Combined Dynamic Alpha + Selective N-Tokens Steering: Hallucination Guardrail Evaluation

Summary: This notebook evaluates a combined guardrail approach for Llama-3.1-8B that integrates both Dynamic Alpha (risk-proportional steering strength) and Selective N-Tokens Steering (applying the intervention only to the first N tokens). This method applies a dynamic, risk-scaled correction for high-risk prompts, but only during the initial generation steps, maximizing hallucination reduction while minimizing latency and preserving answer quality. This combined approach outperforms both individual ablations and is used for all further evaluations on other datasets.

- **Dynamic Alpha:** Steering strength (alpha) is scaled based on prompt risk, providing stronger correction for riskier prompts.
- **Selective N-Tokens:** Steering is applied only to the first 10 generated tokens, focusing intervention where it is most effective.

Key Results (TruthfulQA Benchmark):

- **Baseline Model:** Accuracy: 38.57%, Hallucination Rate: 61.43%, Avg Latency: 3.86s
- **Combined Guarded Model:** Accuracy: 52.04%, Hallucination Rate: 47.96%, Avg Latency: 3.56s
- **Relative Error Reduction:** 21.93%
- **Latency Increase:** -7.78% (latency decreased)

This combined guardrail achieves the best trade-off between hallucination reduction, accuracy, and latency, and is therefore used for all subsequent cross-domain evaluations.

Environment and Requirements Setup

Unzips the project folder and installs all required Python packages for Colab execution.

Project Path and Colab Configuration

Adds the project directory to Python's path and sets the environment to 'colab' in the config file for compatibility.

```
import sys
import os

# Add project directory to Python's path
project_path = '/content/HallucinationVectorProject'
if project_path not in sys.path:
    sys.path.append(project_path)

# Programmatically set the environment to 'colab' in the config file
config_file_path = os.path.join(project_path, 'config.py')
with open(config_file_path, 'r') as f:
    lines = f.readlines()
with open(config_file_path, 'w') as f:
    for line in lines:
        if line.strip().startswith('ENVIRONMENT ='):
            f.write('ENVIRONMENT = "colab"\n')
        else:
            f.write(line)
print("Environment configured for Colab execution.")
```

```
Environment configured for Colab execution.
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Create a project directory to keep things organized
import os
PROJECT_DIR = "/content/drive/MyDrive/mistral-HallucinationVectorProj
DATA_DIR = os.path.join(PROJECT_DIR, "data")
os.makedirs(DATA_DIR, exist_ok=True)

print(f"Project directory created at: {PROJECT_DIR}")
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Load Artifacts and Model Setup

Loads all required model artifacts, including the hallucination vector, risk classifier, and config thresholds, and prepares the model for evaluation.

```
import time
import pandas as pd
import torch
import csv
from tqdm import tqdm
import joblib
```

```
from unsloth import FastLanguageModel

# Import custom modules
import config
import utils

# This global dictionary will hold our models, tokenizer, vectors, et
artifacts = {}

def load_all_artifacts():
    """Loads all necessary model and project artifacts into the globa
    if artifacts: return
    print("Loading all necessary artifacts for evaluation...")
    # Load the model from Hugging Face
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    )

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

    artifacts['model'] = model
    artifacts['tokenizer'] = tokenizer
    artifacts['v_halluc'] = torch.load("/content/drive/MyDrive/mistra
    artifacts['risk_classifier'] = joblib.load("/content/Hallucinatio
    artifacts['thresholds'] = {
        "tau_low": config.TAU_LOW,
        "tau_high": config.TAU_HIGH,
        "optimal_alpha": config.OPTIMAL_ALPHA
    }

# Load everything
load_all_artifacts()
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuni
🦥 Unsloth Zoo will now patch everything to make training faster!
Loading all necessary artifacts for evaluation...
Loading model and tokenizer: unsloth/llama-3-8b-Instruct-bnb-4bit
==((====))==  Unsloth 2025.9.7: Fast Llama patching. Transformers: 4.5
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform:
O^O/ \_/ \    Torch: 2.8.0+cu126. CUDA: 7.5. CUDA Toolkit: 12.6. Trito
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.32.post2. FA2 = Fal
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which a
model.safetensors: 100%                         5.70G/5.70G [00:40<00:00, 197MB/s]

generation_config.json: 100%                    220/220 [00:00<00:00, 22.8kB/s]

tokenizer_config.json:      51.1k/? [00:00<00:00, 5.25MB/s]

tokenizer.json:      9.09M/? [00:00<00:00, 88.8MB/s]

special_tokens_map.json: 100%                   345/345 [00:00<00:00, 24.5kB/s]
Model and tokenizer loaded successfully.
```

Combined Selective Activation Steering and Guardrail Function

Defines a context manager to apply the steering vector only to the first N tokens, with dynamic risk-proportional strength, and a function to generate answers using this combined intervention.

```
import time
import torch
from contextlib import contextmanager

# Import our project's config and utils modules
import config
import utils

print("Defining combined logic for 'Dynamic Alpha + Selective Steerin

# --- 1. The SelectiveActivationSteerer Class ---

class SelectiveActivationSteerer:
    def __init__(self, model, steering_vector, layer_idx, coeff=1.0,
        self.model = model
        self.vector = steering_vector
        self.layer_idx = layer_idx
        self.coeff = coeff
        self.steering_token_limit = steering_token_limit
        self._handle = None
        self._layer_path = f"model.layers.{self.layer_idx}"
        self.call_count = 0

    def _hook_fn(self, module, ins, out):
        self.call_count += 1
        if self.call_count <= self.steering_token_limit:
            steered_output = out[0] + (self.coeff * self.vector.to(ou
            return (steered_output,) + out[1:]
        return out

    def __enter__(self):
        self.call_count = 0
        try:
            layer = self.model.get_submodule(self._layer_path)
            self._handle = layer.register_forward_hook(self._hook_fn)
        except AttributeError:
            raise AttributeError(f"Could not find the layer at path:
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self._handle:
            self._handle.remove()


# --- 2. The New `answer_guarded_combined` Function ---
# This function combines the logic from both successful ablations.

def answer_guarded_combined(prompt_text: str, max_new_tokens: int = 1
    """
    Generates a response using the guardrail with DYNAMIC alpha and S
    """
    start_time = time.time()

    risk_score = utils.get_hallucination_risk(
        prompt_text, artifacts['model'], artifacts['tokenizer'],
        artifacts['v_halluc'], artifacts['risk_classifier']
    )

    full_prompt = f"<|begin_of_text|><|start_header_id|>system<|end_h
    inputs = artifacts['tokenizer'](full_prompt, return_tensors="pt")
    input_token_length = inputs.input_ids.shape[1]

    if risk_score < artifacts['thresholds']['tau_high']:
        path = "Fast Path (Untouched)"
        with torch.no_grad():
            outputs = artifacts['model'].generate(**inputs, max_new_t
    else:
        # From Dynamic Alpha Ablation: Calculate dynamic steering str
        optimal_alpha = artifacts['thresholds']['optimal_alpha']
        tau_high = artifacts['thresholds']['tau_high']
        scaling_factor = (risk_score - tau_high) / (1.0 - tau_high +
        dynamic_alpha = optimal_alpha * max(0, min(1, scaling_factor)

        path = f"Combined Steer Path (α={dynamic_alpha:.2f}, N={steer

        # From Selective N-Tokens Ablation: Use the steerer with a to
        # We pass our newly calculated `dynamic_alpha` as the coeffic
        with SelectiveActivationSteerer(
            artifacts['model'], artifacts['v_halluc'], config.TARGET_
            coeff=dynamic_alpha,
            steering_token_limit=steering_token_limit
        ):
            with torch.no_grad():
                outputs = artifacts['model'].generate(**inputs, max_n

    answer = artifacts['tokenizer'].decode(outputs[0, input_token_len
    latency = time.time() - start_time

    return {"answer": answer.strip(), "risk_score": risk_score, "path_

print("New function `answer_guarded_combined` is now defined and ready
```

```
Defining combined logic for 'Dynamic Alpha + Selective Steering' exper
New function `answer_guarded_combined` is now defined and ready for th
```

Suppress Warnings Suppresses specific sklearn warnings for cleaner output during evaluation.

```
import warnings

warnings.filterwarnings(
    "ignore",
    message="X does not have valid feature names",
    category=UserWarning,
    module="sklearn"
)
```

Run Combined Guardrail Evaluation

Runs the evaluation loop on the TruthfulQA test set, applying the combined guardrail and saving results for each prompt.

```
# --- EXPERIMENT PARAMETER ---
STEERING_TOKEN_LIMIT = 10 # The 'N' for our selective steering

GUARDED_RESULTS_PATH_COMBINED = os.path.join("/content/HallucinationV
BASELINE_RESULTS_PATH = os.path.join("/content/HallucinationVectorPro

print(f"New guarded results will be saved to: {GUARDED_RESULTS_PATH_C

# Load the test set
test_df = pd.read_csv("/content/HallucinationVectorProject/data/final_

# --- Resilient Evaluation Loop ---
guarded_headers = ['prompt', 'answer', 'risk_score', 'path_taken', 'l
utils.initialize_csv(GUARDED_RESULTS_PATH_COMBINED, guarded_headers)

processed_guarded = utils.load_processed_prompts(GUARDED_RESULTS_PATH_

print("Starting response generation for baseline and DYNAMIC guarded 
for _, row in tqdm(test_df.iterrows(), total=len(test_df), desc="Dyna
        prompt = row['Question']

        # Guarded Run
        if prompt not in processed_guarded:
            try:
                result = answer_guarded_combined(prompt, steering_token_l
                with open(GUARDED_RESULTS_PATH_COMBINED, 'a', newline='',
                    csv.writer(f).writerow([prompt] + list(result.values(
            except Exception as e:
                print(f"Error on guarded prompt: {prompt}. Error: {e}")

        # We alraedy have Baseline Run results

    print("Dynamic Alpha experiment generation complete.")
```

```
New guarded results will be saved to: /content/HallucinationVectorProj
Initialized CSV file at: /content/HallucinationVectorProject/results/c
Starting response generation for baseline and DYNAMIC guarded models..
Dynamic Alpha Evaluation: 100%|██████████| 617/617 [36:36<00:00,  3.56
```

Run Judging, Analyze, and Summarize Results

Runs the judging process on generated answers, merges with ground truth, and computes final performance metrics for the combined guardrail experiment.

```
from evaluate_guardrail import run_judging_process
import utils
import config
import pandas as pd

# --- Redefine paths for the analysis ---
GUARDED_JUDGED_PATH_COMBINED = os.path.join("/content/HallucinationV
BASELINE_JUDGED_RESULTS_PATH = os.path.join("/content/HallucinationV
GUARDED_RESULTS_PATH_COMBINED = os.path.join("/content/Hallucination
BASELINE_RESULTS_PATH = os.path.join("/content/HallucinationVectorPr

# Load the test set
test_df = pd.read_csv("/content/HallucinationVectorProject/data/fina

# Load the newly generated results
guarded_df = pd.read_csv(GUARDED_RESULTS_PATH_COMBINED)
baseline_df = pd.read_csv(BASELINE_RESULTS_PATH)

# Merge with ground truth
guarded_merged_df = pd.merge(guarded_df, test_df, left_on='prompt',
baseline_merged_df = pd.merge(baseline_df, test_df, left_on='prompt'

# --- Run Judging ---
secrets = utils.load_secrets()
run_judging_process(guarded_merged_df, GUARDED_JUDGED_PATH_COMBINED,
# Assuming baseline is already judged, if not, uncomment below
# utils.run_judging_process(baseline_merged_df, config.BASELINE_JUDG

# --- Analyze and Print Final Report ---
guarded_judged_df = pd.read_csv(GUARDED_JUDGED_PATH_COMBINED)
baseline_judged_df = pd.read_csv(BASELINE_JUDGED_RESULTS_PATH)

baseline_accuracy = baseline_judged_df['is_correct'].mean()
guarded_accuracy = guarded_judged_df['is_correct'].mean()
baseline_error_rate = 1 - baseline_accuracy
guarded_error_rate = 1 - guarded_accuracy
relative_error_reduction = (baseline_error_rate - guarded_error_rate
baseline_latency = baseline_judged_df['latency_seconds'].mean()
guarded_latency = guarded_judged_df['latency_seconds'].mean()
latency_increase_percent = (guarded_latency - baseline_latency) / ba

summary_data = {
    "Metric": ["Accuracy", "Hallucination Rate", "Avg Latency (s)",
    "Baseline Model": [f"{baseline_accuracy:.2%}", f"{baseline_error
    "Guarded Model (Dynamic Alpha)": [f"{guarded_accuracy:.2%}", f"{
}
summary_df = pd.DataFrame(summary_data)

print("\n--- Final Performance Summary (Dynamic Alpha Experiment) --
display(summary_df)
```

```
Loading secrets...
Secrets loaded successfully.

--- Starting Corrected Judging Process for combined_guarded_judged_r
Initialized CSV file at: /content/HallucinationVectorProject/results
Found 0 already judged prompts. Resuming...
Judging combined_guarded_judged_results.csv: 100%|██████████| 617/61
--- Final Performance Summary (Dynamic Alpha Experiment) ---
```

	Metric	Baseline Model	Guarded Model (Dynamic Alpha)
0	Accuracy	38.57%	51.22%
1	Hallucination Rate	61.43%	48.78%
2	Avg Latency (s)	3.86	3.56
3	Relative Error Reduction	N/A	20.58%
4	Latency Increase	N/A	-7.80%