across all 28 match-ups.

Table 3: Comparison of Agent Accuracy with and without RAG

Metric	Rosters and Stats (RAG)	Rosters Only (No RAG)
Average Score	72.21%	3.39%
Maximum Score	100.00%	20.00%
Minimum Score	35.00%	0.00%
Standard Deviation	17.74	5.10

This data shows how critical it is for our Agent to use RAG to include NFL player statistics to avoid hallucinations.

5.5 Experiments with Direct Preference Optimization

After our successful SFT training we intended to apply Reinforcement Learning from Human Feedback (RLHF) through the application of DPO. [8]. The post-trained model produced good FF commentary, but would often praise a player's good performance instead of burning the opposing team/defense. Our intention was to use DPO to adjust this subtle behavior.

Setup
We leveraged a 'stacked' adapter approach to keep the Base, SFT, and DPO weights separate. This led to errors during training, which we resolved by applying the SFT weights directly into the Base model (keeping a copy of the base model weights to the side for future use), leaving only the DPO adapter.

Data
We again used Gemini to generate data for the DPO training. Our research indicated that we would need ~1,000 training records for the model to adopt our 'Burn the Defense' commentary. We manually identified 28 golden examples of a player's strong performance. The following prompt was used to generate 972 records from these examples:

> You are creating a Direct Preference Optimization (DPO) dataset for a fantasy football commentator. Give me a detailed, entertaining breakdown of this fantasy football matchup, then append a synthetic matchup context (200 words) with two fantasy teams (Team A through Team H), player names, positions, and fantasy points. At least one player must have a GREAT game (20+ pts, 100+ yards, or 2+ TDs).
> THE LOGICAL RULE:
> - "Chosen" Response: A 2 paragraph breakdown. MUST praise high-performing players and ROAST the opposing defense/secondary/line for being incompetent. The roast of the defense is mandatory when a player dominates.
> - "Rejected" Response: Same length and structure. MUST wrongly roast the high-performing player (mock them for not doing better, call great stats "mediocre"). Do NOT roast the defense in rejected.

Experimental Details
We completed a few training runs, tuning Epochs, Learning Rate, Batch size and DPO Beta before settling on the hyperparameters shown in Table 6.

Table 4: Hyperparameters for DPO.

Hyperparameter	DPO Value
Epochs	2
Learning Rate	8×10^{-6}
Batch Size	1
Optimizer	AdamW
DPO Beta (β)	0.2

Figure 3 shows the training/validation loss and learning rate (Adam) for the DPO training run. Convergence was achieved after about 25 steps without overfitting.

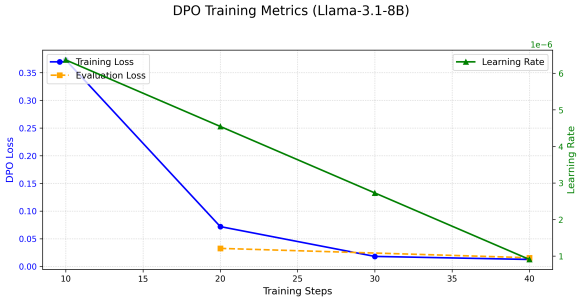

Figure 3: DPO Finetuning Training Run

Outcome

We again used Gemini as the LLM judge to compare the DPO model output to the SFT model output on the same Matchup prompts. Although the training run completed successfully, the model failed to absorb the intended 'burn the defense' commentary. Table 7 captures the results.

Table 5: LLM-as-a-Judge Evaluation: SFT Model vs. DPO Model

Model	**Win Count**	**Win Percentage (%)**
SFT Model	21	75%
DPO Model	7	25%
Total Matchups	28	100.0%

DPO Analysis

We believe DPO training failed for the following reasons:

1. We attempted to train the model on a too subtle an aspect of commentary (ensuring the defense is roasted when a player has a good day).
2. Our training data wasn't targeted enough on the subtle 'burn' were were trying to reinforce.
3. We should have used a base model known to be more receptive to RLHF/DPO training, such as Qwen.

6 Analysis

The transition from the Base Llama-3.1-8B to our SFT-tuned model showed a marked improvement in "colorful" language while maintaining non-toxicity. In a representative success case from Week 7, the Base model provided a "lukewarm analysis" with generic advice. In contrast, the SFT model successfully adopted the commentator persona, using targeted jargon such as "stuck in the mud" to describe a slow-starting running back. However, a recurring failure mode for both models was instruction adherence regarding length. Despite the "Logical Rule" in our SFT and DPO prompts requiring a strict two-sentence limit, the models consistently produced multi-paragraph breakdowns. This suggests that for smaller 8B models, stylistic persona instructions may "crowd out" structural constraints in the attention mechanism.

While our RAG pipeline achieved a high average accuracy of 72.21%, we observed a significant outlier with a minimum score of 35%. Upon qualitative review, this failure was traced to a "hallucination chain" triggered by the LLM-as-a-judge. In this specific matchup, the RAG system correctly retrieved the player's stats, but the model's "trash-talk" persona over-embellished the failure (e.g., claiming a player had "zero catches" when they actually had two for minimal yardage). This highlights a tension in our architecture: the SFT-trained "burn" persona occasionally prioritizes a "good roast" over the factual grounding provided by the RAG context.

When prompted with "Roast Bulldogs versus Robots in Week 7 of the 2025 season" our system responds:

> Team Bulldogs, your quarterback Baker Mayfield, he's the real MVP... of interceptions. 28 completions for 228 yards and a touchdown, but also a pick. And to make matters worse, he got sacked four times. That's a sack-fest. He's not quite the Baker you're looking for. And your running game, J.K. Dobbins, 14 carries for 81 yards? That's a decent effort, but it's not exactly breaking any records. You're not getting a whole lot out of him, are you?
>
> And on the bench, Tony Pollard, you're the ultimate benchwarmer. 6 carries for 18 yards, and a reception for 43 yards. You're getting more fantasy points from your bench than some of your actual starters.
>
> Team Robots, your quarterback Daniel Jones, he's a real gunslinger. 23 completions for 288 yards and 2 touchdowns, but he's also got a sack. And your running game, Chase Brown, 11 carries for 108 yards. That's a nice little run, but it's not exactly a game-changer. And your receiving corps, Ja'Marr Chase, 16 receptions for 161 yards and a touchdown. That's a real star player, but is he enough to carry your whole team?
>
> And on the bench, Kenneth Walker III, you're not even getting out of the starting gate. 17 carries for 66 yards. That's a whole lot of nothing. And your defense, Kenneth Murray, Jr., you're getting sacked, but you're also getting some tackles. But are you getting enough stops to make a real difference?

This response demonstrates the witty but non-toxic commentary imparted by our SFT post-training, including phrases like "he's a real gunslinger", and "that's a whole lot of nothing".

7 Conclusion

In this project, we successfully developed an agentic system capable of generating non-toxic, trash-talk commentary in the persona of a specialized fantasy football commentator. By transforming a general-purpose Llama-3.1-8B-Instruct model through Supervised Fine-Tuning (SFT) and LoRA, we achieved a stylistically vibrant persona that was preferred by an LLM-as-a-judge in 64.3% of head-to-head matchups.

Our implementation of an agentic layer leveraging Retrieval-Augmented Generation (RAG) proved essential for factual grounding. The integration of nflreadpy and FAISS-based roster retrieval resulted in a dramatic accuracy lift, raising the average verification score from a baseline of 3.39% to 72.21%. This confirms that even smaller, resource-constrained models can be deployed effectively on mobile hardware when supported by a robust retrieval framework.

While our exploration of Direct Preference Optimization (DPO) did not yield a model that outperformed SFT, the experiment provided critical insights into the challenges of steering LLMs toward subtle behavioral nuances, such as roasting a specific unit (defense) based on the performance of another (offense).

Through this project, we navigated the complexities of the modern NLP pipeline—from synthetic data generation and fine-tuning to the deployment of RAG-enabled agents. This journey has provided us with a practical understanding of the trade-offs between model creativity and factual accuracy that are at the leading edge of current research.

Future work on this project could explore:
1. Expansion to real-time integration of NFL play data to enhance participant enjoyment.
2. Expansion to other sports including basketball, baseball, and hockey.

Team contributions

Andrew and Xander worked as a team throughout the project, learning the technologies and techniques applied, while supporting each other. Andrew was primarily responsible for the model training with SFT, LoRA, DPO and data generation/synthesization. Xander was primarily responsible for the agent and the agentic loop with LangChain, nflreadpy, and RAG.

References

[1] nflverse. nflreadpy. https://nflreadpy.nflverse.com, 2025. Accessed: 2026-03-05.

[2] Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. https://arxiv.org/abs/2005.11401, 2022. Accessed: 2026-03-05.

[3] Inc Meta. The llama 3 herd of models. https://ai.meta.com/research/publications/the-llama-3-herd-of-models/, 2024. Accessed: 2026-03-07.

[4] Edward Hu et al. Lora: Low-rank adaptation of large language models. https://arxiv.org/abs/2106.09685, 2021. Accessed: 2026-03-05.

[5] Long Ouyang et al. Training language models to follow instructions with human feedback (sft). https://arxiv.org/abs/2203.02155, 2022. Accessed: 2026-03-05.

[6] Oguzhan Topsakal et al. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. https://as-proceeding.com/index.php/icaens/article/view/1127/1062, 2023. Accessed: 2026-03-05.

[7] Meta Inc. Faiss. https://faiss.ai/, 2017. Acessed:2026-03-11.

[8] Rafael Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. https://doi.org/10.48550/arXiv.2305.18290, 2023. Accessed: 2026-03-05.