# CSE 587 Final project report

(Goutham Deepak, Vedant Sawant)

## Introduction

The idea of the project was to generate approaches for any research problem. For this the topic we chose was LLM hallucination. LLM hallucination is one of the biggest problem which is currently being faced as LLMs are becoming more prevalent day by day where its use case in increasing. The LLM hallucinating and giving the wrong output can cause real world harm by spreading misinformation which can be costly. So a lot of research is currently being done on what causes LLMs to hallucinate and how to mitigate it. So we chose this topic so that our fine tuned LLM can generate approaches as well as help the user understand any doubts he will have by answering it. So we created 2 datasets and generated a pipeline for fine tuning to make it robust and can give a very clear and concrete guidance.

## Dataset Creation

We started with 156 research papers about LLM hallucination. This was for both LLM hallucination detection and mitigation. We used s Python script using Selenium to get all the paper's PDF links. This was done in google scholars. We used a pdf parser to get the contents of the PDF and make it into a text file. We sent each text file to the OpenAI o4-mini model with a prompt like "Extract and rewrite the full technical methodology section as numbered steps". We also extracted other info from the paper. So now we had all the technical methodologies in the form of giving or suggesting the solution which has been implemented in the paper rather than saying how the author did it. So the dataset had1289 questions

- The paper title
- the cleaned methodology text
- whether the LLM is white-box or black-box
- which dataset the paper used

**Question–Answer Pairs Dataset**

Using that first CSV, we made about ten simple questions for each paper. This wasn't template based. It was based on the technical terms of the paper along with the important stuff in the methodologies. We also used the OpenAI o4-mini model for this and generated about

**Testing Dataset**

For the testing dataset we hand created 50 questions which were a mix of different methodologies which could tell us which models performed better in which cases. These included questions to give the whole methodologies as well as small technical stuff in methodologies.

# LLM selection and training details:

**Base model chosen**: We've chosen Meta's **Llama-2-7b-chat-hf** as the foundation for all our fine-tuning experiments—it's a publicly released, instruction-tuned 7 billion-parameter conversational model that excels at dialogue and instruction-following tasks, while remaining compact enough to run on a single 16–24 GB GPU

**Model overview**: Meta-Llama/Llama-2-7b-chat-hf is the 7 billion-parameter, chat-optimized variant of the Llama 2 family, released July 2023. It's an auto-regressive, decoder-only Transformer fine-tuned for dialogue use cases.

- **Core architecture**
  - **Layers & dimensions**
    - 32 Transformer decoder layers
    - Hidden size (model dimension): 4096
    - MLP (intermediate) size: 11008
    - 32 attention heads (head dimension = 4096 / 32 = 128)
  - **Attention & embeddings**
    - Standard multi-head self-attention (no grouped-query attention at 7 B scale)
    - Rotary position embeddings (RoPE) for relative positional information Byte-pair encoding (BPE) vocabulary of 32 000 tokens
  - **Activation & normalization**
    - SwiGLU activation in feed-forward blocks
    - RMSNorm layers instead of LayerNorm
  - **Context window**
    - Up to 4 096 input tokens (double Llama 1's 2 048)
- **Pretraining**
  - Trained on ∼2 trillion tokens from publicly available sources (Common Crawl, Wikipedia, public-domain books, code, etc.)
  - Pretraining data cutoff: September 2022
- **Fine-tuning & alignment**
  - Supervised fine-tuning on instruction datasets
  - Reinforcement learning from human feedback (RLHF) via rejection sampling + PPO
  - Over 1 million new human-annotated examples for helpfulness and safety
- **Benchmarks & use case**
  - Excels at instruction following and dialogue tasks
  - Outperforms other open-source chat models on most benchmarks, and rivals closed-source systems (e.g., ChatGPT, PaLM) in human evaluations for helpfulness and safety

Fine-tunes Llama:

- **Model A – QA-Only Adapter**
  - **Dataset:** all "instruction–response" QA pairs built from your Excel summaries
  - **Size:** ~3× (number of rows) records covering generic hallucination prompts, type-specific, dataset-specific, and "Tell me about…" instructions
- **Model B – Summary-Only Adapter**
  - **Dataset:** 156 paper summaries on LLM hallucination (one summary → one record)
  - **Prompt:** each summary framed as "### Response" to a generic or title-based instruction
- **Model C – Combined Adapter**
  - **Dataset:** union of QA-pairs + paper summaries (~combined size of A+B)
  - **Training flow:**
    1. Load base quantized model + fresh LoRA adapter → fine-tune on QA data (as in Model A)
    2. Without reinitializing, continue fine-tuning the same adapter on the summary data (as in Model B)
    3. Optionally, run an extra epoch over the full combined set to smooth transitions
- **Training:**
  - 4-bit quantized Llama-2 base + single LoRA adapter
  - 3 epochs, batch size 1 with 16× gradient accumulation
  - LR = $2 \times 10^{-5}$, fp16, paged_adamw_8bit

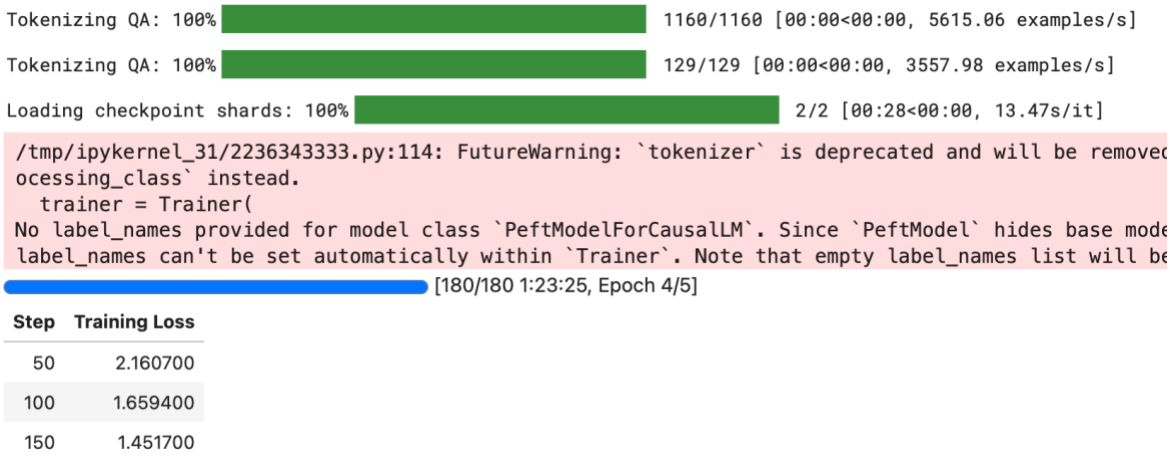
# Experiment conducted

The idea of our project was to build a pipeline where when the user asked a prompt to help him solve the problem of LLM hallucination it would give him suggestions on how to go about it. The goal was to find out which model out of the 3 produced better results. What kind of questions are answered better in by each model and also see if the length of the tokens generated by the 3 varies a lot. This is done as our training dataset is widely different and we wanted to see if that affected it based on how it should.

The user can even ask questions related to it and get things clarified. We created 2 datasets to do this. The first dataset had one row per paper with columns for the exact title, the cleaned methodology text, the Dataset which it used and the type of LLM (white box or black box). The second dataset consists of question and answer pair. For each paper we filled in almost 10 questions which gave an initial question that could be asked based on the methodology. This had 1289 high quality examples where the answer was based on the methodology. The model we chose for fine tuning was Llama-2-7B. We chose this as it was ideal for our experiment as 7 billion parameters is large enough to capture language patterns yet be small enough so that we can fine tune it on a single GPU. The model was loaded with 4-bit precision using BitsAndBytesConfig to reduce the GPU requirements. LoRA adapters was applied with rank r = 16, alpha = 32, and dropout = 0.05. We also used a few memory saving tricks so we could

fine tune on just one GPU. We turned on gradient checkpointing so that the model only stores some of its internal data as it runs and recomputes the rest on the fly. So this will cut down RAM use. We also switched off the model's generation cache which is used to speed up inference but isn't needed during training. Instead of updating all seven billion weights we froze the original model parameters and only trained the small LoRA adapter. By combining these steps we kept our GPU memory requirements low enough.

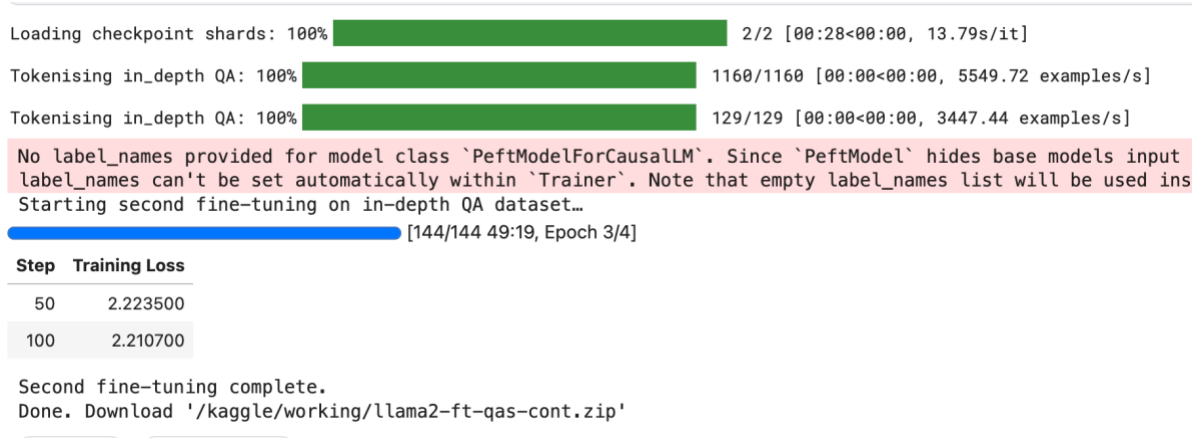
In the first script we fine-tuned Llama-2-7B directly on our 1289 question answer examples. We began by reading the Excel file of QA pairs and turning each row into a single text block. We ran these through the Llama-2 tokenizer. Then we loaded the full Llama-2-7B-chat model in 4-bit precision (using BitsAndBytesConfig with NF4 quantization and double quant), attached a small LoRA adapter (rank = 16, alpha = 32, dropout = 0.05), and enabled two memory-saving tricks: gradient checkpointing (which recomputes activations on the fly) and disabling the model's inference cache. We froze all seven billion original weights so only the LoRA adapter learned anything new, and then trained for five passes over the data with a micro-batch size of one plus sixteen-step gradient accumulation, a learning rate of $2\times10^{-5}$, mixed-precision FP16, and the paged_adamw_8bit optimizer. When it finished, we saved just the adapter weights and tokenizer into a folder called llama2-ft-qas.




The second script follows almost exactly the same pattern but instead of question–answer pairs we feed in our 156 distilled methodology summaries. We read the summaries Excel and, for each paper, automatically generated about ten prompts like "Tell me about the X method" or "How would you solve hallucination using a Y approach?". Each of those became a training example which was like "Instruction/Response" template and tokenized to 256 tokens. We loaded a 4-bit Llama-2-7B model with a new LoRA adapter and used the identical hyperparameters: batch size 1 + 16 accumulation, learning rate $2\times10^{-5}$, FP16, paged_adamw_8bit and trained for three epochs. The result was saved under llama2-ft-qas, that knows each paper's technical steps in detail but hasn't yet learned the full QA dialogue style for answering inside.

In the third, "sequential" script we fine tuned it on both the datasets together. We reloaded the original Llama-2 base model in 4-bit mode with CPU offloading enabled so it still fit on one GPU, then loaded the methodology-only adapter into it. Next, we read the original QA Excel again, built and tokenized the same 1,560 question–answer prompts to 256 tokens, and pointed

a new Trainer at this combined model. We fine-tuned for four more epochs on the QA data with most settings unchanged—batch size 1 + 16 accumulation, FP16, paged_adamw_8bit but we gently lowered the learning rate to $5\times10^{-6}$ . We saved and zipped the as llama2-ft-qas-cont/, giving us a model that first learned all the raw methodologies in depth and then learned how to present them as clear, user-friendly answers.



# Results

We evaluated three versions of our fine-tuned Llama-2 model on a range of questions about detecting and fixing hallucinations. Below is what we saw, in simple language but with enough detail to understand each model's strengths and weaknesses.

| Dataset | Correct | Incorrect |
|---|---|---|
| Combined | 31 | 19 |
| Description | 25 | 25 |
| QA-only | 28 | 22 |

**Evaluation metrics:**

**Accuracy Formula**
Accuracy = (Correct) / (Correct + Incorrect) × 100%

We tested it with 50 prompts. We used the OpenAIs GPT o4 mini API to check and verify each and every output and see if it was similar to the ground truth.

Using our test results (out of 50 prompts each):

- **Combined (methodology → QA):** 31 correct, 19 incorrect → 31/50 × 100% = **62%**
- **QA-only:** 28 correct, 22 incorrect → 28/50 × 100% = **56%**
- **Description-only:** 25 correct, 25 incorrect → 25/50 × 100% = **50%**

**Analysis of Results**

- **Combined model (62%)**: By first fine-tuning on raw methodology summaries and then continuing on the QA pairs this version learned both the deep technical steps and how to answer direct answers in detail. As a result, it performed best overall especially on prompts asking "How do I go about applying this methodology?" or "What's the step-by-step process?". So it performed the best.
- **QA-only model (56%)**: Trained only on templated question and answer examples, this model was most fluent when handling prompts which were technical. But it sometimes missed broader procedural details that the combined model retained. The average length of the generated texts were also less
- **Description-only model (50%)**: It was only trained on the methodology steps. It knew each paper's steps in depth but hadn't practiced turning them into concise answers. It was good when we asked open ended methodology questions ("Explain the X method"), and gave long explanations. However, it struggled to answer the Q and A format questions crisply. The average length of the generated text were also more.

**Short-answer model**
This version gives you answer in lesser number of tokens. The response is fast and focused these answers were—they cut right to the chase. On the downside, they often skipped over the reasoning behind the suggestion.

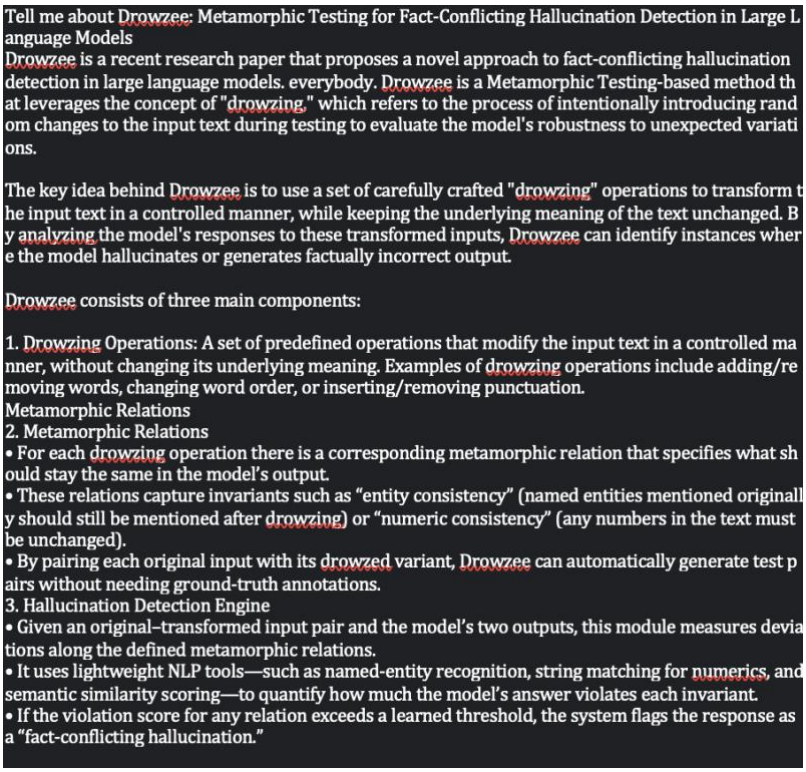
**Speed:** About **7–8 seconds** per response.

Tell me about Drowzee: Metamorphic Testing for Fact-Conflicting Hallucination Detection in Large L anguage Models
Drowzee is a novel approach for detecting fact-conflicting hallucinations in large language models. Fi nzeln, et al. proposed Drowzee at the 2022 Conference on Empirical Methods in Natural Language Pr ocessing (EMNLP). Hallucinations are unexpected or false information generated by models during i nference. Drowzee combines metamorphic testing and adversarial training to improve hallucination detection.

Metamorphic testing involves applying transformations to the input or output of a model to evaluate its robustness. In the context of Drowzee, these transformations are designed to expose the model's sensitivity to factual inconsistencies. Drowzee's adversarial training involves adding noise to the inp ut or output, which encourages the model to generalize to unseen data while minimizing hallucinatio n errors.

**Detailed-answer model**
Here, the model writes a fuller explanation that often spans a few paragraphs. It walks you through multiple steps—for example, how to build the reference corpus, how to encode both your outputs and references, and how to score their similarity—sometimes even mentioning alternative methods like perplexity checks or human review. But when you're in a hurry, having to read several paragraphs to find the main takeaway can feel tedious.

**Speed:** About **17–18 seconds** per response.


Tell me about Drowzee: Metamorphic Testing for Fact-Conflicting Hallucination Detection in Large L anguage Models
Drowzee is a recent research paper that proposes a novel approach to fact-conflicting hallucination detection in large language models. everybody. Drowzee is a Metamorphic Testing-based method th at leverages the concept of "drowzing," which refers to the process of intentionally introducing rand om changes to the input text during testing to evaluate the model's robustness to unexpected variati ons.

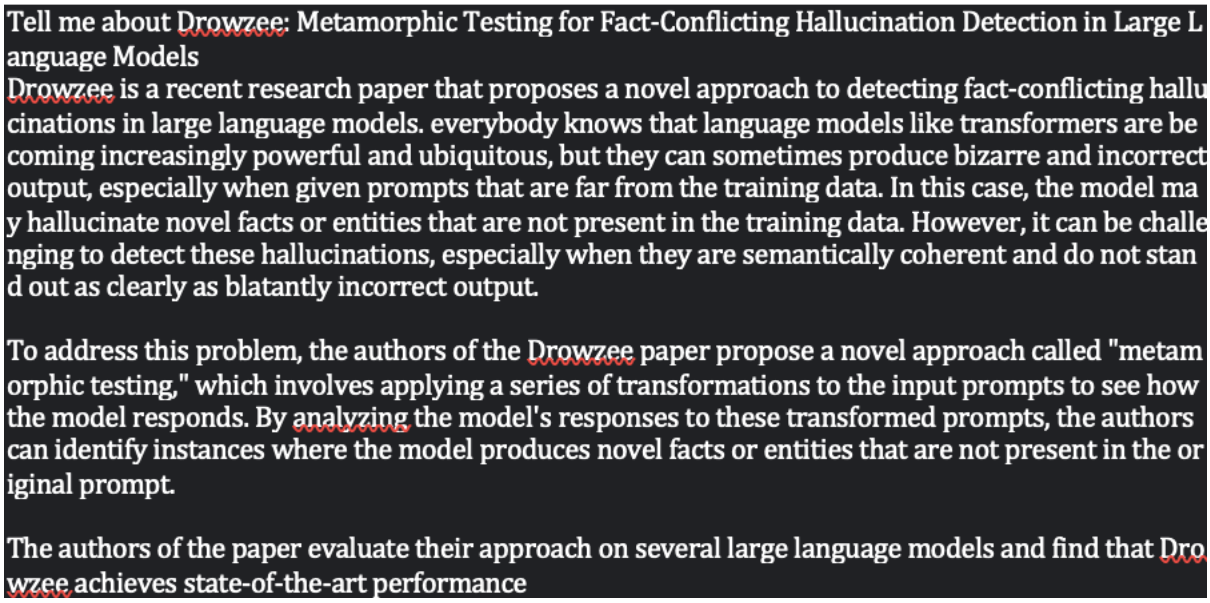
The key idea behind Drowzee is to use a set of carefully crafted "drowzing" operations to transform t he input text in a controlled manner, while keeping the underlying meaning of the text unchanged. B y analyzing the model's responses to these transformed inputs, Drowzee can identify instances wher e the model hallucinates or generates factually incorrect output.

Drowzee consists of three main components:

1. Drowzing Operations: A set of predefined operations that modify the input text in a controlled ma nner, without changing its underlying meaning. Examples of drowzing operations include adding/re moving words, changing word order, or inserting/removing punctuation.
Metamorphic Relations
2. Metamorphic Relations
• For each drowzing operation there is a corresponding metamorphic relation that specifies what sh ould stay the same in the model's output.
• These relations capture invariants such as "entity consistency" (named entities mentioned originall y should still be mentioned after drowzing) or "numeric consistency" (any numbers in the text must be unchanged).
• By pairing each original input with its drowzed variant, Drowzee can automatically generate test p airs without needing ground-truth annotations.
3. Hallucination Detection Engine
• Given an original–transformed input pair and the model's two outputs, this module measures devia tions along the defined metamorphic relations.
• It uses lightweight NLP tools—such as named-entity recognition, string matching for numerics, and semantic similarity scoring—to quantify how much the model's answer violates each invariant.
• If the violation score for any relation exceeds a learned threshold, the system flags the response as a "fact-conflicting hallucination."

**Combined model**: This variant blends both styles. It begins with the quick “one-sentence” solution, then immediately follows with a succinct explanation of why it works and how to do it in practice. For most questions, you get your direct answer first, then just enough background to understand the method without wading through a full academic-style write-up.

**Speed:** Also about **17–18 seconds** per response.




**Overall observations**

- **Speed vs. depth:** The short-answer model wins when you only need a swift answer. The detailed model shines when you want a teach-yourself tutorial and learn everything in detail.

# Your thoughts (e.g., what you learned in this project; what issues you found in this project, etc.)

What we learned?

Trade-offs between speed and depth: It became clear that there is no one “perfect” answer style. Very short responses are fast but can leave readers hanging, whereas very detailed ones teach well but can frustrate users in a hurry. The combined approach struck the best balance, which matches what many real-world assistants need.

We saw firsthand how much easier it is to adapt a large base model when using 4-bit quantization plus LoRA adapters. Instead of re-training all 7 billion parameters (which would have blown our GPU memory), we updated only a small fraction of weights and still got strong task performance.

Issues Faced

The idea of our project was to build a pipeline where when the user asked a prompt to help him solve the problem of LLM hallucination it would give him suggestions on how to go about it. The user can even ask questions related to it and get things clarified. We created 2 datasets to do this. The first dataset had one row per paper with columns for the exact title, the cleaned methodology text, the Dataset which it used and the type of LLM (white box or black box). The second dataset consists of question and answer pair. For each paper we filled in almost 10 questions which gave an initial question that could be asked based on the methodology. This had 1289 high quality examples where the answer was based on the methodology.

The model we chose for fine tuning was Llama-2-7B. We chose this as it was ideal for our experiment as 7 billion parameters is large enough to capture language patterns yet be small enough so that we can fine tune it on a single GPU. The model was loaded with 4-bit precision using BitsAndBytesConfig to reduce the GPU requirements. LoRA adapters  was applied with rank r = 16, alpha = 32, and dropout = 0.05. We also used a few memory saving tricks so we could fine tune on just one GPU. W turned on gradient checkpointing so that the model only stores some of its internal data as it runs and recomputes the rest on the fly. So this will cut down RAM use. We also switched off the model’s generation cache which is used to speed up inference but isn’t needed during training. Instead of updating all seven billion weights we froze the original model parameters and only trained the small LoRA adapter. By combining these steps we kept our GPU memory requirements low enough.

We ran into mismatches between NumPy, PyTorch, bitsandbytes' BitsAndBytesConfig, and the PEFT/LoRA code. Certain combinations would throw import errors or break API calls (e.g. quantization flags not recognized, adapter hooks failing). We solved this by pinning each package to tested versions-PyTorch 2.x, bitsandbytes ,â•0.39, PEFT ,â•0.5.0-and isolating the setup in a fresh virtual environment.