Exploring and Leveraging Class Vectors for Classifier Editing

Jaeik Kim[1] **Jaeyoung Do**[† 1,2,]
AIDAS Laboratory, [1]IPAI & [2]ECE, Seoul National University
† indicates corresponding author
{jake630, jaeyoung.do}@snu.ac.kr

Abstract

Image classifiers play a critical role in detecting diseases in medical imaging and identifying anomalies in manufacturing processes. However, their predefined behaviors after extensive training make post hoc model editing difficult, especially when it comes to forgetting specific classes or adapting to distribution shifts. Existing classifier editing methods either focus narrowly on correcting errors or incur extensive retraining costs, creating a bottleneck for flexible editing. Moreover, such editing has seen limited investigation in image classification. To overcome these challenges, we introduce Class Vectors, which capture class-specific representation adjustments during fine-tuning. Whereas task vectors encode task-level changes in weight space, Class Vectors disentangle each class's adaptation in the latent space. We show that Class Vectors capture each class's semantic shift and that classifier editing can be achieved either by steering latent features along these vectors or by mapping them into weight space to update the decision boundaries. We also demonstrate that the inherent linearity and orthogonality of Class Vectors support efficient, flexible, and high-level concept editing via simple class arithmetic. Finally, we validate their utility in applications such as unlearning, environmental adaptation, adversarial defense, and adversarial trigger optimization.

1 Introduction

Classifiers have long been fundamental in Computer Vision (CV), applied in diverse fields from medical imaging [35] to anomaly detection [84]. With the rise of Vision Transformers (ViTs) [13, 44], their classification capabilities have significantly improved, leading to the widespread availability of fully fine-tuned models across various tasks. As a result, open platforms such as HuggingFace now offer extensive collections of classifiers, enabling plug-and-play usage for diverse applications [66]. However, even within the same task, users may have specific requirements for certain deterministic rules. For example, in disease diagnosis, some users may prioritize minimizing errors for specific conditions they handle, as even minor misclassifications can have serious consequences. Alternatively, others may require a classifier that performs reliably in their own distributional context, such as a snowy environment. Thus, a *one-size-fits-all* classifier is impractical for meeting diverse user needs within the current model supply chain. This highlights the importance of classifier editing, which modifies class-specific knowledge post hoc while preserving unrelated prior knowledge [72].

Despite its importance, classifier editing remains challenging because deeply optimized models encode rigid behaviors shaped by their training distributions. For example, data scarcity for vehicles in snowy scenes often teaches the model to adopt the shortcut *vehicle* + *snow* $\rightarrow$ *snowplow*, causing it to mislabel buses in snow as snowplows [60, 26]. Moreover, efficiently modifying classifiers with minimal data in real-world scenarios remains an open problem. Recent classifiers (*e.g.*, ViTs), for instance, require significantly more training compared to traditional CNNs [64, 68],

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

making knowledge modifications with few samples more difficult [39, 36] and increasing the risk of introducing new biases [67, 5]. As a result, existing classifier editing methods are computationally intensive [80] and often rely on auxiliary information such as object mask, requiring representations to be modified one by one for each image [60]. These challenges, amplified by the sparse information density of visual data [23], require defining "*where-to-edit*" and remains underexplored in vision models, whose scope is largely restricted to correcting image-wise misclassifications [62, 80].

To address these challenges, we revisit recent image classifiers through the lens of a model's adaptation to specific classes during training by introducing the novel concept of *Class Vectors*. Class Vectors capture per-class representation shifts during fine-tuning by computing the difference between the centroid representations of pretrained and fine-tuned models. Inspired by *task vectors* [29], which represent weight updates for tasks during fine-tuning, Class Vectors aim to disentangle class-specific behavior from task-wide adaptations. Although task vectors are effective for task-level applications such as model alignment [21, 6, 43, 25] and detoxification [29, 83], they inherently capture task-level modifications, limiting their applicability for fine-grained classifier editing. In contrast, Class Vectors operate at the class level and can be applied either by directly steering latent representations via a training-free approach or by mapping them back into weight space for model editing. This class-level modification alleviates existing editing constraints by replacing predictive rules across an entire class rather than adjusting the model on an image-by-image basis.

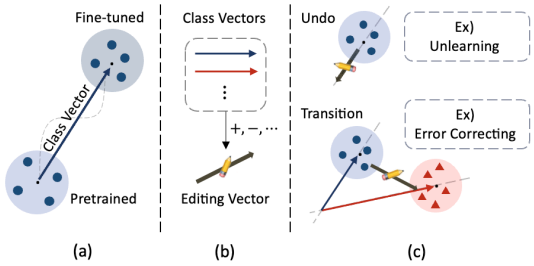


Figure 1: Class Vector and its applications. (a) Class Vector captures centroid representation adaptation in the latent space. (b) Editing vector with high-level concepts using arithmetic operations on Class Vectors. (c) Editing vectors can undo predictive behaviors by reversing the adaptation direction, or transition the classifier logic to correct errors.

Our findings reveal that linear trajectories exist along which the model adapts to specific classes during fine-tuning, forming the basis of Class Vectors. This behavior remains barrier-free despite the complexity of high-dimensional representation learning and the nonlinear characteristics of modern classifiers [57, 56], supported by *Cross-Task Linearity* (CTL) [87]. We then explore two key properties of Class Vectors that enable their effective use in classifier editing: (1) linearity and (2) independence. During inter-class interpolation, predictions and logits transition smoothly along linear paths. Furthermore, we demonstrate that modifying a target class's representations does not influence other classes, confirming that Class Vectors act independently. This behavior is supported by the *Neural Collapse* [55]: during fine-tuning, class-specific feature shifts become quasi-orthogonal, enabling targeted adjustments with minimal interference. These fundamental properties of Class Vectors enable precise editing of class-specific predictive rules through simple arithmetic operations such as addition, subtraction, and scaling (Fig. 1), offering several advantages as follows:

- **High efficiency**: Enables edits without retraining via latent steering, or for specific tasks can be trained in under 1.5 seconds using fewer than 1.5K parameters and a single sample.
- **High-level interaction**: Facilitates intuitive high-level concept editing, also allowing non-expert users to perform edits without neural network expertise.
- **Flexibility**: Provides precise control over the degree and nature of edits by adjusting the scaling coefficients of Class Vectors to align with user intentions.

We present extensive experiments demonstrating the effectiveness of Class Vectors in real-world applications such as model unlearning, adapting to unfamiliar environments, preventing typographic attacks, and optimizing triggers for backdoor attacks.

2 Related Work

Adaptation vectors. Empirical studies of neural network loss landscapes show that fine-tuning proceeds along convex, well-aligned directions in weight space [18, 73]. Building on this, *task vectors* [29], the weight delta from a pretrained model to its fine-tuned version, have been used in classifiers [76, 78], large language models [77, 81], and LoRA adapters [74, 10], enabling multi-task

learning [27, 79], detoxification [61], and style mixing [74]. Meanwhile, in large language models (LLMs), *in-context vectors* steer model outputs at inference time by encoding task- or example-level instructions as additive offsets in latent space (*e.g.*, for controllable text generation [43, 25, 47]). Such adaptation vectors typically perform model editing at the global, task-wide level. In contrast, our *Class Vectors* isolate adaptations at the per-class level in latent feature space, providing localized, class-specific control with minimal interference to other classes. Unlike task vectors, which impose global weight shifts, Class Vectors capture intrinsic, persistent representation shifts that can be mapped to weights. They differ also from in-context vectors—transient, label-agnostic offsets—and from concept activation vectors (CAVs) [33], which describe rather than edit concepts.

Characterizing Neural Networks. Early work [18] demonstrated that the loss landscape along the straight-line path from random initialization to a fully trained model is nearly convex, suggesting that training could follow a linear trajectory. Linear Mode Connectivity (LMC) [28, 15, 49] then showed that independently trained models on the same task maintain almost constant loss under linear weight interpolation, and the concept of task vectors [29, 53] revealed that scaling these vectors yields semantically meaningful performance changes. Layerwise Linear Feature Connectivity (LLFC) [86] extended this phenomenon to the feature space, proving that at every layer the feature maps of an interpolated model align proportionally with the linear blend of the feature maps of the original models that show LMC. More recently, Cross-Task Linearity (CTL) [87] found that models fine-tuned on different tasks still exhibit approximate linear behavior in their features under weight interpolation, and Neural Collapse (NC) described how penultimate-layer features converge to equidistant class prototypes [55]. In this work, we show that pretrained-to-fine-tuned model pairs also satisfy a CTL with feature alignment, and we use NC to establish the independence of Class Vectors.

3 Foundations for Class Vectors

Given n data $\{x_1, x_2, \ldots, x_n\} \subset \mathcal{X}_{\text{task}}$ with corresponding k labels $\{c_1, c_2, \ldots, c_k\} \subset \mathcal{Y}$, image classifier is defined as $\mathcal{M}(\cdot, \theta \in \mathbb{R}^d) : \mathcal{X}_{\text{task}} \mapsto \mathcal{Y}$, comprising an encoder $f(\cdot, \theta^{\text{e}} \in \mathbb{R}^{d_e}) : \mathcal{X}_{\text{task}} \mapsto \mathcal{Z}$ and a classification head $g(\cdot, \theta^{\text{h}} \in \mathbb{R}^{d_h}) : \mathcal{Z} \mapsto \mathcal{Y}$. Here, classifier is represented as $\mathcal{M} = g \circ f$, where $\circ$ denotes function composition. Let $\theta^e_{\text{pre}} \in \mathbb{R}^{d_e}$ represent the pretrained weight and $\theta^e_{\text{ft}} \in \mathbb{R}^{d_e}$ be the fine-tuned weight of the classifier encoder for a specific task. We first define the Class Vector.

Definition 3.1 (Class Vector)**.** *For a class c, let $S = \{s_1, \ldots, s_{|S|}\} \subset \mathcal{S}$ denote the set of its samples. The* Class Vector $\kappa_c \in \mathbb{R}^m$ *is the difference between the expected last-layer representations of the fine-tuned and the pretrained encoders (i.e., penultimate layer of models):*

$$\kappa_c \;=\; \mathbb{E}_{s\in S}\big[f\big(s, \theta^{\text{e}}_{\text{ft}}\big)\big] \;-\; \mathbb{E}_{s\in S}\big[f\big(s, \theta^{\text{e}}_{\text{pre}}\big)\big].$$

The centroid representation of a fine-tuned classifier z^c_{ft} for a class c can be formulated as $z^c_{\text{ft}} = z^c_{\text{pre}} + \kappa_c$, where z^c_{pre} denotes pretrained centroid representation and $S = \mathcal{X}_{\text{task}}$.

3.1 Formal Justification

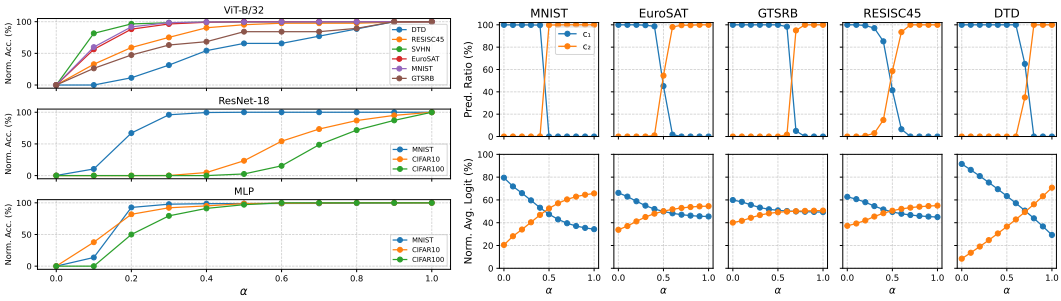
We aim to demonstrate that the class-specific changes induced by fine-tuning are captured by a *single latent vector* $\kappa_c := z^c_{\text{ft}} - z^c_{\text{pre}}$, so that merely scaling κ_c interpolates a smooth path of class-c behavior. To justify this claim, we build on two well-documented phenomena.

(i) Task-level weight linearity. Prior work [29] shows that the *task vector* $\tau = \theta^e_{\text{ft}} - \theta^e_{\text{pre}}$ captures a linearly meaningful direction in *weight space*: moving the weights along τ, $f\big(x; \theta^e_{\text{pre}} + \lambda\tau\big)$, causes predictable performance shifts as λ varies [18].

(ii) Cross-Task Linearity (CTL) [87]. When two fine-tuned checkpoints θ_i, θ_j originate from the same θ_{pre}, weight interpolation is almost equivalent to latent interpolation for *every* input x:

$$f(x; \alpha\theta_i + (1-\alpha)\theta_j) \;\approx\; \alpha\, f(x; \theta_i) + (1-\alpha)\, f(x; \theta_j).$$

CTL thus bridges weight-space linearity to latent-space linearity. In Theorem 3.1, beyond any pair of fine-tuned weights, we show that it is *even tighter* on the segment connecting the pretrained model to its fine-tuned checkpoint, and we confirm that this pretrain-to-finetune interpolation traces a semantically meaningful path in latent space.



(a) (b)

Figure 2: (a) Line-search between z^c_{pre} and z^c_{ft} explores linearly evolving representation. (b) Linear interpolation between cross-Class Vectors with ViT-B/32 shows smooth transition between classes.

Theorem 3.1 (CTL between pretrained and fine-tuned weights)**.** *Suppose the function $f : \mathbb{R}^p \to \mathbb{R}$, and two fine-tuned weights θ_i and θ_j satisfy CTL* [87]*. Let θ_{pre} be the pre-trained weights. Define*

$$\delta_{\text{pre},i} = \left| f(\alpha\theta_{\text{pre}} + (1-\alpha)\theta_i) - \big((1-\alpha)f(\theta_{\text{pre}}) + \alpha f(\theta_i)\big) \right|,$$

$$\delta_{i,j} = \left| f(\alpha\theta_i + (1-\alpha)\theta_j) - \big((1-\alpha)f(\theta_i) + \alpha f(\theta_j)\big) \right|.$$

If $\|\theta_i - \theta_{\text{pre}}\| < \|\theta_i - \theta_j\|$, then $\delta_{\text{pre},i} < \delta_{i,j}$: the segment from θ_{pre} to θ_i shows strictly smaller CTL deviation, hence is more linear*, than the segment between two fine-tuned solutions $\theta_i \to \theta_j$.*

The proof is provided in Appendix §C.1. Using Theorem 3.1, we can apply CTL to pair of θ_{pre} and θ_{ft}. Applying CTL to the set of inputs $x_c \in \mathcal{D}_c$ for a single class c and averaging over x_c yields:

$$f\big(x_c; \theta_{\text{pre}} + \alpha\tau\big) \;\approx\; f(x_c; \theta_{\text{pre}}) + \alpha\kappa_c, \qquad x_c \in \mathcal{D}_c.$$

Thus scaling the fixed vector κ_c in latent space reproduces the effect of moving θ_{pre} along τ for class-c samples, the class-specific adaptation. Fig. 2a (Top) visualizes this effect on ViT-B/32 across six downstream tasks: as α increases from 0 to 1, class-c accuracy (normalized by the fully fine-tuned score) rises smoothly and concavely, confirming the linear path predicted by the theory. We observe similar behavior in both an MLP and ResNet-18 [22] with two other tasks (CIFAR10, CIFAR100), indicating that Class Vectors arise independently of network architecture or finetuning specifics (Fig. 2a (Middle), Fig. 2a (Bottom)). Experimental evidence for the inequality $\|\theta_i - \theta_{\text{pre}}\| \;<\; \|\theta_j - \theta_i\|$ and training details for all models are in Appendix Fig. 8 and §D.3.

Take-away. The adaptation required for a single class can be approximated by scaling a single latent vector κ_c; class-wise representation learning often reduces to simple vector arithmetic.

3.2 Class Vector-based Editing

To edit classifiers, we first construct an editing vector $z_{\text{edit}} \in \mathbb{R}^m$ in the latent space by linearly using the Class Vectors (§4). Prior work has shown that task vectors (in the weight space) and in-context vectors (in the latent space) can steer model behavior; our approach supports both injection modes.

Latent-space injection. Given $r = f(x, \theta^e_{\text{ft}})$, we shift the representation by z_{edit} and obtain $\hat{y} = g(r + z_{\text{edit}}, \theta^h)$. To avoid collateral edits in other classes (*i.e.*, to ensure localization of the edit), we gate the shift with $\beta = \mathbf{1}[\text{sim}(r) > \gamma]$, where $\text{sim}(r)$ is given by the cosine similarity to z^c_{ft} and γ denotes thresholds for gating (Algorithm.1). Please note that in most cases, z^c_{ft} is known when constructing z_{edit} (§4), and latent space injection does not require additional training.

Weight-space mapping. Latent-space manipulation of the classifier cannot fundamentally alter the deterministic rules encoded in the model's weights, leaving decision boundaries unchanged [48, 30]. It also imposes additional gating computations on the editor and requires maintaining z^c_{ft}. Following previous editing approaches [80, 60, 46] that embed edits directly into the model weights, we introduce a method for permanently embedding editing vectors into the model parameters. To mapping editing vectors in the weight space, we learn $\phi_{\text{edit}} : \mathbb{R}^m \to \mathbb{R}^{d_e}$ such that

$$\theta^e_{\text{edit}} = \theta^e_{\text{ft}} + \phi_{\text{edit}}(z_{\text{edit}}),$$

Algorithm 1 Latent-space injection

Inputs: encoder $f(\cdot;\theta^e_{\text{ft}})$; head $g(\cdot;\theta^h)$; x; z_{edit}; class centroid z^c_{ft}; threshold γ
Output: logits $\hat{y}$
$r \leftarrow f(x;\theta^e_{\text{ft}})$
$\text{sim}(r) \leftarrow \frac{r^\top z^c_{\text{ft}}}{\|r\|\,\|z^c_{\text{ft}}\|}$
$\beta = \mathbf{1}[\text{sim}(r) > \gamma]$
$r_{\text{edit}} \leftarrow r + \beta\, z_{\text{edit}}$
$\hat{y} \leftarrow g(r_{\text{edit}};\theta^h)$
return $\hat{y}$

Algorithm 2 Weight-space mapping

Input: encoder $f(\cdot;\theta^e)$, data $\mathcal{X}_{\text{task}}$, z_{edit}, collecting number N, class c, epochs T
Freeze all but editable block $\mathcal{L}$
for $t = 1,\dots,T$ **and each** $(x,y) \in \mathcal{X}_{\text{task}}$ **do**
  Collect N class-c references $r_c = \{f(x) \mid y = c\}$
  Set $r_{\text{target}} = \text{mean}(r_c) + z_{\text{edit}}$ **if** r_{target} **= None**
  $r = f(x)$; $\ell = \| r[y=c] - r_{\text{target}}\|^2$
  Update $\mathcal{L}$ with $\nabla_{\mathcal{L}}\ell$
end for
return θ^e_{edit}

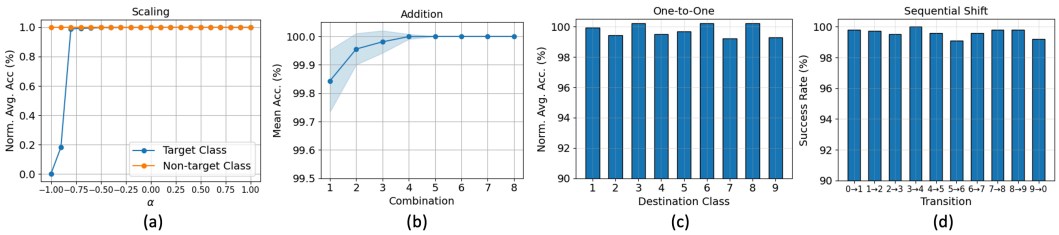


Figure 3: Independence of Class Vectors in MNIST. (a) Scaling the target class representation using $z_{\text{edit}} = \alpha \cdot \kappa_{c_1}$ (b) Adding non-target Class Vectors to the target class based on the combination count. (c) Modifying the target class to each destination class ($z_{\text{edit}} = \kappa_{\text{des.}} - \kappa_{\text{tar.}}$), with the averaged task accuracy. (d) Shifting all representations from $c_i \rightarrow c_{i+1}$ simultaneously with transition success rate.

minimising $\|f(x, \phi_{\text{edit}}(z_{\text{edit}})) - z_{\text{edit}}\|^2$. To make this more practical, assuming a linear mapping f (a common assumption [81, 17]), the optimization reduces to θ^e_{edit}:

$$\theta^e_{\text{edit}} = \underset{\theta^e_{\text{edit}}}{\text{argmin}} \|f(x,\theta^e_{\text{edit}}) - (f(x,\theta^e_{\text{ft}}) + z_{\text{edit}})\|^2.$$

Similar to latent space steering, to ensure that the edit only affects the intended class, we first collect reference samples from that class and compute their latent representations. We then add the Class Vector κ_c to each of these reference embeddings to form fixed target representations, and train the few encoder layers to map the original embeddings onto these shifted targets (Algorithm. 2). Theorem 3.2 shows that the mapping of Class Vectors from $\mathbb{R}^m$ into the model's weight space $\mathbb{R}^{d_e}$ admits infinitely many solutions, implying that it remains effective across diverse mapping configurations.

Theorem 3.2 (Existence of a Mapping)**.** *Let $\phi_{\text{edit}} : \mathbb{R}^m \rightarrow \mathbb{R}^{d_e}$ be any mapping that sends latent Class Vectors to weight perturbations applied in the encoder's final layer or a small subset of layers. Under the assumption that these edits are sufficiently small and confined to that small subset of layers of an overparameterized encoder (e.g., a ViT) with $d_e \gg m$, there exist infinitely many distinct ϕ_{edit}.*

See Appendix §C.2 for a detailed proof. Note that Theorem 3.2 guarantees the existence of a valid mapping for overparameterized encoders, provided that the edit is restricted to a local subset of layers. This theoretical result suggests that such mappings may be inherently robust to the specific training procedure used for the encoder. The training setup for mapping are provided in Appendix §D.3.2.

3.3 Properties for Effective Classifier Editing

We now explore the properties of Class Vectors. Throughout our experiments, we employ the classes listed in Tab. 7 and validate our findings via the weight-space mapping approach.

Linearity. The linearity between two classes in a fine-tuned model is crucial for editing, as barriers or divergence along the path may cause z_{edit} to fail, leading to unpredictable behavior. We interpolate between two fine-tuned classes c_1 and c_2 by $z_{\text{edit}} = -\alpha\,\kappa_{c_1} + \alpha\,\kappa_{c_2}$. This effectively shifts the model's adaptation from c_1 to c_2. As shown in Fig. 2b, predictions and logits change smoothly:

Table 1: Comparison of class unlearning with baselines, including the *mean* and *std* accuracies.

Method	MNIST		EuroSAT		GTSRB		RESISC45		DTD	
	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)
Pretrained	53.4±36.8	51.7±4.2	66.5±21.5	53.4±2.5	81.8±17.7	41.7±1.2	76.8±18.5	66.2±0.4	36.0±35.8	44.9±0.8
Fine-tuned	99.9±0.1	99.8±0.0	99.9±0.2	99.8±0.0	99.6±0.0	99.2±0.0	99.3±0.8	96.8±0.0	73.5±16.9	82.3±0.4
Retrained	0.1±0.1	76.4±0.0	0.0±0.0	85.7±0.0	41.8±25.3	57.5±0.0	33.9±20.6	75.0±0.0	14.5±26.5	55.5±0.0
NegGrad	0.0±0.0	43.4±10.3	0.0±0.0	11.6±1.0	0.0±0.0	15.6±25.6	0.0±0.0	2.5±0.6	0.0±0.0	13.6±16.7
Random Vector	99.9±0.1	99.8±0.0	99.9±0.1	80.9±26.1	99.6±0.5	98.2±0.7	100±0.0	79.4±19.1	72.0±31.8	51.1±11.1
Class Vector	0.0±0.0	99.7±0.0	0.0±0.0	99.5±0.2	0.0±0.0	98.6±0.0	28.2±26.1	94.6±7.2	13.5±16.5	78.1±0.8
Class Vector†	0.0±0.0	96.2±0.1	0.0±0.0	99.7±0.0	0.0±0.0	93.4±0.0	10.0±10.9	90.7±3.2	15.2±18.7	72.9±0.1

samples switch cleanly from c_1 to c_2 at the midpoint, with no detours to other classes. This results show that Class Vectors permit precise linear edits of the classifier.

Independence To modify class c_1 towards c_2 without effect other classes, we require $f(x', \theta_{\text{ft}}^e) = f(x', \theta_{\text{ft}}^e + \phi(z_{\text{edit}}))$ for all $x' \notin c_1$, while $f(x_{c_1}, \theta_{\text{ft}}^e + \phi(z_{\text{edit}})) = f(x_{c_2}, \theta_{\text{ft}}^e)$. Neural Collapse (NC) phenomenon [55] states that, near the end of training, (i) all penultimate-layer features belonging to the same class tightly collapse to a class mean, and (ii) these class means themselves form a simplex Equiangular Tight Frame (ETF) centred at the global mean. Building on NC structure, we show that a class-specific update vector κ_c exerts negligible influence on the embeddings of every other class.

Theorem 3.3 (Independence of Class Vectors)**.** *Suppose (i) the pretrained class embeddings collapse to a common mean $\bar{z}^{\text{pre}}$, that is $z_c^{\text{pre}} \approx \bar{z}^{\text{pre}}$; (ii) after fine-tuning the embeddings follow a centre-shifted ETF form $z_c^{\text{ft}} = \mu + u_c$ with $\sum_c u_c = 0$; and (iii) the global drift $\|\mu - \bar{z}^{\text{pre}}\|$ is negligible compared to the class-specific update $\|u_c\|$. Then, for any two distinct classes $c \neq c'$,*

$$\cos\big(\kappa_c,\, z_{c'}^{\text{ft}}\big) \;\approx\; 0,$$

i.e. the Class Vector κ_c is approximately orthogonal to the fine-tuned embedding of every other class.

A detailed proof and empirical evidence that strongly support these conditions for ViT are presented in Appendix §C.3. In Fig. 3, we empirically evaluate the independence of Class Vectors. It shows that Class Vectors preserve the accuracy of non-target classes and ensure independent edits across classes, even when multiple classes are edited simultaneously.

4 Editing Classifiers

We now introduce editing applications with Class Vectors. For all applications, we first design z_{edit}, then steer the models in latent spaces or map it to the weight space to alter their predictive behavior.

4.1 Experimental Setups

To evaluate Class Vectors for classifier editing, we extract pretrained and fine-tuned class centroids from three widely adopted CLIP encoders, ViT-B/16, ViT-B/32, and ViT-L/14 [58], and form Class Vectors as their differences. In §4.3 and §4.4, Class Vectors are derived from initialized and pretrained encoders, as they predict ImageNet classes, the pretraining dataset for these classifiers. We denote latent-space steering by *Class Vector* and weight-space mapping by *Class Vector*†, using a default similarity threshold of $\gamma = 0.5$. As baselines, we include *Retrained* [80, 60], which retrains only the target class using cross-entropy loss (or excludes it entirely in the unlearning setting), and *Random Vector*, which is initialized to match the magnitude of z_{edit} and mapped to the weight space to test for non-semantic effects. Among all considered methods, only the *Class Vector* method enables latent steering without requiring any additional training.

We note that *Task Vector* [29] is not included as a baseline in our main experiments, since it operates at the task-wide level and is unsuitable for evaluating class-wise editing. For completeness, we additionally provide comparative results between task vectors and Class Vectors in the class unlearning setting in Tab. 17. Additional task-specific baselines are described in their respective sections, and full experimental details in Appendix §D.

4.2 Class Unlearning

Since ViT classifiers are exposed to a wide range of data during training, they naturally encode information across all classes. Practically, class unlearning is intended to modify classifier decision boundaries to prevent classification into specific categories, while minimizing unintended changes to non-target classes, often motivated by privacy or security concerns [52]. Existing class unlearning methods typically involve retraining the model or performing multi-stage post hoc edits, which are computationally expensive and risk inadvertently erasing features shared across classes [7].

To evaluate Class Vectors on class unlearning, we set $z_{\text{edit}} = \lambda \cdot \kappa_c$ with $\lambda = -1.5$ to effectively modify the model to unlearn adapted predictive rules for class c. We utilize the ViT-B/16 model for our experiments (See Appendix §E for results on other ViTs). For an additional baseline method, we retrain models with gradient ascent to the target class (*i.e.*, , NegGrad) following previous studies [37, 40]. For mapping, we utilize the single reference samples in the subset of each task's test set and evaluate on the remaining data, training only the final layer's layer normalization of the encoder. The first five labels in each task are used as the target (forget) class in each experiment. As shown in Tab. 1, Class Vectors demonstrate the most effective editing strategy for unlearning the target class (ACC_f) while preserving the performance of the non-target classes (ACC_r). In contrast, random vectors have minimal effects, indicating that Class Vectors point to meaningful directions in the latent space. Additionally, retraining struggles with limited data and trainable parameters.

4.3 Adapting to New Environment

Imbalanced training scenes often lead classifiers to reduced performance in specific contexts [65]. Thus, adapting open-hub classifiers to suit individual users' environments in an efficient manner is necessary in practice. Following a previous study [60], we examine a scenario where the model struggles to classify objects in snowy environments (Fig. 4). This occurs when the representation of snowy objects fails to accurately capture the object's features due to the ambiguous influence of snow.

Figure 4: Adapting the classifier to a snowy environment. Red text marks the misclassifications made by the original model, while blue text shows the correct predictions after classifier editing.

Class Vectors can effectively address this through high-level concept-based arithmetic operations. Specifically, our goal is to eliminate the snow features from the image representation. Therefore, the objective of z_{edit} is:

$$c_{\text{object}} = g(z^{\text{object}}, \theta^h) = g(z^c + z_{\text{edit}}), \theta^h)), \qquad (1)$$

where z_{edit} is designed to eliminate the representation activated by the model when snow is input, ensuring that the model focuses solely on the object.

We consider practical scenarios when only limited external samples are accessible. Namely, the editor can obtain a few images of the target object c_1 from external sources. We set z_{edit} as $\lambda(\kappa_{\text{snow}+c_1} - \kappa_{c_1})$. Here, $\kappa_{\text{snow}+c_1}$ denotes the representation adaptation of the model with snowy images. At a high level, this retains only the model adaptation related to snow. Editor can steer the model or map z_{edit} to satisfy Eq. 1, with $\lambda < 0$ used to suppress snow features. We evaluate on Snowy ImageNet [60] (7 classes, 20 images each), using 5 reference samples per class for mapping and testing on the remainder for mapping experiments. As an additional baseline, DirMatch [60] trains models to align images to target-class representations using external samples individually. We train only the final MLP and layer-norm in the transformer block, reporting mean ± standard deviation accuracy across classes. With $\lambda = -1.0$ and 4 external samples (Fig. 10), Class Vectors deliver a 10–20% improvement over the pre-edit classifier (Tab. 2a), underscoring their high-level interactions and effectiveness.

4.4 Defending Against Typography Attacks

Vision-language pretrained models like CLIP have exhibited vulnerabilities to typography attacks, where the text in an image leads to misclassification of the model [16]. Thus, mitigating typographic attack risk is crucial in safety-critical domains, such as medical imaging, before deploying classifiers. Similar to §4.3, given an object with text written on it, the representation is expected to become

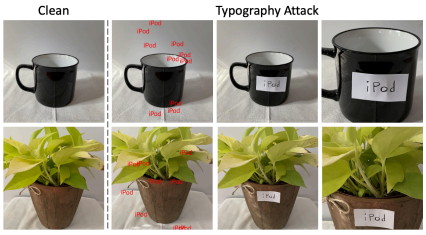
Table 2: Results on classifier editing with Class Vectors in two scenarios: (a) adapting to a new distribution (snowy environments) and (b) defending against typographic attacks.

Method	Average (↑)		
	ViT-B/16	ViT-B/32	ViT-L/14
Pretrained	55.2±24.6	53.4±29.9	60.2±22.5
Retrained	55.8±48.4	55.8±48.5	75.3±15.5
Random Vector	26.2±23.0	16.2±22.7	49.7±26.1
DirMatch	72.0±22.8	73.9±23.5	74.6±16.4
Class Vector	69.7±2.6	72.2±3.8	71.3±3.2
Class Vector†	72.7±21.4	76.2±21.2	78.3±16.9

(a)

Method	Average (↑)		
	ViT-B/16	ViT-B/32	ViT-L/14
Pretrained (Clean)	75.0±38.2	100±0.0	100±0.0
Pretrained (Attack)	48.9±38.2	76.7±33.1	38.9±19.4
Retrained	80.0±34.2	66.7±47.1	98.8±2.5
Random Vector	41.1±30.7	74.4±38.6	33.3±21.4
DirMatch	97.7±3.1	91.1±8.3	87.8±13.0
Class Vector	88.9±22.0	98.9±2.5	93.3±7.7
Class Vector†	98.9±2.5	99.0±2.5	93.3±6.7

(b)



(a)

(b)

Figure 5: Visual examples for the scenarios in §4.4 and §4.5: (a) examples of typographic attacks that cause the model to misclassify inputs as iPod. (b) optimized backdoor triggers on traffic-sign images.

z^{object} after classifier editing, allowing the deterministic rules of the model to focus solely on the object for accurate recognition. Thus, our goal aligns with that described in Eq. 1. Practically, the editor can easily generate small set of augmented samples by directly adding text to objects or performing data augmentation. Using them, the Class Vectors $\kappa_{\text{text+object}}$ for text-affected images and κ_{object} for clean images can be derived by feeding each image sets into the model. Finally, we define $z_{\text{edit}} = \lambda(\kappa_{\text{text+object}} - \kappa_{\text{object}})$, where z_{edit} removes adaptations from text-object images at a high-level, isolating the model's clean object representations.

Following a previous study [60], we utilize web-sourced image sets with 6 classes from ImageNet that include both clean and text with objects. Each image is augmented into 15 images per class (Fig. 5a). For mapping, we use a total of 4 reference images for training and evaluate on the remaining images. We use the same baselines as §4.3, setting $\lambda = -1.5$. For DirMatch, text-augmented images are trained to directly align with clean image representations. Results in Tab. 2b demonstrate that the Class Vector effectively defends against typographic attacks, achieving average accuracies on par with or exceeding clean-image performance across models.

4.5 Adversarial Trigger Optimizations

Class Vectors can also be utilized for backdoor attacks, which alter a model's logic using adversarial triggers such as small patches [20] or imperceptible noise [85], typically optimized by training the classifier to misclassify triggered images [82]. However, primary limitations of these approaches are the requirement for large amounts of triggered samples and full access to the training process to adjust the model's weights. We consider a scenario where an editor (*i.e.*, attacker) aims to mislead a classifier into misclassifying specific classes, without altering the model's weights. The attacker knows the classifier's architecture but lacks access to the user's model or training process. With knowledge of the architecture, the attacker can acquire a model with the same design from an open hubs. By embedding the intended representation into a trigger patch or an invisible trigger using the same model architecture, the attacker can cause the classifier to misclassify any object or scene where the pattern appears, whether on the object itself or attached to the camera lens.

To effectively embed malicious representations into the trigger, the attacker aims to optimize the initialized trigger x_{trigger}'s representation into $z_{\text{edit}} = \lambda \cdot (\kappa_{c_2} - \kappa_{c_1})$ such that when x_{trigger} is attached to an image, the classifier misclassifies c_1 as c_2. Unlike previous sections, z_{edit} is mapped to the pixel

space by training the pixels in x_{trigger} (Algorithm 4):

$$x_{\text{trigger}} = \underset{x_{\text{trigger}}}{\operatorname{argmin}} \, \| f(x_{\text{trigger}}, \theta_{\text{ft}}^{e}) - z_{\text{edit}} \|^2. \tag{2}$$

Note that, because the backdoor triggers are optimized, latent-space steering cannot be applied in this scenario. Class Vectors are evaluated across real-world tasks, including GTSRB, RESISC45, and SVHN. For mapping, 30, 10, and 200 samples are selected from each test set, with performance assessed on the remaining data. We additionally include BadNet [20] for baselines, where weights are manipulated with unlearnable triggered images to classify into the destination class.

We measure Attack Success Rate (ASR) on triggered images and Clean Accuracy (CA) on clean images, averaged across tasks. For DirMatch, the triggers are optimized directly toward the destination representation of target class. The scaling coefficient λ for z_{edit} is set to 1.5 for small patches, using 0.8% of total pixels and 1.0 for invisible noise by default. All experiments use ViT-B/32 (see Appendix §E for results on other ViTs). Tab. 3 shows that Class Vectors achieve high ASR, achieving high effectiveness without modifying model weights. As shown in Fig. 5b, triggers are optimized to be either very small or stealthy, making them imperceptible.

Table 3: Results on backdoor attacks with optimized triggers.

Method	**Small Patch**		**Invisible**	
	ASR (↑)	CA (↑)	ASR (↑)	CA (↑)
Pretrained	19.6±26.2	43.8±9.7	22.2±27.9	43.8±9.7
Finetuned	0.0±0.1	95.2±7.1	0.0±0.1	95.2±7.1
BadNet	100±0.0	10.0±5.0	100±0.0	9.3±7.4
Random Vector	0.1±0.1	95.2±7.1	39.4±55.8	95.2±7.1
DirMatch	96.5±4.7	95.2±7.1	96.8±4.5	95.2±7.1
Class Vector†	99.8±2.8	95.2±7.1	99.0±1.4	95.2±7.1

Table 4: Results of class unlearning on MNIST using ResNet18, ResNet50, and ConvNeXT-Tiny.

Method	**ResNet18**		**ResNet50**		**ConvNeXT-Tiny**	
	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)
Retrained	99.8	99.5	89.9	99.6	78.2	99.4
NegGrad	14.2	97.2	1.0	95.0	0.0	11.2
Random Vector	99.7	99.4	99.5	99.4	99.5	99.3
Class Vector	2.2	97.0	11.5	84.2	**0.0**	95.3
Class Vector†	**0.0**	**99.4**	**0.0**	**99.1**	**0.0**	**99.1**

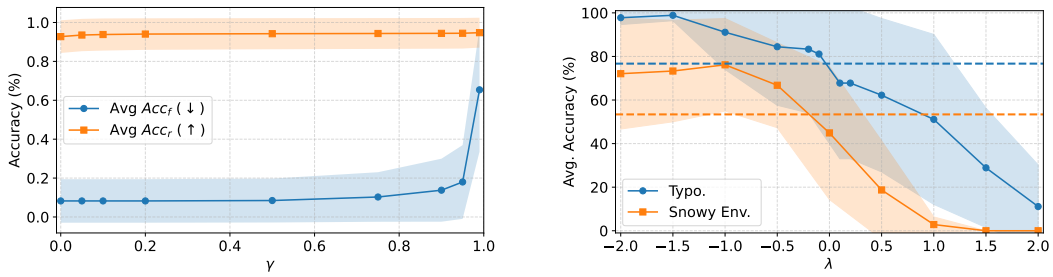
Table 5: Class unlearning with BERT-Base.

Method	**AG-NEWS**		**DBPedia-14**		**20-Newsgroups**	
	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)
Retrained	71.9	89.3	98.4	99.0	59.8	66.5
NegGrad	0.0	48.9	0.0	93.8	0.0	47.0
Random Vector	93.8	94.3	98.6	99.1	62.9	67.9
Class Vector	0.0	93.2	0.0	96.9	0.0	57.9
Class Vector†	**3.2**	**94.4**	**0.0**	**99.1**	**0.0**	**63.8**

4.6 In-Depth Analysis

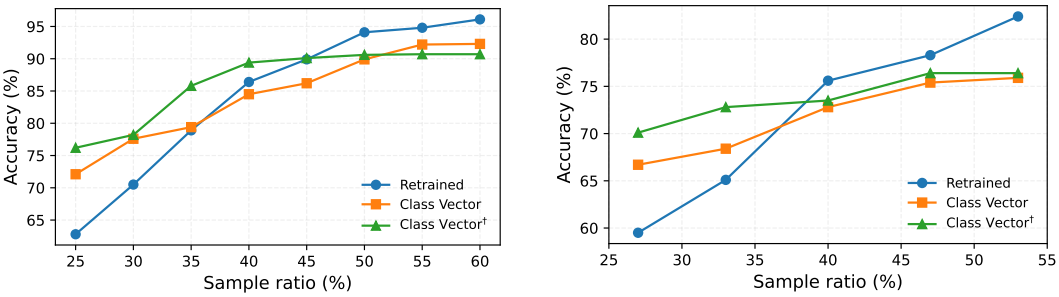
Model Generality across Architectures. To examine the architectural generality of Class Vectors, we extend our analysis beyond ViT encoders to convolutional and language models, including ResNet18, ResNet50, ConvNeXT-Tiny [45], and BERT-Base [12]. For each model, Class Vectors are derived from the pretrained and fine-tuned representations and applied to the class unlearning setting, targeting the first class in the dataset. As shown in Tab. 4 and Tab. 5, Class Vector maintains strong forgetting performance while preserving non-target accuracy across all architectures. Notably, both the latent-space variant (*Class Vector*) and its weight-space mapping (*Class Vector*†) exhibit consistent gains compared to gradient-based or random baselines, confirming that Class Vectors capture transferable, semantically meaningful directions independent of network type.

Impact of threshold in latent space steering. In the latent-space steering method, Class Vector injection is applied only to inputs whose cosine similarity to the target feature exceeds the threshold γ. To examine the impact of γ, we perform a systematic γ sweep accompanied by class-unlearning



(a) (b)

Figure 6: In-depth analysis: (a) Effect of the cosine-similarity threshold γ on class-unlearning clean accuracy; (b) Effect of the scaling coefficient λ on controllable editing. Horizontal dashed lines denote pretrained model performance.



(a) (b)

Figure 7: Effect of sample ratio on editing performance. Each curve shows how model accuracy varies as the proportion of available samples increases for (a) Snowy ImageNet and (b) Typo Attack.

evaluations to assess both editing efficacy and collateral impact. In Fig. 6a, editing remains effective for $\gamma \in [0.0, 0.5]$, and, even at $\gamma = 0$, we observe that edits retain independence, owing to the independence of Class Vectors (Theorem 3.3).

Impact of scaling coefficient. The scaling coefficient λ determines both the strength and direction of an edit. To make editing intuitive for non-experts, λ should yield predictable, controllable outcomes. For instance, in defense against typographic attacks, a more negative λ aggressively removes the learned "iPod" adaptation, while in other cases a milder edit suffices. We sweep λ in the snowy-environment and typography-attack (Fig. 6b) scenarios, observing clear trends: negative values erase snow features and restore correct predictions, whereas positive values amplify them and degrade performance. This demonstrates that λ can be tuned reliably based on high-level editing goals.

Impact of sample size. Retraining-based methods improve with more samples, but such abundance is rare in real deployments. To test scalability, we varied the sample ratio—the portion of available target data—in Snowy ImageNet and Typo Attack (Fig. 7a, 7b). With less than 30% of data, both Class Vector and Class Vector† outperform retraining, which only catches up after 35%. This highlights Class Vector's strong data efficiency, leveraging latent semantic directions instead of full parameter optimization, and maintaining competitiveness even under low-data regimes.

5 Conclusion and Future Work

We have introduced Class Vectors, which capture class-specific representation adaptations during training. Open-hub models can leverage Class Vectors to modify predictive rules for task-specific personalization. Our analysis of their linearity and independence, supported by extensive experiments, highlights their potential for efficient and interpretable classifier editing across diverse applications. While our work primarily focuses on image classification, we anticipate future extensions of Class Vectors to natural language processing (NLP) and generative models, including Large Language Models (LLMs) and image generative models.

Acknowledgements

This work was supported in part by National Research Foundation of Korea (NRF) grant (RS-2025-00560762), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant (RS-2025-02263754, RS-2025-25442338, IITP-2025-RS-2024-00397085, RS-2021-II211343). This research was also conducted as part of the Sovereign AI Foundation Model Project (Data Track, 2025-AI Data-wi43), organized by the Ministry of Science and ICT (MSIT) and supported by the National Information Society Agency (NIA). J. Do is with ASRI, Seoul National University.

References

[1] A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

[2] O. Avrahami, O. Fried, and D. Lischinski. Blended latent diffusion. *ACM transactions on graphics (TOG)*, 42(4):1–11, 2023.

[3] S. Balasubramanian, S. Basu, and S. Feizi. Decomposing and interpreting image representations via text in vits beyond clip. *arXiv preprint arXiv:2406.01583*, 2024.

[4] D. Bau, S. Liu, T. Wang, J.-Y. Zhu, and A. Torralba. Rewriting a deep generative model. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 351–369. Springer, 2020.

[5] L. Beyer, O. J. Hénaff, A. Kolesnikov, X. Zhai, and A. v. d. Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[6] R. Bhardwaj, D. D. Anh, and S. Poria. Language models are homer simpson! safety re-alignment of fine-tuned language models through task arithmetic. *arXiv preprint arXiv:2402.11746*, 2024.

[7] W. Chang, T. Zhu, H. Xu, W. Liu, and W. Zhou. Class machine unlearning for complex data via concepts inference and data poisoning. *arXiv preprint arXiv:2405.15662*, 2024.

[8] Z. Chen, Y. Bei, and C. Rudin. Concept whitening for interpretable image recognition. *Nature Machine Intelligence*, 2(12):772–782, 2020.

[9] G. Cheng, J. Han, and X. Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

[10] R. Chitale, A. Vaidya, A. Kane, and A. Ghotkar. Task arithmetic with lora for continual learning. *arXiv preprint arXiv:2311.02428*, 2023.

[11] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 3606–3613, 2014.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[13] A. Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *Proc. of Int'l Conf. on Learning Representations (ICLR)* , 2021.

[14] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, B. Tran, and A. Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019.

[15] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 3259–3269. PMLR, 2020.

[16] G. Goh, N. Cammarata, C. Voss, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

[17] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[18] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *Proc. of Int'l Conf. on Learning Representations (ICLR)* , 2015.

[19] Y. Goyal, A. Feder, U. Shalit, and B. Kim. Explaining classifiers with causal concept effect (cace). *arXiv preprint arXiv:1907.07165*, 2019.

[20] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[21] H. A. A. K. Hammoud, U. Michieli, F. Pizzati, P. Torr, A. Bibi, B. Ghanem, and M. Ozay. Model merging and safety alignment: One bad model spoils the bunch. *arXiv preprint arXiv:2406.14563*, 2024.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[23] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009, 2022.

[24] P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[25] R. Hendel, M. Geva, and A. Globerson. In-context learning creates task vectors. *arXiv preprint arXiv:2310.15916*, 2023.

[26] J. Hinns and D. Martens. Exposing image classifier shortcuts with counterfactual frequency (cof) tables. *arXiv preprint arXiv:2405.15661*, 2024.

[27] C. Huang, P. Ye, T. Chen, T. He, X. Yue, and W. Ouyang. Emr-merging: Tuning-free high-performance model merging. *arXiv preprint arXiv:2405.17461*, 2024.

[28] G. Ilharco, M. Wortsman, S. Y. Gadre, S. Song, H. Hajishirzi, S. Kornblith, A. Farhadi, and L. Schmidt. Patching open-vocabulary models by interpolating weights. *Proc. of Neural Information Processing Systems (NeurIPS)*, 35:29262–29277, 2022.

[29] G. Ilharco, M. T. Ribeiro, M. Wortsman, S. Gururangan, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. *Proc. of Int'l Conf. on Learning Representations (ICLR)* , 2023.

[30] H. Karimi, T. Derr, and J. Tang. Characterizing the decision boundary of deep neural networks. *IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2019.

[31] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila. Alias-free generative adversarial networks. *Proc. of Neural Information Processing Systems (NeurIPS)*, 34:852–863, 2021.

[32] A. Kasirzadeh and I. Gabriel. In conversation with artificial intelligence: aligning language models with human values. *Philosophy & Technology*, 36(2):27, 2023.

[33] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 2668–2677. PMLR, 2018.

[34] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 2668–2677. PMLR, 2018.

[35] H. E. Kim, A. Cosa-Linan, N. Santhanam, M. Jannesari, M. E. Maros, and T. Ganslandt. Transfer learning for medical image classification: a literature review. *BMC medical imaging*, 22(1):69, 2022.

[36] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[37] S. Kodge, G. Saha, and K. Roy. Deep unlearning: Fast and efficient gradient-free class forgetting. *Trans. on Machine Learning Research (TMLR)*, 2024.

[38] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images.(2009), 2009.

[39] A. Kumar, T. Ma, and P. Liang. Understanding self-training for gradual domain adaptation. In *International conference on machine learning*, pages 5468–5479. PMLR, 2020.

[40] M. Kurmanji, P. Triantafillou, J. Hayes, and E. Triantafillou. Towards unbounded machine unlearning. *Proc. of Neural Information Processing Systems (NeurIPS)*, 36, 2024.

[41] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[42] E. Liu. Leveraging intermediate neural collapse with simplex etfs for efficient deep neural networks. *arXiv preprint arXiv:2412.00884*, 2024.

[43] S. Liu, H. Ye, L. Xing, and J. Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. *arXiv preprint arXiv:2311.06668*, 2023.

[44] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proc. of Int'l Conf. on Computer Vision (ICCV)*, pages 10012–10022, 2021.

[45] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 11976–11986, 2022.

[46] K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual associations in gpt. *Proc. of Neural Information Processing Systems (NeurIPS)*, 35:17359–17372, 2022.

[47] J. Merullo, C. Eickhoff, and E. Pavlick. Language models implement simple word2vec-style vector arithmetic. *arXiv preprint arXiv:2305.16130*, 2023.

[48] D. Mickisch, F. Assion, F. Greßner, W. Günther, and M. Motta. Understanding the decision boundary of deep neural networks: An empirical study. *arXiv preprint arXiv:2002.01810*, 2020.

[49] S. I. Mirzadeh, M. Farajtabar, D. Gorur, R. Pascanu, and H. Ghasemzadeh. Linear mode connectivity in multitask and continual learning. *Proc. of Int'l Conf. on Learning Representations (ICLR)* , 2021.

[50] E. Mitchell, C. Lin, A. Bosselut, C. D. Manning, and C. Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR, 2022.

[51] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, 2011.

[52] T. T. Nguyen, T. T. Huynh, Z. Ren, P. L. Nguyen, A. W.-C. Liew, H. Yin, and Q. V. H. Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

[53] G. Ortiz-Jimenez, A. Favero, and P. Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Proc. of Neural Information Processing Systems (NeurIPS)*, 36, 2024.

[54] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[55] V. Papyan, X. Han, and D. L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40): 24652–24663, 2020.

[56] N. Park and S. Kim. How do vision transformers work? *arXiv preprint arXiv:2202.06709*, 2022.

[57] R. K. Rachman, D. R. I. M. Setiadi, A. Susanto, K. Nugroho, and H. M. M. Islam. Enhanced vision transformer and transfer learning approach to improve rice disease recognition. *Journal of Computing Theories and Applications*, 1(4):446–460, 2024.

[58] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[59] M. T. Ribeiro and S. Lundberg. Adaptive testing and debugging of nlp models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3253–3267, 2022.

[60] S. Santurkar, D. Tsipras, M. Elango, D. Bau, A. Torralba, and A. Madry. Editing a classifier by rewriting its prediction rules. *Advances in Neural Information Processing Systems*, 34: 23359–23373, 2021.

[61] D. Shirafuji, M. Takenaka, and S. Taguchi. Bias vector: Mitigating biases in language models with task arithmetic approach. *arXiv preprint arXiv:2412.11679*, 2024.

[62] M. Sotoudeh and A. V. Thakur. Provable repair of deep neural networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 588–603, 2021.

[63] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[64] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *Trans. on Machine Learning Research (TMLR)*, 2021.

[65] J. Tanha, Y. Abdi, N. Samadi, N. Razzaghi, and M. Asadpour. Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big data*, 7:1–47, 2020.

[66] M. Taraghi, G. Dorcelus, A. Foundjem, F. Tambon, and F. Khomh. Deep learning model reuse in the huggingface community: Challenges, benefit and trends. *arXiv preprint arXiv:2401.13177*, 2024.

[67] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.

[68] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 10347–10357. PMLR, 2021.

[69] T. P. Van, T. M. Nguyen, N. N. Tran, H. V. Nguyen, L. B. Doan, H. Q. Dao, and T. T. Minh. Interpreting the latent space of generative adversarial networks using supervised learning. In *2020 International Conference on Advanced Computing and Applications (ACOMP)*, pages 49–54. IEEE, 2020.

[70] B. Wallace, M. Dang, R. Rafailov, L. Zhou, A. Lou, S. Purushwalkam, S. Ermon, C. Xiong, S. Joty, and N. Naik. Diffusion model alignment using direct preference optimization. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 8228–8238, 2024.

[71] C. Wang and P. Golland. Interpolating between images with diffusion models. *Proc. of Int'l Conf. on Machine Learning (ICML)*, 2023.

[72] S. Wang, Y. Zhu, H. Liu, Z. Zheng, C. Chen, and J. Li. Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 57(3):1–37, 2024.

[73] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022.

[74] X. Wu, S. Huang, and F. Wei. Mixture of lora experts. *Proc. of Int'l Conf. on Learning Representations (ICLR)* , 2024.

[75] J. Xu and H. Liu. Quantifying the variability collapse of neural networks. In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 38535–38550. PMLR, 2023.

[76] P. Yadav, D. Tam, L. Choshen, C. A. Raffel, and M. Bansal. Ties-merging: Resolving interference when merging models. *Proc. of Neural Information Processing Systems (NeurIPS)*, 36, 2023.

[77] P. Yadav, T. Vu, J. Lai, A. Chronopoulou, M. Faruqui, M. Bansal, and T. Munkhdalai. What matters for model merging at scale? *arXiv preprint arXiv:2410.03617*, 2024.

[78] E. Yang, Z. Wang, L. Shen, S. Liu, G. Guo, X. Wang, and D. Tao. Adamerging: Adaptive model merging for multi-task learning. *arXiv preprint arXiv:2310.02575*, 2023.

[79] E. Yang, L. Shen, Z. Wang, G. Guo, X. Chen, X. Wang, and D. Tao. Representation surgery for multi-task model merging. *arXiv preprint arXiv:2402.02705*, 2024.

[80] Y. Yang, L.-K. Huang, S. Chen, K. Ma, and Y. Wei. Learning where to edit vision transformers. *Proc. of Neural Information Processing Systems (NeurIPS)*, 2024.

[81] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

[82] Y. Yuan, R. Kong, S. Xie, Y. Li, and Y. Liu. Patchbackdoor: Backdoor attack against deep neural networks without model modification. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9134–9142, 2023.

[83] J. Zhang, J. Liu, J. He, et al. Composing parameter-efficient modules with arithmetic operation. *Proc. of Neural Information Processing Systems (NeurIPS)*, 36:12589–12610, 2023.

[84] W. Zhang, Q. Yang, and Y. Geng. A survey of anomaly detection methods in networks. In *2009 International Symposium on Computer Network and Multimedia Technology*, pages 1–3. IEEE, 2009.

[85] N. Zhong, Z. Qian, and X. Zhang. Imperceptible backdoor attack: From input space to feature representation. *arXiv preprint arXiv:2205.03190*, 2022.

[86] Z. Zhou, Y. Yang, X. Yang, J. Yan, and W. Hu. Going beyond linear mode connectivity: The layerwise linear feature connectivity. *Proc. of Neural Information Processing Systems (NeurIPS)*, 36:60853–60877, 2023.

[87] Z. Zhou, Z. Chen, Y. Chen, B. Zhang, and J. Yan. On the emergence of cross-task linearity in the pretraining-finetuning paradigm. *arXiv preprint arXiv:2402.03660*, 2024.

Supplementary Material

Contents

A Broader Discussion **17**

A.1 Extended Related Work 17

A.2 Limitations 17

A.3 Ethic Statement 17

A.4 Licenses 17

B Algorithms **18**

C Supplementary Theoretical Analysis **19**

C.1 Theoretical Justification of Class Vectors 19

C.2 Existence of a Mapping Function 20

C.3 Class Vectors Preserve Inter-Class Orthogonality 21

D Experimental Details **21**

D.1 Class Configurations 21

D.2 Task Details 22

D.3 Training Details 23

E Additional Experiments **25**

F Mapping Sensitivities **26**

G Computational Analysis **26**

H Supplementary Figures **27**

A Broader Discussion

A.1 Extended Related Work

Model interventions Model intervention aims to adapt trained models to new knowledge for specific user needs. Retraining an entire model is time-consuming and data-intensive, driving the development of more efficient methods that use limited data. These include model alignment [1, 54, 70, 32], debugging [59], and editing [29, 60, 50, 4]. Model intervention in Computer Vision (CV) remains limited. Previous work propose editing generative adversarial networks (GANs) by identifying and modifying specific locations [4], while this approach is extended to classifiers for debugging errors by mapping new rules to existing ones [60]. However, these methods require editing locations or additional data. More recent meta-learning-based approach [80] address this but remain computationally demanding.

Latent representations Latent space interpolation in generative models aims to blend the styles of different images by navigating their latent vectors. This approach has been widely explored in GANs [69, 31] and diffusion models [71, 2], while its application to classifiers remain largely unexplored. Meanwhile, recent efforts to explain deep neural networks have focused on linking models' internal processes to high-level concepts, such as analyzing human-understandable features or testing their influence on predictions (*i.e.*, decomposability) [8, 14, 34, 3, 19]. Building on these ideas, we investigate representation adaptation in the latent space, revealing properties and enabling effective, high-level editing.

A.2 Limitations

While Class Vectors enable efficient and interpretable classifier editing across a variety of tasks, our method exhibits several limitations. First, the approach assumes that class representations are well-structured and approximately linearly separable in the latent space. This assumption is empirically supported by CTL and Neural Collapse, but may not hold in scenarios with high intra-class variance, noisy labels, or long-tailed distributions. Second, the method currently focuses on single-label classification tasks where each example is associated with a single semantic class. Extending Class Vector-based editing to multi-label or hierarchical classification, where class boundaries are less distinct and often overlapping, remains an open challenge. In addition, the latent-space steering method relies on access to the centroid representation of each class, which may not be readily available in privacy-constrained or black-box settings. Similarly, the weight-space mapping method requires updating a small subset of layers with a few reference samples, which still assumes partial access to model internals. This limits the applicability of our method in fully closed-source environments. Despite these limitations, our work lays the foundation for structured classifier editing and invites further research into expanding its scope to more complex, unconstrained settings.

A.3 Ethic Statement

This work does not involve research with human participants, sensitive data, or personally identifiable information. All experiments were conducted using publicly available pretrained models and open-source datasets. We include limited web-sourced or user-generated imagery (e.g., for typographic attack evaluation), and such images are used solely for non-commercial, academic research purposes under fair use or Creative Commons–compliant terms. While our method enables editing of classifiers for beneficial purposes such as unlearning and robustness, we recognize that similar techniques may be repurposed for malicious intent, such as backdoor trigger optimization. To mitigate such risks, we release code and dataset under a non-commercial license (CC-BY-NC-SA 4.0) and emphasize that practical deployment of editing techniques should be preceded by careful threat modeling and access control.

A.4 Licenses

We plan to release our code under the Apache 2.0 license.

B Algorithms

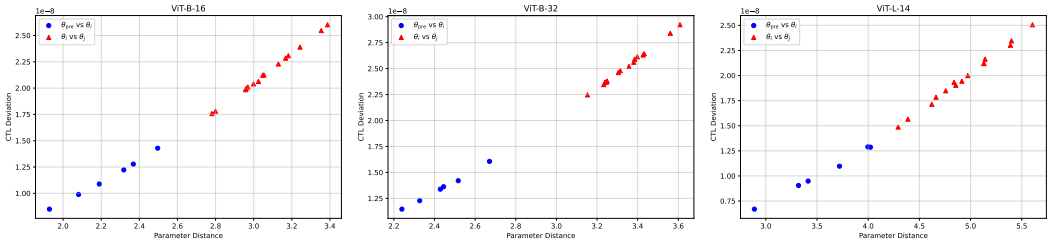
We present algorithms that leverage Class Vectors by mapping them into non-latent spaces. For latent space steering, refer to Algorithm 1. Specifically, we introduce two approaches: (1) standard mapping via an encoder, and (2) pixel-space mapping for adversarial trigger optimization.

Algorithm 3 Pseudocode for optimizing classifier encoder with Class Vector

Require: Classifier encoder ($f(\cdot, \theta^e)$ and trainable layers), Dataset $\mathcal{X}_{\text{task}}$, Editing vector z_{edit}, Number of epochs T, Target class c, Learning rate η
Ensure: Edited encoder weight θ^e_{edit} such that $f(x \in \mathcal{X}_{\text{task}}, \theta^{\text{e}}) = z^c + z_{\text{edit}}$
Freeze all layers of θ^{e} **except** the final trainable layers
$\mathbf{r}_{\text{target}} \leftarrow$ None
Initialize list $\mathcal{R} \leftarrow [\,]$ {to store class-c representations}
for epoch $= 1$ to T **do**
  for each mini-batch (X, Y) in $\mathcal{X}$ **do**
    $\mathbf{r} \leftarrow f(X, \theta^{\text{e}})$ {Encoder representation}
    if $\mathbf{r}_{\text{target}} =$ None **then**
      Collect $\mathbf{r}_c \leftarrow \mathbf{r}[Y = c]$, **Append** $\mathbf{r}_c$ to $\mathcal{R}$ {Representations for target class}
      if enough class-c reps in $\mathcal{R}$ **then**
        $\overline{\mathbf{r}} \leftarrow \text{mean}(\mathcal{R})$ {Average representation **when reference sample number is satisfied**}
        $\mathbf{r}_{\text{target}} \leftarrow \overline{\mathbf{r}} + z_{\text{edit}}$ {Add editing vector}
      end if
      Continue to next mini-batch if $\mathbf{r}_{\text{target}} =$ None
    end if
    Filter $\widetilde{\mathbf{r}} \leftarrow \mathbf{r}[Y = c]$ {Only align class-c representations}
    Compute alignment loss: $\ell = \|\widetilde{\mathbf{r}} - \mathbf{r}_{\text{target}}\|^2$
    Backpropagation with ℓ
  end for
end for
return M {Edited encoder weight θ^e_{edit}}

Algorithm 4 Pseudocode for optimizing adversarial trigger with Class Vector

Require: Classifier encoder $f(\cdot, \theta^e)$, Trainable trigger x_{trigger}, Dataset $\mathcal{X}_{\text{task}}$, Editing vector z_{edit}, Number of epochs T, Target class c, Learning rate η
Ensure: Edited encoder weight θ^e_{edit} such that $f(x + x_{\text{trigger}}, \theta^{\text{e}}) = z^c + z_{\text{edit}}$
$\mathbf{r}_{\text{target}} \leftarrow$ None
Initialize list $\mathcal{R} \leftarrow [\,]$ {to store class-c representations}
for epoch $= 1$ to T **do**
  for each mini-batch (X, Y) in $\mathcal{X}$ **do**
    $\mathbf{r} \leftarrow f(X + x_{\text{trigger}}, \theta^{\text{e}})$ {Encoder representation}
    if $\mathbf{r}_{\text{target}} =$ None **then**
      Collect $\mathbf{r}_c \leftarrow \mathbf{r}[Y = c]$, **Append** $\mathbf{r}_c$ to $\mathcal{R}$ {Representations for target class}
      if enough class-c reps in $\mathcal{R}$ **then**
        $\overline{\mathbf{r}} \leftarrow \text{mean}(\mathcal{R})$ {Average representation **when reference sample number is satisfied**}
        $\mathbf{r}_{\text{target}} \leftarrow \overline{\mathbf{r}} + z_{\text{edit}}$ {Add editing vector}
      end if
      Continue to next mini-batch if $\mathbf{r}_{\text{target}} =$ None
    end if
    Filter $\widetilde{\mathbf{r}} \leftarrow \mathbf{r}[Y = c]$ {Only align class-c representations}
    Compute alignment loss: $\ell = \|\widetilde{\mathbf{r}} - \mathbf{r}_{\text{target}}\|^2$
    Backpropagation with ℓ
  end for
end for
return x_{trigger} {Edited trigger x_{trigger}}



(a) ViT-B/16 (b) ViT-B/32 (c) ViT-L/14

Figure 8: Empirical validation of Theorem 3.1: CTL deviation increases with parameter distance. Blue markers denote interpolation between pretrained and fine-tuned weights ($\theta_{\text{pre}} \leftrightarrow \theta_i$); red markers denote interpolation between different fine-tuned weights ($\theta_i \leftrightarrow \theta_j$).

C Supplementary Theoretical Analysis

In this section, we provide theoretical analyses, including proofs for the theorems, as well as empirical evidence supporting them.

C.1 Theoretical Justification of Class Vectors

Here, we prove Theorem 3.1 and provide empirical results supporting the assumptions illustrated in the figure.

Theorem 3.1 (CTL between pretrained and fine-tuned weights)**.** *Suppose the function $f : \mathbb{R}^p \to \mathbb{R}$ is three-times differentiable on an open convex set $\Theta \subset \mathbb{R}^p$, and that its Hessian is spectrally bounded at every $\theta_0 \in \Theta$: $\lambda_{\min} \leq \left\|\nabla^2 f(\theta_0)\right\| \leq \lambda_{\max}$* [87]*. Let θ_{pre} be the pre-trained weights and θ_i, θ_j two fine-tuned weights that satisfy CTL. Define*

$$\delta_{\text{pre},i} = \left|f(\alpha\theta_{\text{pre}} + (1-\alpha)\theta_i) - \left((1-\alpha)f(\theta_{\text{pre}}) + \alpha f(\theta_i)\right)\right|,$$
$$\delta_{i,j} = \left|f(\alpha\theta_i + (1-\alpha)\theta_j) - \left((1-\alpha)f(\theta_i) + \alpha f(\theta_j)\right)\right|.$$

If $\|\theta_i - \theta_{\text{pre}}\| < \|\theta_i - \theta_j\|$, then $\delta_{\text{pre},i} < \delta_{i,j}$: the segment from θ_{pre} to θ_i shows strictly smaller CTL deviation, hence is more linear*, than the segment between two fine-tuned solutions $\theta_i \to \theta_j$.*

Proof. From Theorem 5.1 in *Zhou et al.* [87], for any pair $\theta_a, \theta_b \in \Theta$, if $f : \mathbb{R}^p \to \mathbb{R}$ is three-times differentiable on open convex domain Θ, and its Hessian satisfies the spectral bound

$$\lambda_{\min} \leq \|\nabla^2 f(\theta)\| \leq \lambda_{\max}, \quad \forall \theta \in \Theta,$$

then the CTL deviation is bounded by:

$$\delta_{\theta_a,\theta_b} = |f(\alpha\theta_a + (1-\alpha)\theta_b) - [\alpha f(\theta_a) + (1-\alpha)f(\theta_b)]| \leq \frac{\alpha(1-\alpha)}{2}\lambda_{\max}\|\theta_a - \theta_b\|^2 + E,$$

where the remainder term $E = \mathcal{O}(\|\alpha\theta_a + (1-\alpha)\theta_b - \theta_0\|^3)$ vanishes as the interpolation point approaches θ_0.

Now fix $\theta_{\text{pre}}, \theta_i, \theta_j \in \Theta$, and assume:

$$\|\theta_i - \theta_{\text{pre}}\| < \|\theta_i - \theta_j\|.$$

Apply the bound to each deviation:

$$\delta_{\text{pre},i} \leq \frac{\alpha(1-\alpha)}{2}\lambda_{\max}\|\theta_i - \theta_{\text{pre}}\|^2 + E_1,$$
$$\delta_{i,j} \leq \frac{\alpha(1-\alpha)}{2}\lambda_{\max}\|\theta_i - \theta_j\|^2 + E_2,$$

for small remainder terms E_1, E_2 of order $\mathcal{O}(\|\cdot\|^3)$.

Because $\|\theta_i - \theta_{\text{pre}}\| < \|\theta_i - \theta_j\|$, it follows that:

$$\delta_{\text{pre},i} < \delta_{i,j} \quad \text{up to a cubic-order error.}$$

Therefore, the CTL deviation along the interpolation path from θ_{pre} to θ_i is strictly smaller than that between θ_i and θ_j in the small-distance regime. □

Fig. 8 illustrates the empirical validation of Theorem 3.1, which states that the CTL deviation between two parameter vectors is upper-bounded by a quadratic function of their Euclidean distance. CTL deviations are computed using a synthetic quadratic loss function as a surrogate for the true task loss. In particular, the deviation tends to be smaller when interpolating between a pretrained model and a fine-tuned model ($\theta_{\text{pre}} \leftrightarrow \theta_i$) than between two independently fine-tuned models ($\theta_i \leftrightarrow \theta_j$).

C.2 Existence of a Mapping Function

The mapping function ϕ_{edit} effectively exists in practical editing scenarios, such as small weight modifications within an overparameterized encoder. We demonstrate that KL-divergence–based mapping also performs effectively (Tab. 19). Furthermore, experiments across various learning rate configurations (Tab. 18) show that the mapping is robust to different optimization setups.

Theorem 3.2 (Existence of a Mapping)**.** *Let $\phi_{\text{edit}} : \mathbb{R}^m \to \mathbb{R}^{d_e}$ be any mapping that sends latent editing vectors to weight perturbations applied in the encoder's final layer or a small subset of layers. Under the assumption that these edits are sufficiently small and confined to that small subset of layers of an overparameterized encoder (e.g., a ViT) with $d_e \gg m$, there exist infinitely many distinct ϕ_{edit}.*

Proof. For any input x, if the editable–parameter perturbation w is sufficiently small, then

$$f(x; \theta + w) = f(x; \theta) + J_f(\theta)w + o(\|w\|),$$

so a first–order Taylor approximation around θ is valid. Let $J := J_f(\theta) \in \mathbb{R}^{m \times d_e}$ denote the Jacobian of the encoder output with respect to the editable parameters, evaluated at θ. Since edits are restricted to an overparameterised subset of layers with $d_e \gg m$, we assume $\text{rank}(J) = m$ (full row rank).

We seek a perturbation $w \in \mathbb{R}^{d_e}$ that realises a target change $z \in \mathbb{R}^m$ in the encoder output, i.e.

$$Jw = z.$$

Because J has full row rank, the matrix $JJ^\top \in \mathbb{R}^{m \times m}$ is invertible, and the Moore–Penrose pseudoinverse

$$J^\dagger := J^\top (JJ^\top)^{-1} \in \mathbb{R}^{d_e \times m}$$

satisfies $JJ^\dagger = I_m$. Hence one particular solution of (A) is

$$w_0(z) := J^\dagger z.$$

Let $\mathcal{N} := \ker J = \{v \in \mathbb{R}^{d_e} \mid Jv = 0\}$. Its dimension is $d_e - m > 0$, so the full solution set of (A) is the affine subspace

$$\mathcal{S}(z) = J^\dagger z + \ker J = \{\, J^\dagger z + v \mid v \in \ker J \,\}.$$

Since $\mathcal{N}$ is nontrivial, infinitely many distinct w satisfy $Jw = z$.

Now take any linear map $N \in \mathbb{R}^{d_e \times m}$ whose columns lie in $\ker J$ ($JN = 0$), and define

$$R := J^\dagger + N.$$

Then $JR = JJ^\dagger = I_m$, so for every $z \in \mathbb{R}^m$,

$$J(Rz) = z.$$

The mapping

$$\phi_{\text{edit}}(z) := Rz$$

thus provides a valid weight–space perturbation realising the desired edit. Varying N (or equivalently adding any vector in $\ker J$ to Rz) yields infinitely many distinct mappings $\phi_{\text{edit}} : \mathbb{R}^m \to \mathbb{R}^{d_e}$ that all satisfy $J\,\phi_{\text{edit}}(z) = z$.

Finally, since $\ker J$ is a linear subspace, for any $\varepsilon > 0$ we may rescale the target z (or equivalently R) to ensure $\|\phi_{\text{edit}}(z)\| \le \varepsilon$, so the perturbation remains sufficiently small while achieving the desired first–order effect. Therefore, under the stated conditions, there exist infinitely many sufficiently small weight–space mappings ϕ_{edit} that satisfy $J\,\phi_{\text{edit}}(z) = z$ for all $z \in \mathbb{R}^m$. □

C.3 Class Vectors Preserve Inter-Class Orthogonality

The independence of Class Vectors from other classes is a key property for effective localized editing. Based on Neural Collapse (NC) [55], we provide theoretical evidence supporting this claim.

Theorem 3.3 (Independence of Class Vectors)**.** *Suppose (i) the pretrained class embeddings collapse to a common mean $\bar{z}^{\mathrm{pre}}$, that is $z_c^{\mathrm{pre}} \approx \bar{z}^{\mathrm{pre}}$; (ii) after fine-tuning the embeddings follow a centre-shifted ETF form $z_c^{\mathrm{ft}} = \mu + u_c$ with $\sum_c u_c = 0$; and (iii) the global-shift $\|\mu - \bar{z}^{\mathrm{pre}}\|$ is negligible compared to the class-specific update $\|u_c\|$. Then, for any two distinct classes $c \neq c'$,*

$$\cos\big(\kappa_c,\, z_{c'}^{\mathrm{ft}}\big) \;\approx\; 0,$$

i.e. the Class Vector κ_c is approximately orthogonal to the fine-tuned embedding of every other class.

Proof. Define the editing vector $\kappa_c := z_c^{\mathrm{ft}} - z_c^{\mathrm{pre}}$. With Assumption (i) we write

$$z_c^{\mathrm{pre}} = \bar{z}^{\mathrm{pre}} + e_c, \qquad \text{where } \|e_c\| \ll \|u_c\|.$$

Hence

$$\kappa_c = (\mu + u_c) - (\bar{z}^{\mathrm{pre}} + e_c) = (\mu - \bar{z}^{\mathrm{pre}}) + u_c - e_c.$$

For a distinct class $c' \neq c$ the inner product becomes

$$\langle \kappa_c,\, z_{c'}^{\mathrm{ft}} \rangle = \langle \mu - \bar{z}^{\mathrm{pre}},\, \mu + u_{c'} \rangle + \langle u_c, u_{c'} \rangle - \langle e_c, \mu + u_{c'} \rangle.$$

Assumption (ii) implies $\sum_c u_c = 0$ and that $\{u_c\}$ form an equiangular tight frame, so $\langle u_c, u_{c'} \rangle = -\frac{\|u_c\|^2}{k-1}$ and $\langle \mu, u_{c'} \rangle = 0$ after choosing μ orthogonal to the span of the u_c. Assumption (iii) states $\|\mu - \bar{z}^{\mathrm{pre}}\| \ll \|u_c\|$, so $\langle \mu - \bar{z}^{\mathrm{pre}}, \mu + u_{c'} \rangle$ is negligible compared with $\|u_c\|^2$. Finally, $\|e_c\| \ll \|u_c\|$ makes the last term negligible. Collecting these bounds,

$$\big|\langle \kappa_c,\, z_{c'}^{\mathrm{ft}} \rangle\big| \;\approx\; \frac{\|u_c\|^2}{k-1},$$

which is small when the number of classes k is moderate to large. Since $\|\kappa_c\| \approx \|u_c\|$ and $\|z_{c'}^{\mathrm{ft}}\| \approx \sqrt{\|\mu\|^2 + \|u_{c'}\|^2} = \mathcal{O}(\|u_c\|)$, the cosine similarity satisfies

$$\cos\big(\kappa_c,\, z_{c'}^{\mathrm{ft}}\big) \;=\; \frac{\langle \kappa_c, z_{c'}^{\mathrm{ft}} \rangle}{\|\kappa_c\|\, \|z_{c'}^{\mathrm{ft}}\|} \;\approx\; \frac{1}{k-1} \;\approx\; 0,$$

establishing that each Class Vector κ_c is approximately orthogonal to every other fine-tuned embedding $z_{c'}^{\mathrm{ft}}$ for $c \neq c'$. □

We note that the Neural Collapse (NC) phenomenon emerges across diverse classifier architectures, data distributions, and training paradigms [55, 42, 75]. Here, we validate the underlying assumptions on ViT-B/32. In Fig. 9, we observe that the cosine similarity between pretrained class centroids, $\bar{z}^{\mathrm{pre}}$, is nearly 1. This indicates that the pretrained class embeddings collapse to a common mean, thereby validating Assumption (i). Furthermore, as shown in Tab. 6, the cosine similarities among class centroids closely match the theoretical value of $-1/(k-1)$, strongly supporting the hypothesis that the centroids form an equiangular tight frame (ETF) structure (Assumption (ii)). Furthermore, we show that the global shift in the mean representation across all classes is significantly smaller than class-wise updates, indicating that the editing process remains highly localized (Assumption (iii)). Fig. 9 further shows that the fine-tuned class representations are quasi-orthogonal, providing additional evidence for inter-class independence.

D Experimental Details

D.1 Class Configurations

Tab. 7 shows the class configurations used to evaluate class-vector properties. For each task, we consistently select the first two classes.

Table 6: Verification of Assumptions 2 and 3 across five tasks. Cosine similarity is computed across re-centered class embeddings (Assumption 2). The last column reports the deviation of global drift relative to class-specific update norms (Assumption 3).

Task	Cos.Sim. (Ass. 2)	Theoretical ETF Cos.Sim.	Global-Shift /Cls-update (Ass. 3)
DTD	−0.02	−0.02	0.00
EuroSAT	−0.10	−0.11	−0.12
GTSRB	−0.02	−0.02	−0.02
MNIST	−0.10	−0.11	−0.02
RESISC45	−0.02	−0.02	−0.07

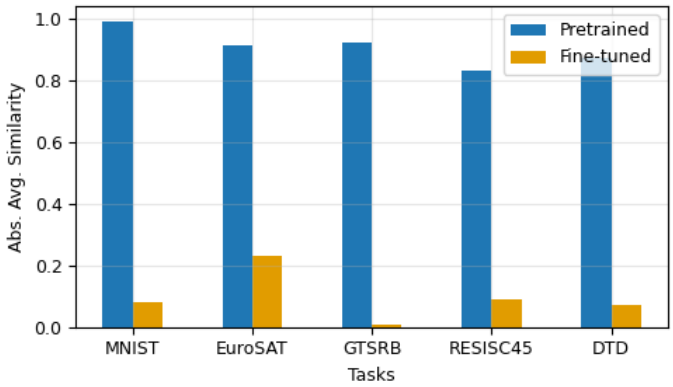


Figure 9: Comparison of cosine similarities across class representations within the pretrained and fine-tuned classifiers.

Table 7: Task overviews and target classes for §3.3

Task	# of classes	Target classes
MNIST	10	0,1
EuroSAT	10	Annual crop, Herbaceous Vegetation
GTSRB	43	20kph speed limit, 30kph speed limit 1
RESISC45	45	Airplane, Airport
DTD	47	Banded, Blotchy

D.2 Task Details

The fundamental properties of linearity and independence, and applications of Class Vectors in the context of unlearning are evaluated in §3.3 and §4.2, using six widely adopted image classification tasks, MNIST [41], EuroSAT [24], SVHN [51], GTSRB [63], RESISC45 [9] and DTD [11]. Additionally, we empirically justify the Class Vectors in both MLP and ResNet-18 architectures, using CIFAR-10 and CIFAR-100 datasets. All images are rescaled to image size 224×224. The details for each task are as follows:

- **MNIST** [41]: A handwritten digit classification task with 60,000 training images and 10,000 test images, categorized into 10 classes from 0 to 9.
- **EuroSAT** [24]: Land use and cover satellite image classification task, containing 13 spectral bands and 10 classes, with 16,000 training images and 5000 test images, 27,000 images in total with validation set.
- **SVHN** [51]: A real-world digit classification benchmark task containing a total of 630,000 images with 2,700 test data of house number plates.
- **GTSRB** [63]: Traffic sign classification task with 43 categories, containing 39,209 training data and 12,630 test data under varied lighting and complex backgrounds.

- **RESISC45** [9]: A benchmark for remote sensing scene classification task with 31,500 images across 45 distinct scene types.
- **DTD** [11]: Image texture classification task with 47 categories, containing a collection of 5,640 texture images, sourced from diverse real-world settings.
- **CIFAR10** [38]: The dataset is a 10-class classification task consisting of 60,000 images, with 6,000 images per class. It contains 50,000 training images and 10,000 test images.
- **CIFAR100** [38]: The dataset is a subset of the Tiny Images dataset and consists of 60,000 color images. The 100 classes are organized into 20 superclasses, with each class containing 600 images.

For evaluating Class Vectors in adapting to snowy environments (§4.3) and defending against typography attacks (§4.4), we use snowy ImageNet and clean and text-attached object images from previous study [60]. For the snowy environment adaptation scenario, we collect clean images for all classes from ImageNet, as shown in Fig. 10. For the defense against typography attacks, we augment images from all classes with: 1) digital text attachment, 2) rotation, random cropping, and color jittering (Fig. 11). The details for each task are as follows:

- **Snowy ImageNet**: A collection of images from ImageNet with snowy environments. It consists of 7 classes with 20 images per class, totaling 140 images.
- **Images for typography attack**: It consists of web-scraped images with 6 classes of indoor objects and text-attached images for each. There are 6 images of each object. We augment each sample into 13 images per class, thus making a total of 90 images, including both original clean and adversarial images.

D.3 Training Details

D.3.1 Class Vector Justification

In Fig. 2b, we justify Class Vectors using three architectures—ViT-B/32, an MLP, and ResNet-18—and train each as follows. The Vision Transformer (ViT-B/32) is fine-tuned per task for approximately 22 epochs with a learning rate of 1e-5 and a batch size of 128. For both the MLP and ResNet-18, we train on MNIST and CIFAR-10 with a learning rate of 1e-3, batch size 128 for 10 epochs, and on CIFAR-100 with a learning rate of 1e-4, batch size 512 for 300 epochs.

D.3.2 Mapping Editing Vectors

We utilize fully fine-tuned weights for the five tasks mentioned above from open-source repositories of Task Arithmetic [29][3]. We use pretrained ViTs to classify ImageNet classes in §4.3 and §4.4, as they are trained on ImageNet. For these tasks, we initialize the encoder and create the representation for all classes from the model to generate Class Vectors. All our experiments first design z_{edit}, then map it to the weight space (*i.e.*, ϕ_{edit}) to achieve classifier editing. The following are detailed training settings for mapping editing vectors. Refer to Algorithm 3 and Algorithm 4 for details on the *reference sample*. All our experiments are conducted on a single NVIDIA A100 GPU.

Exploring properties of Class Vectors §3 explores the fundamental properties of Class Vectors: linearity and independence. To evaluate the linearity of Class Vectors, we train z_{edit} for 15 epochs, using target classes within 1% of the test set, with a learning rate of 1.5e-2 and a single reference sample. For all experiments verifying independence, we train the model for 15 epochs on the 1% test set, as described above, with a learning rate of 5e-2 and one reference sample.

Class unlearning Tab. 8 shows the hyperparameters for class unlearning. We find the best hyperparameter by grid searching [3e-2, 4e-2, 5e-2, 6e-2]. The same hyperparameters with MNIST are applied to SVHN in the cross-task class unlearning scenario. The following are the hyperparameters we adopt to evaluate class unlearning:

[3] `https://github.com/mlfoundations/task_vectors`

Adapting to new environment and defense against typography attacks In Tab. 11, we present hyperparameters for training models in §4.3 and §4.5. Note that we post-train the MLPs and layer normalization in final encoder layer.

Adversarial trigger optimization We present hyperparameters for optimizing adversarial triggers (small patches and invisible noise) in Tab.12 and Tab.13. Transparency-α in Tab.13 denotes the process of blurring the noise in x_{trigger} to make the triggered image stealthy. Specifically, we attack the model with triggered image, $x_{\text{attack}} = (1-\alpha) \cdot x_{\text{original}} + \alpha \cdot x_{\text{trigger}}$. Additionally, for training BadNet [20], we post-train the fine-tuned classifier to misclassify trigger-attached images using cross-entropy loss, with a learning rate of 1e-4, while keeping other settings unchanged.

Table 8: Hyperparameters for class unlearning with ViT-B/32.

Hyperparameters	**MNIST**	**EuroSAT**	**GTSRB**	**RESISC45**	**DTD**
Epochs			15		
Sample size (%)	1	5	1	5	10
Learning rate	4e-2	5e-2	4e-2	3e-2	5e-2
Scaling coefficient			-1.5		
Reference sample (image)			1		

Table 9: Hyperparameters for class unlearning with ViT-B/16.

Hyperparameters	**MNIST**	**EuroSAT**	**GTSRB**	**RESISC45**	**DTD**
Epochs			15		
Sample size (%)	1	5	1	5	10
Learning rate	4e-2	5e-2	6e-2	3e-2	6e-2
Scaling coefficient			-1.5		
Reference sample (image)			1		

Table 10: Hyperparameters for class unlearning with ViT-L/14.

Hyperparameters	**MNIST**	**EuroSAT**	**GTSRB**	**RESISC45**	**DTD**
Epochs			15		
Sample size (%)	1	5	1	5	10
Learning rate	4e-2	3e-2	6e-2	3e-2	6e-2
Scaling coefficient			-1.5		
Reference sample (image)			1		

Table 11: Hyperparameters for adapting to new environment and defense against typography attacks.

Hyperparameters	**Snowy env. (1)**			**Snowy env. (2)**			**Typography attack**		
	ViT-B/16	**ViT-B/32**	**ViT-L/14**	**ViT-B/16**	**ViT-B/32**	**ViT-L/14**	**ViT-B/16**	**ViT-B/32**	**ViT-L/14**
Epochs					15				
Sample size (images)		6			6			4	
Learning rate					1e-4				
Scaling coefficient		2.5			-1.0			-1.5	
Reference sample (image)					1				

Table 12: Hyperparameters for optimizing small patches across classifiers.

Hyperparameters	**SVHN**			**GTSRB**			**RESISC45**		
	ViT-B/16	**ViT-B/32**	**ViT-L/14**	**ViT-B/16**	**ViT-B/32**	**ViT-L/14**	**ViT-B/16**	**ViT-B/32**	**ViT-L/14**
Epochs					100				
Sample size (%)					10				
Learning rate		5			5			10	
Scaling coefficient					1.5				
Reference sample					1				
Patch Size					20×20				

Table 13: Hyperparameters for optimizing invisible noise across classifiers.

Hyperparameters	**SVHN**			**GTSRB**			**RESISC45**		
	ViT-B/16	**ViT-B/32**	**ViT-L/14**	**ViT-B/16**	**ViT-B/32**	**ViT-L/14**	**ViT-B/16**	**ViT-B/32**	**ViT-L/14**
Epochs	15								
Sample size (%)	10								
Learning rate	300			200			600		
Scaling coefficient	1.0								
Reference sample	1								
Patch Size	224×224								
Transparency-α	2e-4								

E Additional Experiments

In this section, we present additional experiments on ViT-B/32 and ViT-L/14 for class unlearning, as well as results for backdoor attacks using adversarial triggers on the ViT-B/16 and ViT-L/14. As shown in Tab.14 and Tab.15, Class Vectors consistently outperform baselines. Meanwhile, random vectors highlight that editing with Class Vectors provides an intuitive way to remove model adaptations. Additionally, gradient ascent (*i.e.*, NegGrad) still struggles to maintain accuracy, while retraining remains ineffective due to data insufficiency. Tab. 16 also demonstrates Class Vector's stable performance in backdoor attack scenario, effectively surpassing other baselines across two different types of adversarial triggers.

Table 14: Comparison of class unlearning with baselines on ViT-B/32.

Method	**MNIST**		**EuroSAT**		**GTSRB**		**RESISC45**		**DTD**	
	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)
Pretrained	54.6±33	47.3±3.4	57.8±21.4	44.5±2.4	43.3±30.1	32.2±1.6	71±24.4	60.1±0.6	35±33.2	44.6±0.7
Fine-tuned	99.8±0.1	99.7±0.0	99.9±0.1	99.8±0.0	99.5±0.6	98.7±0.0	98.9±0.1	96.1±0.0	70.5±16.7	79.6±0.4
Retrained	0.0±0.0	63.7±2.2	0.0±0.0	79.6±1.6	1.2±2.4	48.6±0.7	27.7±22.5	71.8±0.5	17.0±31.6	54.6±0.6
NegGrad	0.0±0.0	33.7±8.7	0.0±0.0	16.6±6.1	0.0±0.0	31.7±27.6	0.0±0.0	6.1±7.6	0.0±0.0	9.0±11.7
Random Vector	99.8±0.1	99.6±0.0	99.9±0.1	99.4±0.2	99.5±0.6	97.2±1.5	97.9±3.5	43.3±23.1	63.5±30.6	42.4±20.5
Class Vector†	0.0±0.0	99.6±0.1	0.0±0.0	92.0±8.8	0.0±0.0	94.6±5.9	7.1±9.0	65.4±17.9	15.0±19.4	66.3±13.2

Table 15: Comparison of class unlearning with baselines on ViT-L/14.

Method	**MNIST**		**EuroSAT**		**GTSRB**		**RESISC45**		**DTD**	
	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)	ACC_f (↓)	ACC_r (↑)
Pretrained	86.7±8.1	75.2±0.9	58.1±28.7	63.0±4.6	85.4±14.2	49.0±1.1	68.5±18.8	71.4±0.5	35.0±33.5	55.8±0.7
Fine-tuned	99.8±0.0	99.7±0.0	99.9±0.2	99.7±0.0	99.6±0.7	99.2±0.0	99.6±0.7	97.3±0.0	76.5±15.7	84.3±0.3
Retrained	42.8±37.0	88.8±1.0	8.7±14.5	90.1±1.3	59.6±32.0	64.2±1.6	36.1±26.7	80.2±0.4	16.5±30.5	62.7±0.9
NegGrad	0.0±0.0	53.7±18.3	0.0±0.0	11.2±1.0	0.0±0.0	21.2±19.9	0.0±0.0	10.5±12.3	0.0±0.0	9.7±10.2
Random Vector	99.8±0.1	99.7±0.0	100.0±0.0	94.2±8.9	99.5±0.7	99.1±0.1	99.5±0.7	94.8±3.0	59.0±46.2	53.7±23.8
Class Vector†	3.7±7.4	97.9±3.6	0.0±0.0	92.7±8.7	9.7±19.3	94.8±5.0	0.0±0.0	91.2±3.5	23.5±34.9	70.5±12.9

Table 16: Results on backdoor attacks with optimized triggers for ViT models.

Method	**ViT-B/16**				**ViT-L/14**			
	Small Patch		**Invisible**		**Small Patch**		**Invisible**	
	ASR (↑)	CA (↑)	ASR (↑)	CA (↑)	ASR (↑)	CA (↑)	ASR (↑)	CA (↑)
Pretrained	32.7±29.0	53.9±9.5	28.5±22.7	53.9±9.5	14.9±20.4	60.1±8.6	22.2±27.9	60.1±8.6
Finetuned	0.3±0.2	98.0±0.9	0.3±0.2	98.0±0.9	0.1±0.1	98.2±0.8	0.1±0.1	98.2±0.8
BadNet	90.9±12.8	29.6±23.9	91.3±12.3	27.4±20.9	70.2±41.1	10.0±5.0	100±0.0	9.3±7.4
Random Vector	0.1±0.1	98.0±0.9	0.2±0.3	98.0±0.9	0.0±0.0	98.2±0.8	1.2±1.7	98.2±0.8
DirMatch	99.8±27.9	98.0±0.9	97.1±2.3	98.0±0.9	69.3±43.2	98.2±0.8	95.3±5.0	98.2±0.8
Class Vector†	100±0.0	98.0±0.9	97.5±2.5	98.0±0.9	100±0.0	98.2±0.8	97.5±1.8	98.2±0.8

Table 17: Task Vector vs. Class Vector in class unlearning. Each cell reports ($ACC_f \downarrow$, ACC_r).

Model	MNIST		EuroSAT		GTSRB	
	Task Vector	Class Vector	Task Vector	Class Vector	Task Vector	Class Vector
ViT-B/32	20.0, 8.8	**0.0, 99.6**	20.0, 10.1	**0.0, 92.0**	20.0, 0.3	**0.0, 94.6**
ViT-B/16	20.0, 8.8	**0.0, 99.7**	20.0, 9.7	**0.0, 99.5**	20.0, 0.5	**0.0, 98.6**
ViT-L/14	0.0, 1.1	**3.7, 97.9**	0.0, 10.7	**0.0, 92.7**	0.0, 1.0	**9.7, 94.8**

F Mapping Sensitivities

Table 18: Impact of learning rate (LR) on Class Vector editing.

LR	MNIST		EuroSAT		GTSRB	
	ACC_f	ACC_r	ACC_f	ACC_r	ACC_f	ACC_r
`2e-5`	0.0	99.7	0.0	99.7	0.0	98.3
`5e-5`	0.0	99.7	0.0	99.8	0.0	98.6
`1e-4`	0.0	99.6	0.0	99.8	0.0	98.0
`2e-4`	0.0	99.4	0.0	99.7	0.0	96.2
`5e-4`	0.0	98.2	0.0	97.8	0.0	60.3

Table 19: Performance of Class Vector mapping with KLD loss across datasets.

	MNIST		EuroSAT		GTSRB		RESISC45		DTD	
	$ACC_f \downarrow$	$ACC_r \uparrow$	$ACC_f \downarrow$	$ACC_r \uparrow$	$ACC_f \downarrow$	$ACC_r \uparrow$	$ACC_f \downarrow$	$ACC_r \uparrow$	$ACC_f \downarrow$	$ACC_r \uparrow$
KLD	0.0	99.7	0.0	99.8	0.0	98.7	0.0	94.5	15.0	79.3
MSE	0.0	96.2	0.0	99.7	0.0	93.4	10.0	90.7	15.2	72.9

Sensitivity to learning rate. Tab. 18 examines the effect of learning rate on class-vector editing across MNIST, EuroSAT, and GTSRB. We report *Forget Accuracy* (↓) and *Retain Accuracy* (↑) for each setting. At low to moderate rates (`2e-5`–`2e-4`), forgetting remains at 0% while retention stays above 94%, peaking at nearly 99.8%. However, at `5e-4`, retention on GTSRB plummets to 60.3%, indicating that an excessively large learning rate destabilizes the edit.

Impact of loss function. Theorem 3.2 demonstrates that mapping solutions can be obtained from infinitely many distinct configurations. Tab. 19 compares KLD vs. MSE (default) as the mapping loss across MNIST, EuroSAT, GTSRB, RESISC45, and DTD. Even with KLD loss, forgetting remains perfect (0%) and retention exceeds 94.5% on all but the most challenging texture dataset (DTD, 79.3%).

G Computational Analysis

Table 20: Analysis on computational complexity of classifier editing with Class Vectors.

Application	Task	# of parameters	Time (s)
Unlearning	MNIST	1.5K	2.5
Adapting to new environment	-	4.7M	1.2
Defending against typography attacks	-	4.7M	10.4
Small patch trigger optimization	RESISC45	0.4K	238
Invisible noise trigger optimization	RESISC45	1.5M	38.1

We evaluate the computational complexity of classifier editing using Class Vectors mapping by measuring the number of trainable parameters and the time required for model editing with ViT-B/32. As shown in Tab. 20, Class Vectors enable efficient classifier editing across diverse applications, requiring a minimum of 1.5K trainable parameters or just 1.2 seconds. This aligns with the philosophy of model intervention, which aims to adjust models quickly using a small number of samples while achieving effective editing performance.

H Supplementary Figures

Figure 10: (Top) Snowy objects for adaptation to a snowy environment. (Middle) and (Bottom) Web-crawled clean object images for each class to isolate snow representation in z_{edit} to eliminate snow representation in the second scenario.

Figure 11: Augmented images across classes for typography attacks.

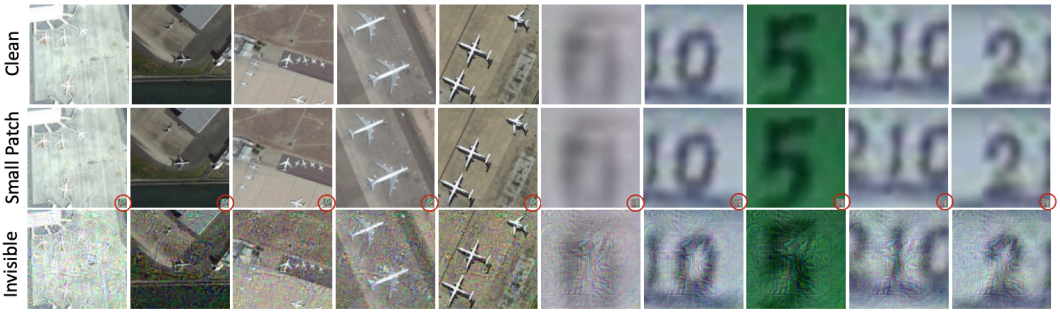


Figure 12: Trigger-attached images in RESISC45 and SVHN.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have included paper's contributions and scope in abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have included the limitations in the conclusion part and in the Appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: For each theorem, we provide its proof and an empirical validation of its assumptions in the Appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Detailed information for reproducing the experimental results is provided in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We are going to release our code either during the review period or shortly thereafter.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide experimental details in Appendix §D.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Throughout our experiments, we consistently report error bars representing standard deviations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the resources information in Appendix §D.3.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have adhered to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss this in Appendix A.3.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: We discuss this in Appendix A.3.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We follow the license and usage guidelines provided by creators of the assets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the license, limitations and training details in Appendix.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development in this research does not involve LLMs.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.