About

This repository provides tools for training CLIP-HBA models with various perturbations and evaluating the perturbations' effects on model performance and model-human alignment. In particular, the training pipeline supports adding input or target noise (e.g. label shuffle, random target embeddings, image noise, etc.) during training, so one can systematically study how these perturbations impact model behavior.

The core model is a CLIP-based HBA network (with DoRA layers) that is instantiated in code as CLIPHBA. The training logic (`run_training_experiment`) reads a config file describing the architecture and perturbation schedule, applies the chosen perturbation strategy via `choose_perturbation_strategy`, and trains the model accordingly. The repository also includes an inference pipeline (`run_inference`) which loads a trained model checkpoint, applies the model to a test dataset (e.g. the THINGS, NOD, or NIGHTS dataset), and collects evaluation results.

The training and inference in this repository is built off of the original CLIP-HBA sourcecode published at https://github.com/stephenczhao/CLIP-HBA-Official/tree/main[1].

Quickstart

1) Clone the repo

```
git clone https://github.com/marrenj/cvpr_perturbations.git cd cvpr_perturbations
```

2) Create environment

```
conda env create -f environment.yml conda activate cvpr_perturbations
```

3) Verify installation

```
python -c "import torch; print(torch.cuda.is_available())"
```

Expected output:

```
True
```

4) Run a training experiment (minimal example)

```
python scripts/run_training.py \ --config configs/training_config.yaml
```

This will:

- Load CLIP-HBA with DoRA layers
- Apply the perturbation schedule defined in the config
- Train the model
- Save checkpoints + logs

5) Run inference on a trained model

Folder Structure

The folder structure is organized into modular components as follows:

- `src/` – Core code. This includes submodules for models (e.g. CLIP-HBA definitions), data (dataset loaders like `ThingsBehavioralDataset`, `NODDataset`, etc.), training (training loops and trainer functions), inference (scripts to run evaluation on held-out data), perturbations (perturbation strategy implementations like `TargetNoisePerturbation`, `LabelShufflePerturbation`, etc.), evaluation (e.g. RSA or behavioral analysis utilities), and generic utility functions (logging, seeding, path setup, etc.). For example, the training code imports from s`rc.models.clip_hba`, `src.data`, and `src.perturbations` to build and perturb the model.
- `scripts/` – High-level entry-point scripts. These scripts parse command-line arguments (e.g. `--config`) and call into the `src.training` or `src.inference` functions to execute the experiment. For example:
  - `scripts/run_training.py` loads a training config and launches the training pipeline.
  - `scripts/run_inference.py` loads an inference config and runs the evaluation (e.g. computing predictions or metrics on a test set).
- `data/` – Data storage (often git-ignored). This may include raw stimulus annotations or preprocessed files used by the datasets (e.g. the image directories and embedding CSVs referenced in configs).
- `notebooks/` – Additional code (e.g. analysis notebooks) as needed.
  - `notebooks/figures/` contains code to generate each figure.
  - `notebooks/neural_processing/` contains code to process fMRI data and calculate model-brain alignment.

Usage

Configure your experiment via the provided YAML files and run the training or inference scripts. For example, to train a model you might run: `python scripts/run_training.py --config configs/training/baseline_seed3.yaml`

And to run inference (evaluation) on a trained model, e.g. on the “Nights” dataset:

```
python scripts/run_inference.py --config configs/inference/nights.yaml
```

Each `--config` path points to a YAML file under `configs`/ that sets all relevant parameters. To customize an experiment, copy or edit one of the YAML templates: change the backbone (e.g. `backbone: ViT-L/14`), dataset paths, or perturbation settings (`perturb_type`, `perturb_epoch`, etc.) in the YAML file. The scripts will load these configs and automatically apply your chosen settings.

```
backbone: ViT-L/14
vision_layers: 2
transformer_layers: 1
rank: 32

epochs: 500
batch_size: 64
lr: 3e-4
criterion: MSELoss
random_seed: 3
cuda: 0
```

```
dataset_type: things
img_annotations_file: path/to/annotations.csv
img_dir: path/to/images

perturb_type: random_target   # e.g. 'None', 'random_target',
'label_shuffle', etc.
perturb_epoch: 10
perturb_length: 5
perturb_seed: 42

checkpoint_path: experiments/2025-...  # where to save model checkpoints
logger: None
```

And similarly the inference configs include fields like `model_weights_path`, `dataset`, and `inference_save_dir`.

These settings are all loaded at runtime (see code in `src/training/trainer.py` and `src/inference/inference_core.py`). For example, the training code uses `choose_perturbation_strategy` (`perturb_type`, `perturb_epoch`, `perturb_length`, `perturb_seed`) to apply the specified noise or shuffle each epoch. The inference code loads a saved checkpoint, constructs the same CLIP-HBA model, applies DoRA layers, and then calls a dataset-specific evaluation function (e.g. `evaluate_nights`) based on `config['dataset']`.

By editing or creating your own YAML files, you can run new experiments. For example, to sweep over different seeds or perturbation lengths, duplicate a config and change the random_seed, perturb_length, etc., then run `run_training.py` for each. The modular structure makes it easy to plug in new perturbations or datasets if needed.

The core logic for training is implemented in `src/training/` (see `run_training_experiment` in `trainer.py`). Perturbation strategies are defined in `src/perturbations/perturbation_utils.py` (classes like `TargetNoisePerturbation`, `LabelShufflePerturbation`, etc.). The inference logic is in `src/inference/` (e.g. `run_inference` in `inference_core.py`). Configuration is fully driven by the YAML files under `configs/`, as shown above. For inference, set `evaluation_type` to `behavioral` or `neural` to extract embeddings, build RDMs, and run RSA against a reference RDM (provided via `reference_rdm_path` or derived from dataset targets); set `evaluation_type: triplet` to evaluate the NIGHTS triplet task via `evaluate_nights`. Each run returns and saves the score per checkpoint epoch.

References

1. Zhao, S. C., Hu, Y., Lee, J., Bender, A., Mazumdar, T., Wallace, M., & Tovar, D. A. (2025). Shifting attention to you: Personalized brain-inspired AI models. arXiv. https://arxiv.org/abs/2502.04658