

Reproducibility Guide for Linguistic Steering Experiments

August 4, 2025

1 Step 1: Environment Setup

Before running any scripts, you must configure your environment and install the necessary Python packages. This project relies on several data science libraries and specific API clients for LLMs.

1. **Install Python Packages:** The project does not include a single setup script. You must install the required packages manually, likely using a 'requirements.txt' file or 'pip'. Based on the imports in the scripts, key dependencies include:
 - Data handling: `pandas`, `numpy`
 - Datasets: `datasets` (from Hugging Face)
 - Machine Learning: `scikit-learn`
 - Plotting: `matplotlib`, `seaborn`
 - API Clients: `openai`, `replicate`, `google-generativeai`
2. **Set API Keys:** The core script requires API credentials for the LLM provider you intend to use. You must provide these either as command-line arguments or by setting them as environment variables.
 - `OPENAI_API_KEY`
 - `REPLICATE_API_TOKEN`
 - `GOOGLE_API_KEY`
3. **Adjective Lists:** Ensure the adjective files (`adjectives08.txt`, `adjectives100.txt`) are present in the project directory.

2 Step 2: Phase 1 - Main Experiment Execution

The primary data generation is handled by `estimate_importance.py`. This script samples questions from the MMLU dataset, queries an LLM with prompts modified by different combinations of adjectives, and uses a regression model to estimate the "Shapley value" or influence of each adjective on the model's correctness.

To run the main experiment, execute the following command, customizing the parameters for your chosen setup:

```
# Example using OpenAI's gpt-4o with the 100-adjective list
python estimate_importance.py --adjectives_file adjectives100.txt --api_provider openai
```

This process can be lengthy and expensive, as it makes many API calls. The script is designed to be resumable; if it's interrupted, it will load the existing output file and continue where it left off.

3 Step 3: Phase 2 - Data Post-Processing (Optional)

The script `correct_outlier.py` is a utility for data cleaning. If the initial data generation resulted in numerical artifacts or extreme outlier values (as noted in the script's default threshold of `1e3`), this script can be used to normalize them.

```
# Example of correcting the raw results and saving to a new file
```

```
python correct_outlier.py --input_file gpt4o_results.json --output_file gpt4o_corrected.js
```

4 Step 4: Phase 3 - Primary Analysis and Visualization

Once you have a clean results file (e.g., `gpt4o_corrected.json`), you can generate the main analysis report using `analyze.py`. This is the core analysis script that produces most of the key tables and plots discussed in the paper.

```
# This script takes a single JSON file as input
```

```
python analyze.py gpt4o_corrected.json
```

This command will create a new directory named `gpt4o_corrected_analysis/`. Inside, you will find:

- `*_summary.json`: A JSON file with key statistics, like top adjectives and domain sensitivity.
- `*_summary_table.md`: A Markdown file ranking all adjectives by their overall impact.
- Multiple plots (`.png` files), including the mean Shapley values, overall impact distribution, and the quadrant analysis.

5 Step 5: Phase 4 - Secondary and Meta-Analyses

The remaining scripts perform more targeted analyses on the generated JSON results.

- **Variance and Hypothesis Testing:** To assess the statistical significance and stability of the results, use `analyze_variance.py` and `onesample_test.py`.

```
# Perform bootstrap analysis on the top 10 adjectives
```

```
python analyze_variance.py --input_results gpt4o_corrected.json --top_k 10
```

```
# Run a one-sample t-test to check if impact is > 0
```

```
python onesample_test.py --input_results gpt4o_corrected.json --top_k 10
```

- **Comparative and Meta-Analysis:** If you have run the experiment with multiple models, you can compare their results using `correlation_ranks.py` and `meta_analysis.py`.

```
# Calculate the rank correlation between two result files
```

```
python correlation_ranks.py gpt4o_corrected.json o3_corrected.json --output ranks.csv
```

```
# Run a full meta-analysis across multiple analysis directories
```

```
python meta_analysis.py GPT4:gpt4o_corrected_analysis/ Opus:o3_corrected_
```

- **Follow-up Experiments:** The `follow_up_analysis.py` script performs more advanced experiments (template variance, second-order interactions) based on the initial results. It re-runs a smaller set of API calls.

```
# Run follow-up tests on the top 5 adjectives from the first run
```

```
python follow_up_analysis.py --input_results gpt4o_corrected.json --top_k
```