ChatGPT

Executive Summary

Neural networks often exploit **shortcut features** – spurious correlations that drive high training accuracy but fail under distribution shift [1] [2]. These "shortcuts" are typically *easy-to-learn* patterns that dominate early training, a phenomenon known as **simplicity bias** [3] [4]. In this work, we propose to **identify and mitigate** such shortcut learning by analyzing weight trajectories and applying *task-vector editing* (aka *task arithmetic*) to "subtract out" spurious behavior. Task vectors, defined as the difference between a pre-trained model and its fine-tuned weights on a task [5], provide a powerful way to steer a model. For example, Ilharco et al. showed that **negating** a task vector reduces performance on that task while leaving other tasks largely intact [6] [7]. We will: (1) survey literature on shortcut learning and model editing; (2) outline experiments to replicate key task-arithmetic results (datasets, models, hyperparameters); (3) develop algorithms (PCA, CCA, gradient probing, representer theorems) to detect "shortcut directions" from weight trajectories; (4) design task-vector based and other interventions to reduce shortcut use; (5) plan a step-by-step experimental pipeline with ablations. Deliverables include tables (e.g. model accuracies for various editing methods), example figures (e.g. weight-space trajectory plots, mermaid flowcharts of our pipeline), and pseudocode for computing and applying task vectors. All claims are grounded in recent research [1] [5] [6] [7] [8] [9].

1. Literature Survey

- **Shortcut learning / spurious features.** Geirhos et al. define shortcuts as decision rules that "perform well on standard benchmarks but fail to transfer to more challenging testing conditions" [1]. In practice, shortcuts correspond to features that correlate with labels in training but are not truly causal. Many works (e.g. Murali et al. 2023) show that models **prefer weakly predictive cues** such as background, watermark, or texture over harder "core" features [2] [3]. This is compounded by *simplicity bias*: early in training, DNNs latch onto the easiest patterns, even if spurious, and then "anchor" on them [3]. (For example, gender can become a proxy for hair color in CelebA [3].) Recent studies empirically confirm that harmful shortcuts tend to appear in the **early layer learning dynamics** [4] [2]. Techniques like **Prediction Depth** (Baldock et al. 2021) or error-based scores can detect which samples/features are learned first, flagging potential shortcuts.

- **Task arithmetic / model editing.** Ilharco et al. introduced *task vectors* as weight-space edits: $$v_T = w_{\text{fine-tuned on task }T} - w_{\text{pretrained}},$$ and showed that we can "add" or "negate" these vectors to steer behavior [5] [6] [7]. For instance, negating a vector fine-tuned on toxic text generation causes the model to "unlearn" toxicity (reducing toxic outputs) [7], while adding task vectors from multiple tasks can yield a one-model solution to multitask problems [10]. Crucially, this editing is efficient (just weight arithmetic) and can be performed without task data. Task arithmetic has also been applied to **bias mitigation**: Ding et al. (2025) showed that subtracting the "provenance" task vector from an NLP model (trained to identify data source) reduces spurious site-specific cues [11] [12]. In parallel, a line of work on *model editing* targets shortcut or factual errors after the fact. For example, Hakemi et al. (2021) conceptualize a spurious feature as a "fictitious subclass" and show that **post-hoc editing of a single weight** can remove this subclass [8] [13]. Yang et al. (2025) propose dynamic rank-one edits to correct a model's behavior on corrupted/spurious

inputs by localizing the responsible layer and adjusting weights there [9]. In NLP, fact-editing methods (e.g. De Cao et al. 2021, Meng et al. 2023) use similar vector-based edits to update factual knowledge, which underscores the versatility of weight-space interventions [14] [15].

- **Early-learning and training dynamics.** Several sources emphasize that monitoring just final accuracy misses how shortcuts emerge. Murali et al. (TMLR 2023) show that **easier-to-learn spurious features** can sometimes even boost training accuracy, but they harm generalization unless the model also learns robust features [16]. They suggest tracking per-example "difficulty" metrics (Prediction Depth) during training to identify when spurious cues are being picked up early [17] [4]. The recent Google research on *early readouts* is also relevant: Tiwari & Shenoy (2024) demonstrate that linear probes on early-layer activations make *confident errors* on spurious-inputs, giving a signal of over-reliance on shortcut features [18]. In summary, the literature suggests that (i) shortcut features are typically captured by early training, (ii) one should analyze intermediate representations or weight trajectories to detect them, and (iii) weight-space editing (task arithmetic or related) can be used to remove or suppress these learned shortcuts [3] [8].

2. Experimental Methods (Reproducing Task-Arithmetic Experiments)

Datasets & Tasks: To replicate Ilharco et al. (ICLR '23), we will use the same vision and language tasks. For vision, we fine-tune a CLIP ViT model (e.g. ViT-L/14) on eight standard image-classification datasets: Stanford Cars [19], Describable Textures (DTD) [20], EuroSAT [21], German Traffic Sign (GTSRB) [22], MNIST [23], RESISC45 (scene classification) [24], SUN397 [25], and SVHN (street number recognition). (We will also consider *spurious-focused tasks*, e.g. an OCR task or face ID task as in Ilharco et al. [26], to capture anthropic shortcuts.) The control task is ImageNet classification. For language, we fine-tune GPT-2 Large on toxic-comment detection (e.g. CivilComments dataset with toxicity score >0.8 [27]).

Models and Training: We start from publicly available **pretrained models**: CLIP-ViT models (Radford et al.) and GPT-2 (Radford et al.). We then fine-tune on each task to obtain task-specific weights. Following Ilharco et al., we **freeze** the output layers (so no new parameters) and fine-tune with AdamW (weight decay 0.1), learning rate 1e-5, batch size 128, for ~2000 iterations (with cosine LR schedule and 200 warm-up steps) [28]. This yields fine-tuned weights w_T for each task T. The *task vector* is $v_T = w_T - w_{\text{pre}}$ [5] [28]. To evaluate, we apply edits of the form $w_{\text{new}} = w_{\text{pre}} + \alpha v_T$ (with scale α chosen on a validation set, typically $\alpha=1$ to apply the full vector). A special case is $\alpha=-1$ (negation) to *forget* task T.

Baselines: In replicating key experiments [6] [7], we include baselines: (1) **Gradient-Ascent Fine-Tuning**: training to *maximize* the loss on T (i.e. move in the direction of increasing error) [29]; (2) **Random Vector**: add a random weight vector of the same norm as v_T [30]; (3) **Fine-Tune on Non-Shortcut Data**: e.g. fine-tune GPT-2 on the non-toxic subset (toxicity <0.2) [31]. These controls help isolate the effect of the task vector edit versus naive approaches.

Extracting Weight Trajectories: To study *when* and *how* shortcuts are learned, we will record snapshots of the full model weights (or key layers) at regular intervals during fine-tuning (e.g. every 50–100 steps). This produces a trajectory w_t in parameter space for each task. We will store these weight vectors to later

analyze their principal directions and dynamics. (For feasibility, one might compress by saving only the dense layers or computing PCA online.)

Metrics for Shortcut Detection: - *Task vs Control Accuracy:* For each edit we measure accuracy on the target task T and on control tasks (ImageNet for vision; Wikitext perplexity for language [7]). A successful *shortcut edit* should drastically reduce T-accuracy (or increase error) while minimally affecting control tasks [6] [7]. - *Feature Importance/Probing:* We will train linear probes or use representer methods to see which input features (e.g. background vs object) the model relies on. If a task vector is truly "spurious", subtracting it should reduce reliance on those features. - *Gradient Alignment:* We can compute the cosine similarity between weight updates at each step and the final task vector v_T [30]. A high alignment early on indicates learning along the shortcut direction. - *Statistical Tests:* All experiments will be run with multiple random seeds. We will compute means and confidence intervals of accuracies, and use paired t-tests (or nonparametric tests) to confirm significance of differences (e.g. negation vs random edit).

3. Analysis of Shortcut Directions

To pinpoint "shortcut directions" in parameter space, we will apply several analysis tools on the collected weight trajectories and task vectors:

- **Principal Component Analysis (PCA):** Stack the weight-difference trajectories ${w_t - w_{\text{pre}}}$ (or the task vectors v_T across tasks) and compute their principal components. We expect that the top PCs capture the most salient features learned early. For example, if the same spurious feature (e.g. background color) appears in multiple tasks, that direction may dominate. Projecting weight trajectories onto the top-2 PCs and coloring by epoch can visualize how training first moves along a "shortcut axis". Figure suggestion: *Weight-space trajectory plot:* a 2D PCA scatter where each point is a model state at a training step, colored by task or epoch, highlighting the straight-line movement when learning a simple shortcut.

- **Canonical Correlation Analysis (CCA):** Compare activations or weight changes between models. For instance, take the activations of an intermediate layer on core-feature examples vs spurious-feature examples, and apply CCA to find correlated subspaces. A high canonical correlation might reveal that the network uses a common subspace for multiple spurious tasks. (This echoes how representational similarity analysis is used for neural networks.)

- **Linear Probes / Representer Theorem:** Train small linear classifiers on hidden-layer representations to decode the presence of known shortcut attributes (e.g. texture, watermark, background class). If a probe on an early layer achieves high accuracy on a spurious label, it indicates the network has already encoded that shortcut. Using the representer theorem (representing the model's output as a weighted sum of training point influences), we can identify which training examples (e.g. those with spurious labels) dominate the model's decisions [32].

- **Gradient Alignment and Loss Landscape:** At each training step, compute the dot product of the gradient with the (normalized) task vector. A consistently positive dot means the model is moving *towards* the direction of that vector. We can plot gradient alignment over epochs: a spike early on suggests the model initially "points" in the shortcut direction. Additionally, we can track the norm of weight difference $|w_t - w_{\text{pre}}|$ and see at what point it saturates.

- **Quantifying Early Learning:** Following Murali et al. and Google Research, we will quantify how early spurious patterns emerge. One approach is to use *"Prediction Depth"*: estimate, for each training example, the epoch at which it is correctly classified for the first time [17]. Easy (potentially spurious) examples will have very early depths. Plotting the distribution of depths for examples that share a spurious attribute vs core examples can reveal if spurious examples are learned systematically earlier. Another metric is the error/confidence of early readouts (as in Tiwari & Shenoy): if an early-layer probe makes very confident but wrong predictions on certain examples, these may correspond to spurious-feature cases [18].

- **Visualization:** Besides PCA plots, we will use UMAP or t-SNE to project model weight differences or task vectors onto 2D for all tasks (color-coded by dataset). Clusters of tasks may appear (e.g. EuroSAT and RESISC45 cluster together as both are satellite datasets [33]). A **mermaid flowchart** will illustrate the pipeline (see below). A **timeline chart** (mermaid) could show training phases: pretraining → shortcut acquisition → editing → evaluation.

4. Interventions to Mitigate Shortcuts

- **Task-Vector Editing:** The primary intervention is *task arithmetic*, extending Ilharco et al. For a known spurious cue (e.g. background class), we define a corresponding "shortcut task". For example, train the model to predict just the background (or texture) labels. The resulting task vector v_{spurious} encodes the model's knowledge of that shortcut. We then subtract or *negate* this vector from our main model: $w_{\text{new}} = w_{\text{orig}} - \alpha v_{\text{spurious}}$. Ideally, this will **reduce reliance** on the spurious feature. By adjusting α, we can control the degree of "forgetting" (as done with toxicity in [7]).

- **Combining Multiple Task Vectors:** If multiple shortcuts are present, we can apply a weighted sum of negations. For example, subtract vectors for both "background" and "color" tasks. Conversely, adding auxiliary positive tasks (core feature detection) might help. We can also borrow from the **analogy** paradigm: if we have a task analogy "clean face photo vs masked photo, male vs female", combining three of these vectors may improve the fourth task without data (per Ilharco et al. Section 5).

- **Alternative Non-Editing Interventions:** In addition to weight editing, we will consider classical mitigation: (i) **Adversarial training** (train model to be invariant to spurious features); (ii) **Reweighting or regularization** (upweight hard examples, use IRM/V-REx style invariance training); (iii) **Data augmentation or SIS** (augment out spurious cues) [3]. These approaches will serve as benchmarks.

- **Evaluation Metrics:** To evaluate mitigation, we measure: (a) **In-distribution accuracy** on target and control tasks; (b) **Robust accuracy** under distribution shift (e.g. test on an OOD dataset without the shortcut). For instance, if background was the shortcut, test on images where background is altered. (c) **Worst-group accuracy:** following robust ML literature (e.g. Waterbirds), compute accuracy on the subgroup where the core label is least correlated with the shortcut. (d) **Trade-off analysis:** We will report any loss in original accuracy versus gains in robustness, possibly plotting accuracy vs α.

5. Experimental Plan

Pipeline Flowchart (Mermaid):

```
flowchart TD
    A[Start: Pretrained Model] --> B[Fine-tune on Task T and S (core and
spurious)]
    B --> C[Compute Task Vectors v_T, v_S]
    C --> D{Editing Operations}
    D -->|Negate v_S| E[Edited Weights w_new = w_pre - v_S]
    D -->|Add v_T+v_S| F[Multi-task Weights w_new = w_pre + v_T + v_S]
    D -->|Analogy combine| G[Analogy Weights via combinations]
    E --> H[Evaluate: Accuracy on task T & control]
    F --> H
    G --> H
```

This chart outlines how we fine-tune on tasks, compute task vectors, apply edits (negation, addition, analogies), and then evaluate.

Steps:
1. **Data preparation:** Collect the eight vision datasets and needed NLP corpora. Preprocess and split into train/val/test.
2. **Model initialization:** Load CLIP-ViT and GPT-2 checkpoints.
3. **Fine-tuning:** For each target task (and optional spurious-only task), fine-tune the model (see hyperparams above [28]). Save model weights at multiple checkpoints.
4. **Compute task vectors:** Subtract pretrained weights from fine-tuned weights to get v_T. For spurious tasks (e.g. background classifier), get v_S.
5. **Weight trajectory logging:** Keep snapshot weights w_t during training for later PCA/CCA analysis.
6. **Editing:** Apply edits: e.g. $w_{\text{neg}} = w_{\text{pre}} - v_T$, $w_{\text{add}} = w_{\text{pre}} + v_{T_1} + v_{T_2}$, analogies etc. Scale vectors based on held-out validation to keep control accuracy $\ge95\% $ [34].
7. **Evaluation:** Test each edited model on all tasks, record metrics. Compare to baselines (gradient ascent, random).
8. **Analysis:** Run PCA/CCA on saved trajectories (PCA on ${w_t}$ or on matrix of task vectors across tasks; CCA on activations of clean vs spurious examples), probe feature usage, plot trajectory embeddings.
9. **Interventions and ablations:** Vary α, try subtracting one vs multiple vectors, compare to adversarially trained model.
10. **Statistical tests:** Use bootstrapping or paired tests to ensure results (e.g. “negating the shortcut vector reduced task accuracy by $X\pm\sigma$” is statistically significant).

Compute and Resources: Fine-tuning CLIP models on these tasks is moderate (each task ~2k steps, vision ViTs on image subsets). We plan to use e.g. 8 GPUs (like NVIDIA A100). The GPT-2 runs are smaller (1k generations for evaluation as in Ilharco). Data loading and model size suggests roughly **tens of GPU-hours**. PCA/CCA on weight vectors is cheap since dimensionality is large but doable with random projections or truncated SVD.

Hyperparameters & Ablations: We will sweep: learning rates (0.5x/2x 1e-5), number of fine-tuning steps (1k vs 2k), and apply 3 random seeds per setting. For editing, vary vector scaling α (0.5–1.5). We will ablate: which layer subsets to edit (all weights vs last n layers). For analogies, we will reproduce the sentiment/task analogies from Ilharco's section 5.

Expected Outcomes: We expect to confirm that **negating shortcut task vectors** sharply drops performance on that spurious cue with minimal collateral damage [6] [7]. E.g. Table 1 below (modeled on Ilharco's) would show target-accuracy plummeting under "negative edit" while ImageNet/control accuracy is unchanged. Multi-task addition should yield a single model nearly as accurate as separate fine-tuned models on each task [10]. PCA of weight trajectories should reveal distinct directions for different tasks, with some common axes among shortcut-heavy tasks. We anticipate early PCs aligning with "easy" features.

6. Deliverables

Tables: We will produce tables such as:

- *Table 1: Model Accuracy for Forgetting Shortcut Tasks.* Columns: Method (None, Fine-tuned, Gradient-Ascent, Random, Negative Task Vector), rows: {Target Accuracy ↓, Control Accuracy}. This mirrors Ilharco's Table 1.
- *Table 2: GPT-2 Toxicity Mitigation.* Columns: Method; % toxic generations; control perplexity. (Similar to Ilharco's Table 2).
- *Table 3: Multi-task and OOD Performance.* Rows: Different editing strategies (e.g. Add tasks, Analogy edit, baseline), columns: average accuracy on tasks, robust/OOD accuracy, accuracy drop.

Each table will cite sources for baseline numbers (e.g. pre-trained/fine-tuned) and highlight how task-arithmetic methods compare [6] [7].

Figures: Besides the suggested flowchart above, we'll suggest figures like: - A *weight trajectory plot* (PCA 2D) where each point is the model weights at a given step (color=epoch), with arrows showing the learning path (gradual arrowheads). - A *PC scatter* of final task vectors (colored by dataset), showing clusters of similar tasks (e.g. EuoSAT vs RESISC45 close together [33]). - A *line chart* of accuracy vs training step for target vs control tasks, illustrating early learning (flat line for control, steep rise for target early on). - Any mermaid *timeline* if needed to depict research milestones.

Pseudocode (Python-like):

```
# Example: Compute and apply a task vector
model_pre = load_pretrained_model()
task_data = load_dataset(task='Cars')
# Fine-tune model on Cars
model_ft = fine_tune(model_pre, task_data, lr=1e-5, steps=2000,
freeze_head=True)
# Compute task vector
v_task = model_ft.weights - model_pre.weights

# Negative editing: create "unlearned" model
```

```
w_new = model_pre.weights - v_task  # equivalent to model_pre + (-1 * v_task)
model_neg = assign_weights(model_pre.copy(), w_new)

# Evaluate
acc_target = evaluate(model_neg, task_data.test)
acc_control = evaluate(model_neg, imagenet_test)
print(f"Target Acc: {acc_target:.1f}%, Control Acc: {acc_control:.1f}%")
```

(We would similarly compute `v_spur` for a spurious-feature task and subtract it.) All code will be implemented in PyTorch, with reproducibility scripts.

Summary of Datasets/Models/Methods: Finally, a table (or bulleted list) will compare the *methods and datasets* side-by-side, e.g.:

Dataset/Task	Model	Shortcut	Detection Method	Intervention
ImageNet (control)	CLIP ViT-L/14	–	–	–
Stanford Cars	CLIP ViT-L/14	background vs car color	Weight negation, PCA	Negate task vector of Cars
Waterbirds (spurious)	ResNet50	water background	Linear probe (mask)	Remove background vector (TAPER)
Toxic Generation	GPT-2 Large	profanity detector	Detoxify metric	Negate toxicity task vector
GLUE-SST2 (text sentiment)	T5-base	dataset domain shift	Accuracy on out-of-domain	Add domain adaptation vectors
…	…	…	…	…

(This is illustrative; actual experiments focus on the tasks described above.)

Throughout, we will cite relevant work for each choice. For example, using Waterbirds as a "shortcut" benchmark and computing worst-group accuracy is standard practice [2]. We will rely on published hyperparameters (as above [28]) and code repositories (e.g. [30] Ilharco's GitHub) as needed.

References: All the above proposals and observations are grounded in the literature [1] [2] [5] [6] [7] [8] [9] [3]. These will be cited exactly in the text at the relevant points.

---

[1] arxiv.org
https://arxiv.org/pdf/2004.07780

[2] [16] [17] Beyond Distribution Shift: Spurious Features Through the Lens of Training Dynamics - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC11029547/

3 18 Intervening on early readouts for mitigating spurious features and simplicity bi
https://research.google/blog/intervening-on-early-readouts-for-mitigating-spurious-features-and-simplicity-bias/

4 [2302.09344] Beyond Distribution Shift: Spurious Features Through the Lens of Training Dynamics
https://arxiv.org/abs/2302.09344

5 6 7 10 14 15 19 20 21 22 23 24 25 26 27 28 29 30 31 33 34 [2212.04089] Editing Models with Task Arithmetic
https://ar5iv.labs.arxiv.org/html/2212.04089

8 13 Single-weight Model Editing for Post-hoc Spurious Correlation Neutralization
https://arxiv.org/html/2501.14182v2

9 Dynamic Model Editing to Rectify Unreliable Behavior in Neural Networks | OpenReview
https://openreview.net/forum?id=1dkL3MVBfV

11 12 Tailoring task arithmetic to address bias in models trained on multi-institutional datasets - ScienceDirect
https://www.sciencedirect.com/science/article/abs/pii/S1532046425000875

32 [2504.15664] An XAI-based Analysis of Shortcut Learning in Neural Networks
https://arxiv.org/abs/2504.15664