Project 2 - Methodology 2: Hallucination Vector Routing

Notebook Summary

This notebook details the process of building and validating a prompt-risk classifier for hallucination in Llama-3.1-8B, using the hallucination vector (`v_halluc`) constructed previously. The workflow includes: (1) generating a labeled dataset of model answers and hallucination judgments using SQuAD with standard, no-context, and distractor-context scenarios, (2) extracting a single feature for each prompt (the projection of the last prompt token's Layer 16 activation onto `v_halluc`), (3) training a logistic regression classifier to predict hallucination risk based on the aforementioned feature, and (4) evaluating the classifier's performance (AUROC ≥ 0.75). The resulting `risk_clf.joblib` model enables real-time risk scoring for any prompt, forming the core of the project's guardrail system.

Step 2: Turning the Vector into a Prompt-Risk Score

Overall Goal: To build and validate a lightweight logistic regression classifier that takes a prompt's projection onto v_halluc and outputs a calibrated probability of hallucination. The final deliverables will be the trained classifier file (risk_clf.joblib) and a report on its predictive performance (AUROC ≥ 0.75).

Setup and Installation

```
# Setup project directories for local execution
import os
import pathlib

# Use the actual project directory instead of generic home directory
PROJECT_DIR = pathlib.Path("/home/ubuntu/HallucinationVectorProject/"
DATA_DIR = PROJECT_DIR / "data"
ARTIFACTS_DIR = PROJECT_DIR / "artifacts" / "llama-3.1-8b"

# Create necessary directories
DATA_DIR.mkdir(parents=True, exist_ok=True)

ARTIFACTS_DIR.mkdir(parents=True, exist_ok=True)
```

```
print(f"Artifacts directory: {ARTIFACTS_DIR}")

print(f"Data directory: {DATA_DIR}")
print(f"Project directory: {PROJECT_DIR}")
```

```
Artifacts directory: /home/ubuntu/HallucinationVectorProject/artifacts
Data directory: /home/ubuntu/HallucinationVectorProject/data
Project directory: /home/ubuntu/HallucinationVectorProject
```

```
# Load API keys from environment variables
import os
from dotenv import load_dotenv
load_dotenv()  # Load variables from .env file if present

# Load HuggingFace token
HF_TOKEN = os.environ.get("HF_TOKEN", "")
if not HF_TOKEN:
    raise ValueError(
        "HF_TOKEN environment variable is required. "
        "Please set it in your .env file or export it before running
    )

# Load ScaleDown API key
SCALEDOWN_API_KEY = os.environ.get("SCALEDOWN_API_KEY", "")
if not SCALEDOWN_API_KEY:
    raise ValueError(
        "SCALEDOWN_API_KEY environment variable is required. "
        "Please set it in your .env file or export it before running
    )

print("✓ API keys loaded successfully from environment variables")
print(f"✓ HF_TOKEN: {HF_TOKEN[:10]}..." if len(HF_TOKEN) > 10 else "✓
print(f"✓ SCALEDOWN_API_KEY: {SCALEDOWN_API_KEY[:10]}..." if len(SCAL
```

```
✓ API keys loaded successfully from environment variables
✓ HF_TOKEN: hf_NrlndFS...
✓ SCALEDOWN_API_KEY: OMJ5hWc0m4...
```

```
import os, torch
from unsloth import FastLanguageModel

os.environ["UNSLOTH_STABLE_DOWNLOADS"] = "1"

def print_gpu_memory():
    if torch.cuda.is_available():
        allocated = torch.cuda.memory_allocated(0) / 1024**3
        reserved = torch.cuda.memory_reserved(0) / 1024**3
        total = torch.cuda.get_device_properties(0).total_memory / 10
        print(f"  GPU 0: {allocated:.1f}GB allocated, {reserved:.1f}G

HF_TOKEN = os.environ.get("HF_TOKEN")
assert HF_TOKEN, "Set HF_TOKEN in your env first (export HF_TOKEN=...
```

```
print("Initial GPU memory:")
print_gpu_memory()

max_seq_length = 4096
model_name = "unsloth/Meta-Llama-3.1-8B-Instruct"

print(f"Loading {model_name} (bfloat16) on single GPU...")

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name          = model_name,
    max_seq_length      = max_seq_length,
    dtype               = torch.bfloat16,
    load_in_4bit        = False,
    token               = HF_TOKEN,
    trust_remote_code   = True,
)

model = FastLanguageModel.for_inference(model)
model.gradient_checkpointing_disable()
model.config.gradient_checkpointing = False
model.config.use_cache = True
model.eval()

print("✓ Model loaded successfully.")
print(f"  Device: {model.device}")
print(f"  Model dtype: {model.dtype}")
print(f"  Max sequence length: {max_seq_length}")
print("Post-load GPU memory:")
print_gpu_memory()
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuni
🦥 Unsloth Zoo will now patch everything to make training faster!
Initial GPU memory:
  GPU 0: 0.0GB allocated, 0.0GB reserved, 39.5GB total
Loading unsloth/Meta-Llama-3.1-8B-Instruct (bfloat16) on single GPU...
Unsloth: WARNING `trust_remote_code` is True.
Are you certain you want to do remote code execution?
==((====))==  Unsloth 2025.10.9: Fast Llama patching. Transformers: 4.
   \\   /|    NVIDIA A100-PCIE-40GB. Num GPUs = 1. Max memory: 39.495
O^O/ \_/ \    Torch: 2.9.0+cu128. CUDA: 8.0. CUDA Toolkit: 12.8. Trito
\        /    Bfloat16 = TRUE. FA [Xformers = 0.0.33+5d4b92a5.d2025102
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which a
Loading checkpoint shards:   0%|          | 0/4 [00:00<?, ?it/s]
✓ Model loaded successfully.
  Device: cuda:0
  Max sequence length: 4096
Post-load GPU memory:
  GPU 0: 15.0GB allocated, 15.1GB reserved, 39.5GB total
```

Phase 1: Dataset Generation and Labeling

Overall Objective:

To programmatically generate a large, high-quality dataset of approximately 2000 labeled examples. Each example will consist of a prompt, a model-generated answer, and a binary label (1 for hallucination, 0 for correct) determined by a Gemini LLM judge. The final artifact of this phase will be a CSV file, squad_labeled_answers.csv, stored in Google Drive.

Methodology Overview:

We will use the SQuAD dataset as our source of truth. For each sampled entry, we will ask our 4-bit Llama 3 model to answer a question based only on the provided context. A specialized LLM judge will then compare the model's answer to the context and the ground-truth answer to determine if any unsupported information was fabricated (i.e., hallucinated).

This data is needed to train the logistic regression model for calculating hallucination risks for user prompts.

```
# Import libraries
import pandas as pd
from datasets import load_dataset
from tqdm.auto import tqdm
import numpy as np
import os

# FIX FOR HUGGINGFACE DATASETS DOWNLOAD ISSUES
os.environ["HF_HUB_ENABLE_HF_TRANSFER"] = "0"
os.environ["HF_DATASETS_OFFLINE"] = "0"

# --- 1. Load the SQuAD dataset ---
print("Loading SQuAD dataset...")
# SQuAD is a standard dataset that doesn't need trust_remote_code
# Using num_proc=1 to avoid parallel download issues
squad_dataset = load_dataset("squad", split="train", num_proc=1, stre
squad_df = squad_dataset.to_pandas()
print(f"Full dataset loaded with {len(squad_df)} rows.")
```

```
Loading SQuAD dataset...
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call
last)
Cell In[16], line 16
     13 print("Loading SQuAD dataset...")
     14 # SQuAD is a standard dataset that doesn't need
trust_remote_code
     15 # Using num_proc=1 to avoid parallel download issues
---> 16 squad_dataset = load_dataset("squad", split="train",
num_proc=1, streaming=True)
     17 squad_df = squad_dataset.to_pandas()
     18 print(f"Full dataset loaded with {len(squad_df)} rows.")

File ~/HallucinationVectorProject/venv/lib/python3.10/site-
packages/datasets/load.py:1386, in load_dataset(path, name, data_dir,
data_files, split, cache_dir, features, download_config,
download_mode, verification_mode, keep_in_memory, save_infos,
revision, token, streaming, num_proc, storage_options,
**config_kwargs)
   1380     raise ValueError(
   1381         "You are trying to load a dataset that was saved using
`save_to_disk`. "
   1382         "Please use `load_from_disk` instead."
   1383     )
   1385 if streaming and num_proc is not None:
-> 1386     raise NotImplementedError(
   1387         "Loading a streaming dataset in parallel with
`num_proc` is not implemented. "
   1388         "To parallelize streaming, you can wrap the dataset
with a PyTorch DataLoader using `num_workers` > 1 instead."
   1389     )
```

```
import json
from datasets import Dataset

# 1) Download once (outside Python) if you haven't:
# wget https://rajpurkar.github.io/SQuAD-explorer/dataset/train-v1.1.

# 2) Build the exact SQuAD schema the HF loader returns
with open("train.json") as f:
    raw = json.load(f)["data"]

examples = []
for article in raw:
    title = article.get("title", "")
    for para in article["paragraphs"]:
        context = para["context"]
        for qa in para["qas"]:
            answers = qa.get("answers", [])
            examples.append({
                "id": qa["id"],
                "title": title,
                "context": context,
                "question": qa["question"],
                "answers": {
```

```
                    "text": [a["text"] for a in answers],
                    "answer_start": [a["answer_start"] for a in answe
                },
            })

    dataset = Dataset.from_list(examples)
    print(len(dataset), dataset.column_names)  # ~87599, ['id','title','c

    # 3) If you want a pandas DataFrame like before:
    squad_df = dataset.to_pandas()
```

```
87599 ['id', 'title', 'context', 'question', 'answers']
```

```
# --- 2. Sample the data ---
# We'll sample 2000 rows for our experiment
N_SAMPLES = 2000
if len(squad_df) > N_SAMPLES:
    sampled_df = squad_df.sample(n=N_SAMPLES, random_state=42).reset_
else:
    sampled_df = squad_df # Use full dataset if it's smaller

print(f"Using {len(sampled_df)} rows for our experiment.")
```

```
Using 2000 rows for our experiment.
```

Strategy: The "No-Context" and "Distractor-Context" Methods

To elicit varying responses that may or may not contain hallucination, instead of changing the instruction (original dataset prompt) to the model, we will change the information (context) we give it. We will create scenarios where the model is more likely to fail and invent an answer because the correct information is either missing or obscured. This is to elicit a mix of halluicnatory and non-hallucinatory responses to effectively train the logistic regression model.

We generate our ~2,000 examples by creating a mix of three different scenarios for each SQuAD entry.

Scenario 1: Standard In-Context (The "Easy" Case - Generates our 0s)

We pass in the prompt + context exactly as taken from the dataset. We do this for 50% (1000 prompts) of our dataset.

`FULL_PROMPT = f"Context:\n{context}\n\nQuestion:\n{question}"`

Outcome: The model answers correctly most of the time. This is the primary source of our negative examples (label 0).

Scenario 2: No-Context (The "Hard" Case - Designed to induce natural hallucinations)

For a SQuAD entry, we deliberately withhold the context.

`FULL_PROMPT = f"Question:\n{question}"`

Outcome: The model's internal knowledge might contain information about the question's topic, but it may be incorrect, incomplete, or subtly different from the SQuAD context's specific answer. Without the grounding context, it is now much more likely to generate a plausible-sounding but factually incorrect answer. This is a primary source of natural positive examples (label 1).

Scenario 3: Distractor-Context (The "Tricky" Case - Also induces natural hallucinations)

For a SQuAD entry, we provide the correct question but pair it with a distractor context — a paragraph from a different, unrelated SQuAD article.

`FULL_PROMPT = f"Context:\n{distractor_context}\n\nQuestion:\n{question}"`

Outcome: The model is still instructed to answer from the context. However, the answer is not there. A well-behaved model should say "I cannot find the answer in the context." A model prone to hallucination might try to synthesize an answer by blending its own knowledge with irrelevant information from the distractor context. This is another excellent source of natural positive examples (label 1).

```
import numpy as np
# --- 3. Create a 'distractor_context' column ---
# For each row, the distractor is a context from another random row.
# We'll shuffle the context column and assign it.
distractor_indices = np.random.permutation(sampled_df.index)
sampled_df['distractor_context'] = sampled_df.loc[distractor_indices,

print("Dataset prepared with distractor contexts. Sample:")
print(sampled_df[['context', 'question', 'distractor_context']].head(

# --- 4. Assign a scenario to each row ---
# We'll divide our 2000 samples into the three scenarios
scenarios = []
# 1000 for standard, 500 for no-context, 500 for distractor
scenario_counts = {'standard': 1000, 'no_context': 500, 'distractor':

for scenario_type, count in scenario_counts.items():
    scenarios.extend([scenario_type] * count)
```

```
# Make sure the scenarios list matches the dataframe length
sampled_df = sampled_df.iloc[:len(scenarios)].copy()
sampled_df['scenario'] = scenarios

print("\nScenario distribution:")
print(sampled_df['scenario'].value_counts())
```

```
Dataset prepared with distractor contexts. Sample:
                                             context  \
0  The world's first institution of technology or...
1  The standard specifies how speed ratings shoul...
2  The most impressive and famous of Sumerian bui...
3  Ann Arbor has a council-manager form of govern...
4  Shortly before his death, when he was already ...

                                            question  \
0         What year was the Banská Akadémia founded?
1  What is another speed that can also be reporte...
2  Where were the use of advanced materials and t...
3           Who is elected every even numbered year?
4  What was the purpose of top secret ICBM commit...

                                  distractor_context
0  One of the earliest sexual orientation classif...
1  The Obama administration began to reengage in ...
2  In 1430 with the Privileges of Jedlnia, confir...
3  The first vertebrates appeared over 500 millio...
4  In 677 (during the reign of Emperor Gaozong), ...

Scenario distribution:
scenario
standard      1000
no_context     500
distractor     500
Name: count, dtype: int64
```

Define Generation and Judging Logic

Define the functions that will generate an answer based on the scenario and get it judged by Gemini.

```
# --- 1. Define the prompt template and generation function ---
SYSTEM_PROMPT = "You are a helpful assistant. Answer the following qu

def generate_squad_answer_multi_scenario(model, tokenizer, row):
    """
    Generates an answer based on the scenario specified in the DataFr
    """
    scenario = row['scenario']
    question = row['question']

    context_to_use = ""
    if scenario == 'standard':
        context_to_use = row['context']
```

```
    elif scenario == 'distractor':
        context_to_use = row['distractor_context']
    # For 'no_context', context_to_use remains an empty string

    # Construct the user prompt
    if context_to_use:
        user_prompt = f"Context:\n{context_to_use}\n\nQuestion:\n{que
    else:
        user_prompt = f"Question:\n{question}"

    # Format into the Llama 3 chat template
    messages = [
        {"role": "system", "content": SYSTEM_PROMPT},
        {"role": "user", "content": user_prompt}
    ]
    full_prompt_templated = tokenizer.apply_chat_template(messages, t

    # Generate the answer with proper device handling
    inputs = tokenizer(full_prompt_templated, return_tensors="pt", ma
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_new_tokens=128,
            do_sample=False,
            pad_token_id=tokenizer.eos_token_id,
            use_cache=True
        )

    response_text = tokenizer.decode(outputs[0][inputs['input_ids'].s

    # We return the templated prompt as well, as this is what the mod
    return response_text.strip(), full_prompt_templated
```

```
import requests
import json
import time
import os

# --- 1. Define the Judge Prompt Template ---
# This prompt is highly specific to our task and gives the judge all
JUDGE_PROMPT_TEMPLATE = """
You are an impartial judge evaluating a language model's answer for i
Your task is to determine if the model's answer contains any fabricat

Here is the provided context:
<context>
{context}
</context>

Here is the question that was asked:
<question>
{question}
</question>
```

```
Here is the ground-truth answer, which is known to be correct and der
<ground_truth_answer>
{ground_truth_answer}
</ground_truth_answer>

Here is the language model's generated answer:
<model_answer>
{model_answer}
</model_answer>

**Your Task:**
Carefully compare the <model_answer> against the <context>. Does the 
Ignore minor differences in phrasing. If the model says it cannot fin

Respond with only the word 'YES' if the model's answer contains fabri
"""

# --- 2. Implement the Judge Function with retry logic ---
def judge_squad_answer(context, question, model_answer, ground_truth_
    """
    Gets a binary hallucination label from the Gemini API judge with 
    Returns: 1 for hallucination ('YES'), 0 for faithful ('NO'), -1 f
    """
    prompt = JUDGE_PROMPT_TEMPLATE.format(
        context=context,
        question=question,
        model_answer=model_answer,
        ground_truth_answer=ground_truth_answer
    )

    url = "https://api.scaledown.xyz/compress/"
    payload = json.dumps({
        "context": "You are an impartial judge evaluating for halluci
        "prompt": prompt,
        "model": "gpt-4o",
        "scaledown": {
          "rate": 0 # no compression
        }
    })
    headers = {'x-api-key': api_key, 'Content-Type': 'application/jso

    for attempt in range(max_retries):
        try:
            response = requests.post(url, headers=headers, data=paylo
            response.raise_for_status()

            response_data = json.loads(response.text)
            content = response_data.get("full_response", "").strip().

            if 'YES' in content:
                return 1
            elif 'NO' in content:
                return 0
            else:
                print(f"Judge Warning: Unexpected response: {content}
```

```
                return -1

        except (requests.exceptions.RequestException, json.JSONDecode
            print(f"ERROR on attempt {attempt + 1}/{max_retries}: {e}
            if attempt < max_retries - 1:
                wait_time = 2 ** attempt  # Exponential backoff
                print(f"Retrying in {wait_time} seconds...")
                time.sleep(wait_time)
            else:
                print("All retry attempts failed")
                return -1

    return -1
```

Run the generation + judging loop and save results to Drive.

```
import os
import pandas as pd
import time
from tqdm.auto import tqdm

# --- 1. Setup paths and constants ---
OUTPUT_CSV_PATH = ARTIFACTS_DIR / 'squad_labeled_answers_multi_scenar
BATCH_SIZE = 15  # Reduced for 70B model memory management
API_KEY = SCALEDOWN_API_KEY  # Use the loaded environment variable

# NEW: simple counter for rows already on disk
written_count = 0

# Memory monitoring helper
def check_and_clear_memory():
    if torch.cuda.is_available():
        allocated = sum(torch.cuda.memory_allocated(i) for i in range
        if allocated > 60:  # If using more than 60GB across all GPUs
            torch.cuda.empty_cache()
        return allocated
    return 0

# --- 2. Resume logic (robust to partial/dirty last lines) ---
try:
    # Try normal read
    existing_df = pd.read_csv(OUTPUT_CSV_PATH)
except FileNotFoundError:
    existing_df = pd.DataFrame()
except Exception:
    # If the file was cut off mid-write, skip bad lines and continue
    existing_df = pd.read_csv(OUTPUT_CSV_PATH, on_bad_lines='skip')

start_index = len(existing_df)
written_count = start_index
print("Resuming from index {}.".format(start_index) if start_index el

# Helper: append a single row immediately (header only if file doesn'
```

```
def append_row_immediately(row_dict):
    df = pd.DataFrame([row_dict])
    file_exists = os.path.exists(OUTPUT_CSV_PATH)
    write_header = not file_exists or os.path.getsize(OUTPUT_CSV_PATH
    # Appending one row at a time keeps progress durable
    df.to_csv(OUTPUT_CSV_PATH, mode='a', header=write_header, index=F

# --- 3. The Main Loop ---
results_list = []          # kept for batch reporting only
start_time = time.time()

pbar = tqdm(total=len(sampled_df), initial=start_index, desc="Generat
for i in range(start_index, len(sampled_df)):
    row = sampled_df.iloc[i]

    try:
        # Memory check before processing
        memory_usage = check_and_clear_memory()

        # --- Generate ---
        model_answer, full_prompt = generate_squad_answer_multi_scena

        # --- Judge ---
        # The judge ALWAYS compares against the original, correct con
        ground_truth_answer = row['answers']['text'][0] if row['answe
        label = judge_squad_answer(
            context=row['context'],  # Always the original context
            question=row['question'],
            model_answer=model_answer,
            ground_truth_answer=ground_truth_answer,
            api_key=API_KEY
        )

        # Prepare the result record once
        result_record = {
            'scenario': row['scenario'],
            'full_prompt': full_prompt,  # The actual text fed to the
            'model_answer': model_answer,
            'ground_truth_answer': ground_truth_answer,
            'hallucination_label': label,
            'original_context': row['context'],  # Keep for reference
            'question': row['question']
        }

        # --- NEW: Save immediately after each item ---
        append_row_immediately(result_record)
        written_count += 1

        # Keep for batch reporting only
        results_list.append(result_record)

        pbar.update(1)

        # --- Batch progress reporting (no re-writing the whole file)
        if (i + 1) % BATCH_SIZE == 0 or (i + 1) == len(sampled_df):
```

```
                elapsed = time.time() - start_time
                avg_time_per_item = elapsed / max(1, len(results_list))
                remaining = len(sampled_df) - (i + 1)
                eta = avg_time_per_item * remaining if avg_time_per_item :

                print(f"\nSaved through index {i}. Total rows on disk: {w
                print(f"GPU: {memory_usage:.1f}GB | ETA: {eta/60:.1f}min"

                # Print count of 1s and 0s for the current batch only
                temp_df = pd.DataFrame(results_list)
                valid_labels = temp_df[temp_df['hallucination_label'] != 
                if not valid_labels.empty:
                    counts = valid_labels.value_counts().sort_index()
                    print("Current batch label counts:")
                    print(counts)
                else:
                    print("No valid labels in this batch.")

                # Clear batch cache to keep memory low
                results_list = []

        except Exception as e:
            print(f"Error processing row {i}: {e}")
            print("Continuing with next row...")
            continue

    pbar.close()
    total_time = time.time() - start_time
    print(f"Phase 1 complete in {total_time/60:.1f} minutes. Labeled data

    # --- Final check of the class balance ---
    # Be tolerant to any trailing partials
    final_df = pd.read_csv(OUTPUT_CSV_PATH, on_bad_lines='skip')
    print("\nFinal Class Balance:")
    print(final_df[final_df['hallucination_label'] != -1]['hallucination_
```

```
Starting from scratch.
Generating & Judging:   0%|          | 0/2000 [00:00<?, ?it/s]

Saved through index 14. Total rows on disk: 15
GPU: 15.1GB | ETA: 56.2min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 29. Total rows on disk: 30
GPU: 15.1GB | ETA: 108.7min
Current batch label counts:
hallucination_label
0    14
1     1
Name: count, dtype: int64

Saved through index 44. Total rows on disk: 45
GPU: 15.1GB | ETA: 149.2min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 59. Total rows on disk: 60
GPU: 15.1GB | ETA: 194.1min
Current batch label counts:
hallucination_label
0    14
1     1
Name: count, dtype: int64

Saved through index 74. Total rows on disk: 75
GPU: 15.1GB | ETA: 236.0min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 89. Total rows on disk: 90
GPU: 15.1GB | ETA: 278.5min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 104. Total rows on disk: 105
GPU: 15.1GB | ETA: 331.5min
Current batch label counts:
hallucination_label
0    13
1     2
Name: count, dtype: int64

Saved through index 119. Total rows on disk: 120
GPU: 15.1GB | ETA: 382.6min
Current batch label counts:
hallucination_label
0    13
```

```
1    2
Name: count, dtype: int64

Saved through index 134. Total rows on disk: 135
GPU: 15.1GB | ETA: 442.8min
Current batch label counts:
hallucination_label
0    13
1    2
Name: count, dtype: int64

Saved through index 149. Total rows on disk: 150
GPU: 15.1GB | ETA: 477.8min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 164. Total rows on disk: 165
GPU: 15.1GB | ETA: 508.8min
Current batch label counts:
hallucination_label
0    14
1    1
Name: count, dtype: int64

Saved through index 179. Total rows on disk: 180
GPU: 15.1GB | ETA: 543.2min
Current batch label counts:
hallucination_label
0    14
```

Phase 2: Feature Calculation

Overall Objective:

To process our squad_labeled_answers_multi_scenario.csv file and add a new column, z_feature, to it. This feature is the dot product of the last prompt token's Layer 16 activation with our v_halluc vector. The final artifact will be an updated CSV file, ready for training our classifier in the next phase.

Methodology Overview:

We load our pre-computed hallucination vector and the labeled dataset. Then, in a resilient, batched loop, we feed each prompt from the dataset into the Llama 3 model, extract the specific activation vector we need, compute the projection, and save the results incrementally.

Load the Hallucination Vector and Labeled Data

Load the necessary artifacts (v_halluc.pt and the CSV from Phase 1) into our Colab environment.

```python
import pandas as pd
import numpy as np
import torch
from tqdm.auto import tqdm

# --- 1. Define paths ---
VECTOR_PATH = ARTIFACTS_DIR / 'v_halluc.pt'
LABELED_DATA_PATH = ARTIFACTS_DIR / 'squad_labeled_answers_multi_scen
OUTPUT_PATH = ARTIFACTS_DIR / 'squad_data_with_features.csv'

# --- 2. Load the hallucination vector ---
v_halluc = torch.load(VECTOR_PATH, map_location='cpu').to(model.devic
print(f"Hallucination vector loaded successfully. Shape: {v_halluc.sh

# --- 3. Load and clean the labeled dataset ---
labeled_df = pd.read_csv(LABELED_DATA_PATH)
print(f"Loaded {len(labeled_df)} labeled examples.")

# Filter out rows where the judge failed (label == -1) and any rows w
initial_rows = len(labeled_df)
labeled_df = labeled_df[labeled_df['hallucination_label'] != -1].drop
labeled_df['hallucination_label'] = labeled_df['hallucination_label']
final_rows = len(labeled_df)

print(f"Filtered out {initial_rows - final_rows} rows with invalid la
print(f"Proceeding with {final_rows} valid examples.")
```

```
print("Cleaned DataFrame sample:")
print(labeled_df.head())
```

```
Hallucination vector loaded successfully. Shape: torch.Size([4096]), D
Loaded 2000 labeled examples.
Filtered out 0 rows with invalid labels.
Proceeding with 2000 valid examples.
Cleaned DataFrame sample:
  scenario                                        full_prompt  \
0  standard  <|begin_of_text|><|start_header_id|>system<|en...
1  standard  <|begin_of_text|><|start_header_id|>system<|en...
2  standard  <|begin_of_text|><|start_header_id|>system<|en...
3  standard  <|begin_of_text|><|start_header_id|>system<|en...
4  standard  <|begin_of_text|><|start_header_id|>system<|en...

                                        model_answer  \
0        The Banská Akadémia was founded in 1735.
1  The camera may also report the SOS-based speed...
2                      Sumerian temples and palaces.
3     The mayor is elected every even-numbered year.
4  The purpose of the top secret ICBM committee, ...

                                 ground_truth_answer  hallucination_la
0                                               1735
1                                    SOS-based speed
2                       Sumerian temples and palaces
3                                              mayor
4  decide on the feasibility of building an ICBM ...

                                    original_context  \
0  The world's first institution of technology or...
1  The standard specifies how speed ratings shoul...
2  The most impressive and famous of Sumerian bui...
3  Ann Arbor has a council-manager form of govern...
4  Shortly before his death, when he was already ...

                                            question
0         What year was the Banská Akadémia founded?
1  What is another speed that can also be reporte...
2  Where were the use of advanced materials and t...
3           Who is elected every even numbered year?
4  What was the purpose of top secret ICBM commit...
```

```
Saved through index 254. Total rows on disk: 255
GPU: 15.1GB | ETA: 750.0min
Current batch label counts:
hallucination_label
0    14
1     1
Name: count, dtype: int64
Saved through index 269. Total rows on disk: 270
GPU: 15.1GB | ETA: 780.4min
Current batch label counts:
hallucination_label
0    13
1     2
Name: count, dtype: int64
Saved through index 284. Total rows on disk: 285
GPU: 15.1GB | ETA: 816.7min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64
Saved through index 299. Total rows on disk: 300
GPU: 15.1GB | ETA: 847.9min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64
Saved through index 314. Total rows on disk: 315
GPU: 15.1GB | ETA: 877.4min
Current batch label counts:
hallucination_label
0    14
1     1
Name: count, dtype: int64
Saved through index 329. Total rows on disk: 330
GPU: 15.1GB | ETA: [illegible]
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64
Saved through index 344. Total rows on disk: 345
GPU: 15.1GB | ETA: 937.0min
Current batch label counts:
hallucination_label
```

Implement the Last-Token Activation Extraction Function

Create a function that takes a prompt, runs it through the model, and returns the specific hidden-state vector of the last token of that prompt at Layer 16.

```
TARGET_LAYER = 16

def get_last_prompt_token_activation(model, tokenizer, prompt_text):
    """
    Runs the model on the prompt and extracts the hidden state of the
    last prompt token at the target layer with memory optimization.
    """
```

```
    try:
        # Tokenize the prompt with truncation for safety
        inputs = tokenizer(prompt_text, return_tensors="pt", truncati

        # Perform a forward pass to get hidden states
        # We don't need to generate text, just get the activations
        with torch.no_grad():
            outputs = model(**inputs, output_hidden_states=True)

        # Extract the hidden states for our target layer
        # The shape is (batch_size, sequence_length, hidden_dim)
        hidden_states = outputs.hidden_states[TARGET_LAYER]

        # The last token's activation is at the final sequence positi
        # Squeeze() removes the batch dimension of 1
        last_token_activation = hidden_states[0, -1, :].squeeze()

        # Clean up to free memory
        del outputs, hidden_states

        return last_token_activation

    except Exception as e:
        print(f"Error in activation extraction: {e}")
        # Return zero vector on error
        return torch.zeros(model.config.hidden_size, dtype=torch.bflo

# --- Test the function with one example ---
example_prompt = labeled_df.iloc[0]['full_prompt']
activation_vector = get_last_prompt_token_activation(model, tokenizer
print(f"Successfully extracted activation for one prompt.")
print(f"Activation vector shape: {activation_vector.shape}")
print(f"Activation vector dtype: {activation_vector.dtype}")
```

```
hallucination_label
0    15
Name: count, dtype: int64
Successfully extracted activation for one prompt.
Activation vector shape: torch.Size([4096])
Activation vector dtype: torch.bfloat16
Saved through index 449. Total rows on disk: 450
GPU: 15.1GB | ETA: 1122.4min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64
Saved through index 464. Total rows on disk: 465
GPU: 15.1GB | ETA: 1142.0min
```

The Main Processing Loop (Extract, Project, Save)

Iterate through our entire dataset, compute the z feature for each prompt, and save the results incrementally.

```
import os
import pandas as pd
from tqdm.auto import tqdm
import torch
import numpy as np
import time

# --- 1. Setup for the loop ---
BATCH_SIZE = 30  # Reduced for 70B model memory management
```

```
    MEMORY_CLEANUP_INTERVAL = 10  # Clear memory every 10 extractions

    # --- 2. Check for existing progress to resume ---
    try:
        df_to_process = pd.read_csv(OUTPUT_PATH)
        start_index = len(df_to_process)
        # Ensure the z_feature column exists if we are resuming an older
        if 'z_feature' not in df_to_process.columns:
            df_to_process['z_feature'] = pd.NA
        print(f"Resuming feature calculation from index {start_index}.")
    except FileNotFoundError:
        start_index = 0
        df_to_process = labeled_df.copy() # Use the DataFrame we loaded i

        # Initialize z_feature column for new run
        print("Initializing 'z_feature' column for new run.")
        df_to_process['z_feature'] = pd.NA
        print("Starting feature calculation from scratch.")

    # --- 3. The Main Loop with memory management ---
    start_time = time.time()
    pbar = tqdm(total=len(df_to_process), initial=start_index, desc="Calc

    for i in range(start_index, len(df_to_process)):
        # Check if the feature has already been computed in a previous ru
        if pd.notna(df_to_process.loc[i, 'z_feature']):
            pbar.update(1)
            continue

        try:
            row = df_to_process.iloc[i]
            prompt = row['full_prompt']

            # --- Extract Activation ---
            last_token_activation = get_last_prompt_token_activation(mode

            # --- Compute Projection (Dot Product) ---
            # Ensure both vectors are on the same device and have the sam
            projection = torch.dot(last_token_activation.to(v_halluc.devi
            z_feature = projection.item() # .item() gets the scalar value

            # Store the feature in the DataFrame
            df_to_process.loc[i, 'z_feature'] = z_feature

            pbar.update(1)

            # Periodic memory cleanup for large model
            if (i + 1) % MEMORY_CLEANUP_INTERVAL == 0:
                if torch.cuda.is_available():
                    torch.cuda.empty_cache()

            # --- Save progress in batches ---
            if (i + 1) % BATCH_SIZE == 0 or (i + 1) == len(df_to_process)
                df_to_process.to_csv(OUTPUT_PATH, index=False)
```

```
            # Progress reporting with time estimates
            elapsed = time.time() - start_time
            avg_time_per_item = elapsed / ((i + 1) - start_index) if
            remaining = len(df_to_process) - (i + 1)
            eta = avg_time_per_item * remaining if avg_time_per_item :

            pbar.set_description(f"Saved batch {(i // BATCH_SIZE) + 1

    except Exception as e:
        print(f"Error processing row {i}: {e}")
        df_to_process.loc[i, 'z_feature'] = pd.NA
        continue

# Final save to ensure the last batch is written
df_to_process.to_csv(OUTPUT_PATH, index=False)
pbar.close()

# Final cleanup
if torch.cuda.is_available():
    torch.cuda.empty_cache()

total_time = time.time() - start_time
print(f"Phase 2 complete in {total_time/60:.1f} minutes. Final datase

# --- Final check of the output ---
final_df = pd.read_csv(OUTPUT_PATH)
print("\nFinal DataFrame with 'z_feature' column:")
print(final_df[['full_prompt', 'hallucination_label', 'z_feature']].h
print(f"\nDescription of z_feature:\n{final_df['z_feature'].describe(
```

```
0    13
1     2
Name: count, dtype: int64

Saved through index 689. Total rows on disk: 690
GPU: 15.1GB | ETA: 1437.1min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 704. Total rows on disk: 705
GPU: 15.1GB | ETA: 1450.8min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 719. Total rows on disk: 720
GPU: 15.1GB | ETA: 1465.0min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 734. Total rows on disk: 735
GPU: 15.1GB | ETA: 1479.3min
Current batch label counts:
```

```
Initializing 'z_feature' column for new run.
Starting feature calculation from scratch.
Calculating z-features:   0%|          | 0/2000 [00:00<?, ?it/s]
Phase 2 complete in 1.5 minutes. Final dataset saved to: /home/ubuntu/

Final DataFrame with 'z_feature' column:
                                         full_prompt  hallucination_la
0  <|begin_of_text|><|start_header_id|>system<|en...
1  <|begin_of_text|><|start_header_id|>system<|en...
2  <|begin_of_text|><|start_header_id|>system<|en...
3  <|begin_of_text|><|start_header_id|>system<|en...
4  <|begin_of_text|><|start_header_id|>system<|en...

   z_feature
0    3.12500
1    2.93750
2    4.93750
3    4.00000
4    5.15625

Description of z_feature:
count    2000.000000
mean        0.863298
std         4.361434
min       -10.937500
25%        -1.984375
50%         2.890625
75%         4.093750
max         7.156250
Name: z_feature, dtype: float64
```

```
hallucination_label
0    13
1     2
Name: count, dtype: int64

Saved through index 749. Total rows on disk: 750
GPU: 15.1GB | ETA: 1493.3min
Current batch label counts:
hallucination_label
0    14
1     1
Name: count, dtype: int64

Saved through index 764. Total rows on disk: 765
GPU: 15.1GB | ETA: 1507.0min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 779. Total rows on disk: 780
GPU: 15.1GB | ETA: 1517.7min
Current batch label counts:
hallucination_label
0    15
Name: count, dtype: int64

Saved through index 794. Total rows on disk: 795
GPU: 15.1GB | ETA: 1529.6min
Current batch label counts:
hallucination_label
0    14
1     1
Name: count, dtype: int64

Saved through index 809. Total rows on disk: 810
GPU: 15.1GB | ETA: 1539.9min
Current batch label counts:
hallucination_label
0    12
1     3
Name: count, dtype: int64

Saved through index 824. Total rows on disk: 825
GPU: 15.1GB | ETA: 1551.7min
Current batch label counts:
hallucination_label
0    12
1     3
Name: count, dtype: int64
```

Phase 3: Training and Validating the Risk Scorer

Objective: To use our (z_feature, hallucination_label) dataset to train a simple, fast classifier and rigorously evaluate its predictive power.

Load Data and Create a Train-Test Split

Prepare our data for supervised learning by splitting it into a training set for the model to learn from and a held-out test set to evaluate its performance on unseen data.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve, classification_
import matplotlib.pyplot as plt
import joblib

# --- 1. Load the dataset with features ---
FEATURES_DATA_PATH = ARTIFACTS_DIR / 'squad_data_with_features.csv'
```

```
df_final = pd.read_csv(FEATURES_DATA_PATH)

print("Loaded dataset with features. Sample:")
print(df_final[['z_feature', 'hallucination_label']].head())

# --- 2. Define Features (X) and Labels (y) ---
# Our feature X is the 'z_feature' column
X = df_final[['z_feature']]
# Our target y is the 'hallucination_label' column
y = df_final['hallucination_label']

# --- 3. Create the Train-Test Split ---
# We'll use a standard 80/20 split.
# 'stratify=y' ensures the proportion of 0s and 1s is the same in bot
# 'random_state=42' makes our split reproducible.
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)

print(f"\nData split into training and testing sets:")
print(f"Training set size: {len(X_train)} samples")
print(f"Test set size: {len(X_test)} samples")
print(f"Hallucination proportion in training set: {y_train.mean():.2%
print(f"Hallucination proportion in test set: {y_test.mean():.2%}")
```

```
INFO:matplotlib.font_manager: generated new fontManager
Loaded dataset with features. Sample:
   z_feature  hallucination_label
0    3.12500                    0
1    2.93750                    0
2    4.93750                    0
3    4.00000                    0
4    5.15625                    0

Data split into training and testing sets:
Training set size: 1600 samples
Test set size: 400 samples
Hallucination proportion in training set: 36.12%
Hallucination proportion in test set: 36.25%
```

```
hallucination_label
0    14
1    1
Name: count, dtype: int64
Saved through index 929. Total rows on disk: 930
GPU: 15.1GB | ETA: 1578.1min
Current batch label counts:
hallucination_label
0    14
1    1
Name: count, dtype: int64
Saved through index 944. Total rows on disk: 945
GPU: 15.1GB | ETA: 1580.6min
Current batch label counts:
hallucination_label
0    14
1    1
Name: count, dtype: int64
Saved through index 959. Total rows on disk: 960
GPU: 15.1GB | ETA: 1579.8min
Current batch label counts:
hallucination_label
```

Fit the Logistic Regression Model

Train the logistic regression classifier on our training data and save the resulting model.

```
# --- 1. Initialize and train the model ---
print("Training the logistic regression model...")
risk_classifier = LogisticRegression(random_state=42)
risk_classifier.fit(X_train, y_train)
```

```
print("Training complete.")

# --- 2. Inspect the learned coefficients (optional but insightful) -
intercept = risk_classifier.intercept_[0]
coefficient = risk_classifier.coef_[0][0]
print(f"Learned Intercept (β₀): {intercept:.4f}")
print(f"Learned Coefficient (β₁): {coefficient:.4f}")
# A positive coefficient means that a higher z_feature value correspo

# --- 3. Save the trained model for later use ---
CLASSIFIER_PATH = ARTIFACTS_DIR / 'risk_clf.joblib'
joblib.dump(risk_classifier, CLASSIFIER_PATH)
print(f"Classifier saved to: {CLASSIFIER_PATH}")
```

```
Training the logistic regression model...
Training complete.
Learned Intercept (β₀): -0.3653
Learned Coefficient (β₁): -0.3187
Classifier saved to: /home/ubuntu/HallucinationVectorProject/artifacts
```

```
0    11
1     4
Name: count, dtype: int64
Saved through index 1019. Total rows on disk: 1020
current batch label counts:
hallucination_label
0    6
1    9
Name: count, dtype: int64
Saved through index 1034. Total rows on disk: 1035
GPU: 15.1GB | ETA: 1601.5min
Current batch label counts:
```

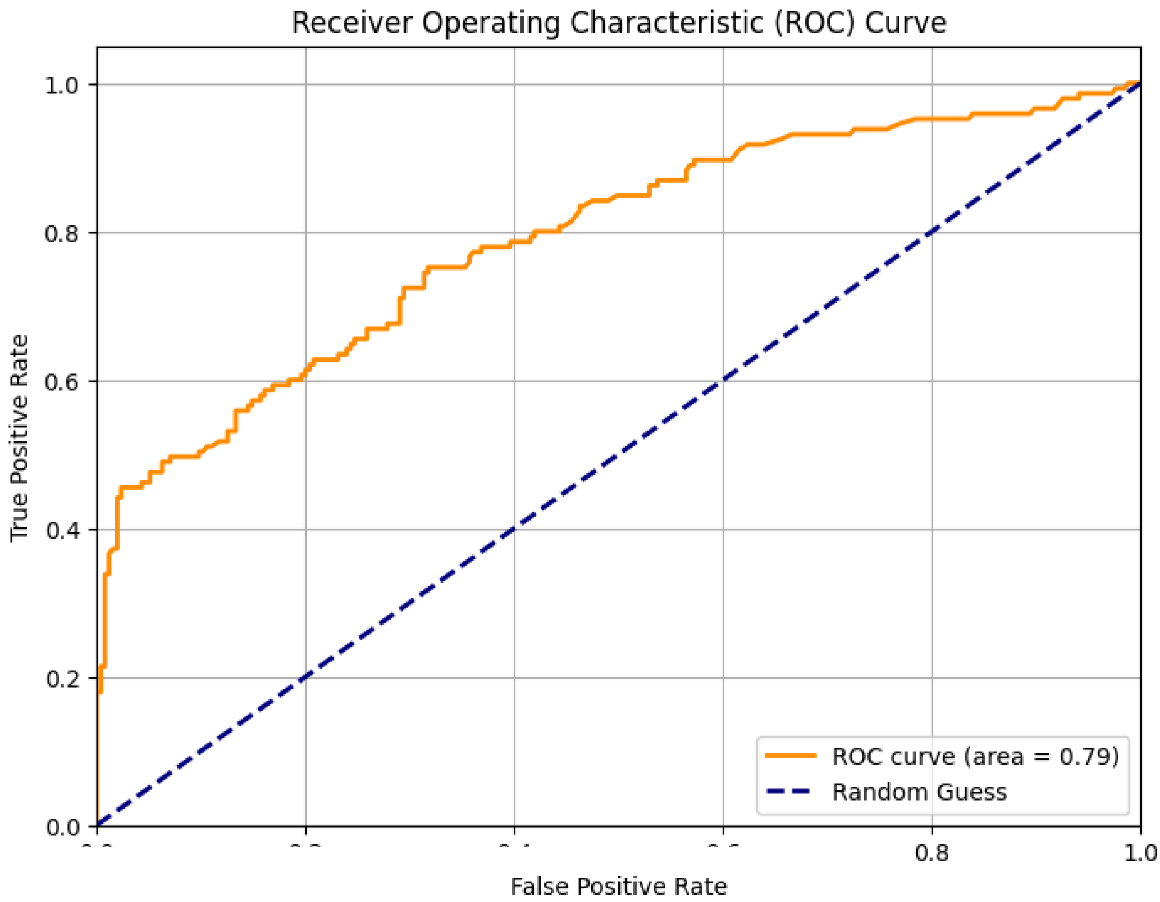
Evaluate Performance on the Test Set

Test our classifier on unseen data and verify that it meets our AUROC ≥ 0.75 success criterion.

```
# --- 1. Predict probabilities on the test set ---
# We need the probability of the positive class (hallucination, which
y_pred_proba = risk_classifier.predict_proba(X_test)[:, 1]

# --- 2. Calculate and validate the AUROC score ---
auroc_score = roc_auc_score(y_test, y_pred_proba)
print(f"\n--- Performance Evaluation ---")
print(f"AUROC Score on Test Set: {auroc_score:.4f}")

if auroc_score >= 0.75:
    print("✅ Success! AUROC meets or exceeds the target of 0.75.")
else:
    print("⚠️ Warning: AUROC is below the target of 0.75.")

# --- 3. Plot the ROC Curve for our report ---
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='R
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
```

```
plt.show()

# --- 4. Generate a detailed classification report ---
# This shows precision and recall at a default 0.5 probability thresh
y_pred_binary = (y_pred_proba >= 0.5).astype(int)
print("\nClassification Report (at 0.5 threshold):")
print(classification_report(y_test, y_pred_binary, target_names=['Fai
```

```
--- Performance Evaluation ---
AUROC Score on Test Set: 0.7903
✅ Success! AUROC meets or exceeds the target of 0.75.
```




```
Classification Report (at 0.5 threshold):
                 precision    recall  f1-score   support

  Faithful (0)       0.78      0.87      0.82       255
Hallucination (1)    0.70      0.56      0.62       145

      accuracy                           0.76       400
     macro avg       0.74      0.71      0.72       400
  weighted avg       0.75      0.76      0.75       400
```

```
GPU: 15.1GB | ETA: 1626.6min
Current batch label counts:
hallucination_label
0    6
1    9
Saved through index 1169. Total rows on disk: 1170
GPU: 15.1GB | ETA: 1631.3min
Current batch label counts:
hallucination_label
0    8
1    7
Name: count, dtype: int64

Saved through index 1184. Total rows on disk: 1185
GPU: 15.1GB | ETA: 1628.6min
Current batch label counts:
hallucination_label
0    9
1    6
Name: count, dtype: int64

Saved through index 1199. Total rows on disk: 1200
GPU: 15.1GB | ETA: 1630.2min
Current batch label counts:
hallucination_label
0    7
1    8
```

```
Name: count, dtype: int64
Saved through index 1214. Total rows on disk: 1215
GPU: 15.1GB | ETA: 1628.5min
Current batch label counts:
hallucination_label
0    9
1    6
Name: count, dtype: int64
```

Phase 4: Create the Real-Time Risk Function

Objective: To encapsulate our entire prediction pipeline into a single, easy-to-use function that will be the core of our Step 3 guardrail.

Essentially, we apply our logistic regression model on the layer-16 last token activation's projection of user prompts on persona vector (z-score) to get the "risk

```
import joblib

# --- 1. Load all necessary artifacts once ---
# This is more efficient than loading them inside the function every 
VECTOR_PATH = ARTIFACTS_DIR / 'v_halluc.pt'
CLASSIFIER_PATH = ARTIFACTS_DIR / 'risk_clf.joblib'

v_halluc_loaded = torch.load(VECTOR_PATH, map_location='cpu').to(mode
risk_classifier_loaded = joblib.load(CLASSIFIER_PATH)
print("Loaded vector and classifier for real-time function.")

# --- 2. Define the final, real-time risk function ---
def get_hallucination_risk(prompt_text, model, tokenizer, v_halluc, c
    """
    Takes a raw prompt and returns a calibrated hallucination risk sc
    """
    try:
        # Step 1: Extract the last-token activation at Layer 16
```