Learning to Forget using Hypernetworks

Jose Miguel Lara Rangel[1] Stefan Schoepf[1] Jack Foster[1]
David Krueger[2] Usman Anwar[1]

Abstract

Machine unlearning is gaining increasing attention as a way to remove adversarial data poisoning attacks from already trained models and to comply with privacy and AI regulations. The objective is to unlearn the effect of undesired data from a trained model while maintaining performance on the remaining data. This paper introduces HyperForget, a novel machine unlearning framework that leverages hypernetworks – neural networks that generate parameters for other networks – to dynamically sample models that lack knowledge of targeted data while preserving essential capabilities. Leveraging diffusion models, we implement two Diffusion HyperForget Networks and used them to sample unlearned models in Proof-of-Concept experiments. The unlearned models obtained zero accuracy on the forget set, while preserving good accuracy on the retain sets, highlighting the potential of HyperForget for dynamic targeted data removal and a promising direction for developing adaptive machine unlearning algorithms.

1 Introduction

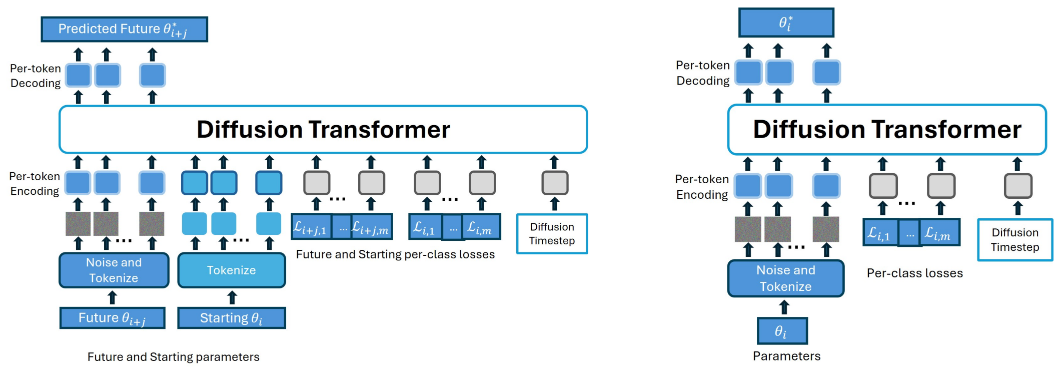
The need for machine learning (ML) models to forget specific points of their training data has become an essential requirement due to increasing security, ethical, and regulatory concerns in AI. Data forgetting is a critical defense mechanism against adversarial attacks that manipulate models to change their behavior or extract training data information from them [16, 41, 5, 39, 31, 4, 34]. Additionally, regulations like the GDPR with its Right-to-be-Forgotten (RTBF) or the EU AI Act enhance individuals' data privacy rights, allowing them to request the deletion of their data [51, 44, 11, 57]. As ML models capture information of their training data in their parameters, aligning them with ethical and regulatory standards requires not only to delete stored data but also to remove its influence on the parameters, which directly impacts the model's capabilities [2, 13].

Machine unlearning (MU) focuses on developing algorithms capable of efficiently and effectively removing the influence of specific data on an ML model, while maintaining unrelated knowledge or capabilities unaffected [1, 28, 2]. Ideally, an unlearned model should behave identically to a model that was never trained on the data being unlearned in the first place. Thus, for randomly initialized models, exact unlearning is achieved when the distribution of unlearned models is identical to the distribution of models trained on the dataset D excluding the forget set $D_f \subseteq D$, either by equating their distribution of parameters or outputs [3, 14, 35, 24, 49]. The gold standard for exact unlearning is retraining the model from scratch without the forget set, which is costly in time and resources as it requires full access to the training dataset and must be done for each forgetting request.

Consequently, research has focused on the development of approximate unlearning methods to mitigate the retraining drawbacks [18, 8, 28, 35, 43]. An ideal unlearning algorithm should be consistent with the retrained model outputs, preserve as much performance as possible on the retain set $D_r = D \backslash D_f$, be faster than retraining, provide guarantees of effective removal of D_f influence, be lightweight, scalable, and avoid recovering unlearned data in strict compliance scenarios. Achieving

[1]University of Cambridge. [2]MILA & Université de Montréal.

AdvML-Frontiers'24: The 3rd Workshop on New Frontiers in Adversarial Machine Learning@NeurIPS'24, Vancouver, CA.



(a) DiHyFo-1 (b) DiHyFo-2

Figure 1: Two implementations of Diffusion Hyperforget Networks. DiHyFo-1 is conditioned on current parameters, current losses and target losses to work as a learned optimizer-deoptimizer, while DiHyFo-2 is conditioned only on target losses to work as a conditioned generator.

all these conditions is challenging and often involves trade-offs based on application needs, forget set size, and available resources [35, 58, 14, 48, 19, 56, 32, 17, 21, 45].

Pre-trained methods are proactive and embed unlearning capabilities into the model's design or training process, which makes them efficient, robust and reliable at removing undesired data points, but require complex designs and higher computational overhead during training [2, 54, 57]. Post-training methods, the most popular type of unlearning methods, are reactive and can be applied on existing models without redesign, but may not always fully remove data influence [8, 12, 16].

Also, unlearning algorithms can be model intrinsic, agnostic, or involve data modifications for faster retraining; and may need full, partial, or no access to training data [2, 8, 48]. Particularly, we refer as Retrieval-Enhanced Unlearning (REU) to methods that use stored metadata or auxiliary data saved during the model's training process, distinct from the actual training data, to induce unlearning–inspired by human Retrieval-Induced Forgetting phenomena [9, 33]. A relevant concern is scalability due to potential challenges in metadata storage and retrieval [8, 28, 35, 50].

We propose a new REU pre-trained framework that treats forgetting as a generative process with a sequence of states with varying levels of forgetting. This process transitions from less to more forgetting, positioning forgetting as the inverse of learning—an unlearning process. By reconstructing these states, we aim to place the model in a state where it has gained essential knowledge on the retain set, but not on the forget set. For this, we employ hypernetworks to generate parameters with reduced performance on the forget set, while preserving performance on the retain set. We name this approach HyperForget and, to the best of our knowledge, it is the first use of hypernetworks for MU.

By integrating diffusion models as hypernetworks [46, 22, 47], we create two Diffusion HyperForget Networks (DiHyFo) shown in Figure 1, and use them to sample unlearned models for MU tasks in Proof-of-Concept (POCs) scenarios, comparing them with models retrained from scratch without the forget sets to show their potential for MU. Notably, while a retrained model should be constructed for each forget set, one DiHyFo can sample unlearned models for all the forget sets and can be interpreted and evaluated using both the parameter and the output space perspectives of unlearning. Our results suggest that the sampled unlearned models effectively achieve zero performance on forget sets while maintaining high accuracy on retain sets, and closely mimic a retrained model. Our contributions are:

1. We conceptualize and approach forgetting for ML models as a generative process, making a model to forget by positioning it in a state where its parameters have gained relevant knowledge and capabilities on the retain set but not on the forget set.
2. We propose a novel REU framework leveraging hypernetworks and diffusion models. We present two implementations and POCs to show their potential for unlearning, and highlight significant limitations and potential future directions.
3. One DiHyFo can sample unlearned models for different forget set configurations, enabling the development of more adaptable unlearning methods for dynamic forgetting requests.

2 Related work

Hypernetworks. A neural network $G(C; \theta_G)$ with learnable parameters θ_G and context input C is a hypernetwork if its output are parameters for another network $F(X; \theta_F)$ that solves a task T with associated data $D = \{X, Y\}$, i.e. $\theta_F = G(C; \theta_G)$. Therefore, in this framework θ_G are optimized to use G to sample parameters θ_F^* that can be used by F to process X and predict Y^* for task T [6, 20].

This approach can reduce the parameter search space and offers adaptability as, unlike conventional parameters that remain fixed after training, it can be conditioned to sample parameters for multiple related tasks [20, 30, 36]. It has shown promise in continual learning, transfer learning, weight pruning, and uncertainty-aware modeling [53, 6, 52, 25, 23]. However, challenges limit broader adoption, such as scalability, lack of understanding of representational capacity, learning dynamics, generalization, and ensuring parameters are valid and not merely memorized or interpolated [6, 26, 25].

Diffusion Hypernetworks. Similar to diffusion models [46, 22], neural networks training with SGD transforms randomly initialized parameters (noise) into a structured set that forms a specific distribution. This inspired the use of diffusion as hypernetworks, what we call Diffusion Hypernetworks (DiHy). While this is still an emerging area of research, early approaches like G.pt [37] show that diffusion models have significant potential to act as hypernetworks. [1] G.pt acts as a learned optimizer. By using a diffusion transformer (DiT) and DDPM [38, 22] it takes current parameters (potentially random noise) with their associated performance and target performance, and update them to obtain the desired performance, indirectly solving optimization problems traditionally handled by SGD. G.pt has demonstrated broader generative properties than other approaches, although it is less precise in achieving target metrics, requires big data sets constructed by saving checkpoints from multiple main network training runs, involves considerable training overhead, and shows limited capabilities to extrapolate to performances beyond its training data [54, 37]. However, despite these limitations, we show that its ability to learn a conditional generative model over weights of the main network is useful towards MU. We replicate some experiments related to G.pt (originally reported in [37]) and list some observations that might be of independent interest. See Appendix A.6.

Furthermore, [10] also used DiT and DDPM to generate unconditioned parameters, [54] improved computational efficiency by incorporating an autoencoder and U-Net for learning on a parameters latent space. Both works used datasets with only optimized parameters, effectively introducing a soft bias for high-performing parameters generation, but limiting capabilities for more diverse parameter distributions. Recently, [27] extended [37] ideas to text-to-model generation, where a customized classifier for a subset of classes of a large dataset can be sampled by prompting text.

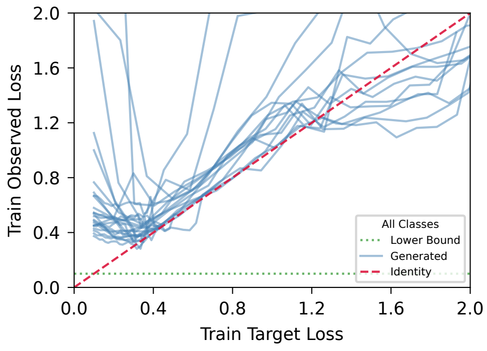
3 Proposed method

We propose **HyperForget** – a hypernetwork based framework for unlearning in which a hypernetwork is trained to learn a distribution over model parameters. The hypernetwork is then used to sample model parameters that solve a forgetting task U. Thus, the resulting model, a HyperForget Network (HyFo), can generate parameters that simultaneously yield high-performance on the retain set D_r and low-performance on the forget set D_f associated to task U. Indeed, it reconstructs the knowledge and capabilities for D_r while effectively forgetting D_f, achieving this efficiently in a single forward pass, eliminating gradient calculations typically required by other unlearning methods, and allowing for dynamic unlearning adaptation in scenarios with frequent and varying forget requests.

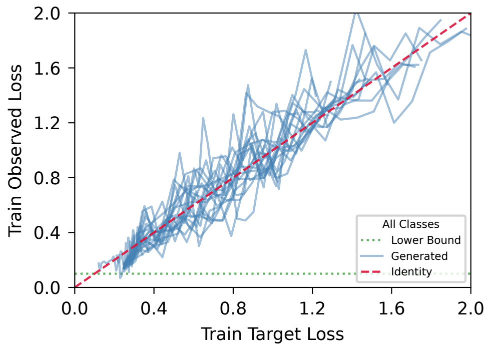
In this work, we use DiT as the hypernetwork. We call the resulting model Diffusion HyperForget Network (DiHyFo). Integrating diffusion models into the HyperForget framework provides a structured approach to gradual unlearning and allows to control the level of data influence removal. We explored two options to construct a DiHyFo for class-level unlearning.

Consider a classification task T with a dataset D formed by examples $x_i \in X$ with associated labels $y_i \in \{1, ..., m\}, m \in \mathbb{N}$, drawn from an unknown distribution P. A model F trained to solve T is requested to forget a subset of classes, i.e., D_f contains data from specific class labels. For this task, we employ a DiHyFo conditioned on class losses to generate parameters that obtain high performance in D_r and low performance in D_f.

[1]Although G.pt was not categorized as hypernetwork in [37], mainly due to it predicting targeted parameters updates instead of direct parameters, we categorize it as a hypernetwork according to our previous definition.



(a) Dihyfo-1 obtained losses for class 2 of MNIST



(b) Dihyfo-2 obtained losses for class 2 of MNIST-4

Figure 2: Examples of observed vs target losses for class 2 using models sampled with DiHyFo.

DiHyFo-1 uses a DiT and follows [37] but conditioned at a class-level. This allows us to control losses for specific classes at the same time, e.g., prompt DiT to generate parameters with high loss values on classes in the forget set, but low loss values on classes in the retain set. This is a different and more challenging optimization problem than the one solved in [37].

During training, current parameters θ, future parameters θ^*, and their associated class losses $\mathcal{L}_1, ..., \mathcal{L}_m$ and $\mathcal{L}_1^*, ..., \mathcal{L}_m^*$ for task T are randomly selected from a run uniformly sampled from a dataset of checkpoints grouped by runs. θ is always from an earlier step than θ^*. Parameters are normalized and tokenized layer-by-layer flattening them into 1D vectors, each corresponding to a unique token. Layers with both weight and bias are decomposed into separate tokens, and each token accepts a maximum number of parameters set to be smaller than the DiT hidden size.

The time-step, losses, and their differences $\Delta_{\mathcal{L}_1,\mathcal{L}_1^*}, ..., \Delta_{\mathcal{L}_m,\mathcal{L}_m^*}$ are tokenized with a frequency-based encoder. All the tokens are linearly projected to the DiT hidden size, and Standard Gaussian noise is added to the future parameters at each time-step, and then passed to the DiT to learn to denoise them. The DiT has a decoder at the end that linearly projects each token back to its original size, and a residual connection to predict the parameter updates $\theta_t^* - \theta_t$. The training objective is to minimize the MSE between the generated parameters and the original future parameters. This enables to sample parameters directly in the parameter space with the desired losses for each class.

At inference, the model takes a set of current parameters (potentially random noise) with current and target losses for each class, and returns the updated parameters for the requested losses by denoising an input Gaussian noise vector via DDPM with fixed variances [22]. Thus, DiHyFo-1's goal is to predict the distribution of updated parameters that achieve each target class loss.

DiHyFo-2 is directly conditioned on the desired class losses, Figure 1b, which has shown good results in other tasks using DiHy [27, 54]. The DiT is conditioned on a large set of examples of parameters with both high and low performance on individual classes, enabling to synthesize new parameters tailored to specific losses directly in the parameter space. Thus, the task of DiHyFo-2 is to predict the distribution of parameters that achieve the desired losses.

Both models use a DiT with hidden dimension of 1536, 12 hidden layers, and 16 attention heads, trained with AdamW [29] maintaining an exponential moving average of the model weights across the training process. Learned positional embeddings, initialized to zero, are applied across all tokens. Additional details of the framework and both models can be found in Appendix A.3

4 Experimental Setup

Checkpoints datasets. To reduce the complexity of the task, we consider a simplified scenario with m classes, out of which r classes are considered as 'pivot' classes. For pivot classes, we only consider high-performing parameters. For the remaining $m - r$ class losses, DiHyFo models are trained on parameters with varied values of losses. In general, any data that a model designer is certain would not need to be forgotten later can act as a pivot, simplifying the learning problem. The forget set can be any subset of the $m - r$ classes, while the retain sets must always include the pivots. More complex scenarios can consider a small number of pivots or no pivots.

Checkpoints of parameters and their class losses are collected during multiple training runs of an MLP classifier trained on MNIST. A random subset of classes is selected for undersampling in each run to capture more diverse losses, and during training checkpoints are randomly selected and evaluated for saving. Similar to prior work [37, 40, 42], we apply permutation augmentation to help DiHyFo learn that weights of neural networks are permutation invariant. To provide DiHyFo-1 with examples of losses increments we delete a random subset of classes during MLP training in some runs and save the corresponding checkpoints. We further use heuristics to track the evolution of class losses and prioritize checkpoints with varied losses across classes over checkpoints with similar losses across different classes. For a detailed description of the checkpoint datasets collection see Appendix A.5.

For our experiments, we consider two variations of MNIST dataset – full MNIST and MNIST-4 with four classes. For MNIST-4, we use classes $\{0,1\}$ as pivots. We save 150 checkpoints during training runs of 25 epochs of a two-layer MLP with two hidden units and ReLU activation, totaling 1.9M checkpoints. For full MNIST, we use $\{5\text{-}9\}$ as pivot classes, seven hidden units MLP, and 200 checkpoints saved per training run for a total of 3.7M checkpoints.

Evaluation procedures. The ability of DiHyFo to generate class-loss targeted parameters is evaluated by analyzing losses on train and test set. Following [37], we use the metric of prompt alignment (see Appendix A.4), which intuitively measures the alignment between the observed losses and target losses. A prompt alignment score of 1 indicates perfect alignment. Further, we include the correlation between both losses as a complementary metric, and plot them along the identity line with a reference of high performance—often the median or average of losses in checkpoints.

As there are no general frameworks or benchmarks in MU, a typical strategy to assess the performance of an MU method is to compare the unlearned model with a model retrained from scratch only on the retain set. The closer the behavior of the unlearned model to the retrained model, the more effective the unlearning. Thus, to assess unlearning capabilities of DiHyFo, we sample unlearned models and compare against a retrained model. We consider $\{2\}$ and $\{2,3\}$ as forget sets for MNIST-4, while $\{2\}$, $\{2,3,4\}$, and $\{0,1,2,3,4\}$ for MNIST. Notably, each forget set required a retraining process, whereas a single DiHyFo model can sample unlearned models for all forget sets, highlighting its suitability for scenarios with dynamic unlearning requests. We compare accuracy on test sets and member inference attacks score (MIA) to assess the unlearning consistency and guarantees against adversarial attacks to infer forget sets information [7]. The output and parameter spaces of the sampled unlearned models are compared against the retrained model by computing the predictions overlap, and the *unlearning score*. Unlearning score is the Jensen-Shannon Divergence (JSD) between the probability distributions outputted by the sampled unlearned model and a candidate retrained model [12, 35, 7]. As we only consider class-forgetting setting, ideally, the predictions overlap on the retain set should be high between the unlearned model and retrained model, while for the forget set a low overlap would be more desirable. Similarly, a high unlearning score value is preferred as this indicates that the unlearned model's predictions are similar to those of the retrained model. The maximum value of the unlearning score can be 1 and the lowest value can be 0.

5 Results and Discussion

Generative performance. We measure the capabilities of each model to generate parameters with the desired loss by computing the prompt alignment [37] and correlation between the target loss and the actual loss obtained with the sampled parameters. Both these metrics show initial fluctuations but tend to increase and stabilize trough training. As depicted in Figure 2, although the observed losses generally align, it is challenging for the model to precisely match the target losses, particularly for higher loss values (see Appendix A.2 for additional results). Similarly, the observed losses for pivots track the target losses, successfully retaining performance but struggling with high losses and jumps between levels, as expected with the dataset structure. As we focus on low-loss regions for pivots, these deviations are less relevant for our evaluations.

Learning bidirectional loss movements at a class level is a challenging task, leading to less precise prompt alignment than in other tasks [37]. However, this is not a major issue as unlearning typically focuses on high and low-performing parameters rather than covering the entire loss range. While

Table 1: Individual unlearned models performance on MNIST under diverse forgetting scenarios. Average over 5 sampled unlearned models.

MNIST	Sampled with DiHyfo-1			Sampled with DiHyfo-2			Retrained Model		
	D_f={2}	D_f={2,3,4}	D_f={0,1,2,3,4}	D_f={2}	D_f={2,3,4}	D_f={0,1,2,3,4}	D_f={2}	D_f={2,3,4}	D_f={0,1,2,3,4}
Accuracy on D_r (↑)	0.9072	0.9390	0.9373	0.7369	0.9433	0.9518	0.9193	0.9479	0.9504
Accuracy on D_f (↓)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
MIA (↓)	0.3682	0.3817	0.4282	0.3580	0.3243	0.3299	0.3299	0.3406	0.3798

Table 2: Assessment of output alignment and behavioral consistency between sampled unlearned models and their corresponding retrained model on MNIST under diverse forgetting scenarios. Average over 5 sampled unlearned models.

MNIST	Sampled with DiHyfo-1			Sampled with DiHyfo-2		
	D_f={2}	D_f={2,3,4}	D_f={0,1,2,3,4}	D_f={2}	D_f={2,3,4}	D_f={0,1,2,3,4}
Outputs Overlap on D_r (↑)	0.9106	0.9511	0.9498	0.7298	0.9364	0.9482
Outputs Overlap on D_f (−)	0.5474	0.7262	0.7498	0.4221	0.7110	0.6375
Outputs Overlap on Test Set (↑)	0.8731	0.8831	0.8471	0.6978	0.8682	0.7881
Unlearning Score (φ) (↑)	0.9885	0.9894	0.9916	0.8478	0.9857	0.9540

for MNIST-4 DiHyFo-2 showed to be better at aligning losses, for MNIST DiHyFo-1 showed better consistency and robustness. See Appendix A.2 for further details.

Unlearning Performance. Table 1 shows that all the sampled unlearned models achieved zero accuracy on the forget sets and maintain a good accuracy on the retain sets, comparable to the retrained models. The obtained MIA scores for all sampled unlearned models are very close to the retrained models, indicating good unlearning and robustness against inference attacks. As shown in Table 2, the comparison of the outputs produced by the sampled unlearned models and the retrained models shows similar predictions on the retain and test sets, aligning with individual performance. Note that the perfect overlap is not expected, as our unlearned model only needs to behave like any retrained model that results from the stochastic nature of the training process, not like one specific retrained model. Moreover, the comparisons of the parameters space show high unlearning scores. Thus, the sampled unlearned models are mimicking the behavior of a retrained model, with DiHyFo-1 performing more consistently across unlearning tasks, particularly with MNIST. See Appendix A.1 for additional results and further discussion.

Overall, the sampled models are effectively unlearning the forget set and no longer rely on the associated data to make predictions, and preserve performance on retain sets without significant signs of catastrophic unlearning [55, 35], closely approximating the unlearning gold standard. Therefore, we have a potential REU method that is model-intrinsic, accuracy-preserving with a good level of completeness, unlearning guarantees, and a potential choice for dynamic unlearning.

6 Limitations and Future Work

In this work, we have provided proof-of-concept experiments for a hypernetworks based approach to unlearning. While our results indicate the potential of this approach, many opportunities for future work exist. The current formulation of HyperForget focuses on full-class unlearning, its extension and potential applicability for other unlearning tasks have yet to be explored. A key limitation of this approach is that the generative model retains the knowledge of forget sets making this approach unsuitable for strict unlearning-for-privacy applications. Furthermore, it is not clear from our experiments whether the proposed approach is scalable or not. The use of diffusion models adds significant computational overhead, and it inherits concerns from DiHy models, such as limited generalization [37, 54]. Additionally, although generated parameters generally approximate target losses, it is difficult for the model to precisely match the target loss.

Future works could consider evaluating our framework on other datasets, e.g., CIFAR-10 or mini-ImageNet, as well as improving the scalability of our approach. Future research could explore alternative architectures, improve checkpoint saving strategies, optimize the process and components, or explore other unlearning tasks. Improvements may come from leveraging model zoos, hypernetworks for architecture-agnostic parameter generation, and learning on latent spaces. Both hypernetworks and MU are in an early stage, and we hope this work inspires future research to expand these areas, contributing to more adaptive, secure, and ethical AI solutions.

References

[1] Thomas Baumhauer, Pascal Schöttle, and Matthias Zeppelzauer. Machine unlearning: Linear filtration for logit-based classifiers. *Machine Learning*, 111(9):3203–3226, 2022.

[2] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *Proceedings - 2021 IEEE Symposium on Security and Privacy, SP 2021*, Proceedings - IEEE Symposium on Security and Privacy, pages 141–159. Institute of Electrical and Electronics Engineers Inc., 2021.

[3] Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *International Conference on Machine Learning*, pages 1092–1104. PMLR, 2021.

[4] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pages 463–480. IEEE, 2015.

[5] Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. Poisoning web-scale training datasets is practical. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 175–175. IEEE Computer Society, 2024.

[6] Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. A brief review of hypernetworks in deep learning. *arXiv preprint arXiv:2306.06955*, 2023.

[7] Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Can bad teaching induce forgetting? unlearning in deep networks using an incompetent teacher. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7210–7217, 2023.

[8] Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Zero-shot machine unlearning. *IEEE Transactions on Information Forensics and Security*, 18:2345–2354, 2023.

[9] Ronald L. Davis and Yi Zhong. The biology of forgetting—a perspective. *Neuron*, 95(3):490–503, 2017.

[10] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14300–14310, 2023.

[11] European Commission. Regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence, amending regulations and directives (artificial intelligence act), 2024. Accessed: 2024-08-11.

[12] Jack Foster, Kyle Fogarty, Stefan Schoepf, Cengiz Öztireli, and Alexandra Brintrup. An information theoretic approach to machine unlearning, 2024.

[13] Jack Foster, Stefan Schoepf, and Alexandra Brintrup. Fast machine unlearning without retraining through selective synaptic dampening. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12043–12051, 2024.

[14] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. Making ai forget you: Data deletion in machine learning. *Advances in neural information processing systems*, 32, 2019.

[15] Shashwat Goel, Ameya Prabhu, and Ponnurangam Kumaraguru. Evaluating inexact unlearning requires revisiting forgetting. *CoRR abs/2201.06640*, 2022.

[16] Shashwat Goel, Ameya Prabhu, Philip Torr, Ponnurangam Kumaraguru, and Amartya Sanyal. Corrective machine unlearning, 2024.

[17] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.

[18] Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11516–11524, 2021.

[19] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.

[20] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[21] Yingzhe He, Guozhu Meng, Kai Chen, Jinwen He, and Xingbo Hu. Deepobliviate: a powerful charm for erasing data residual memory in deep neural networks. *arXiv preprint arXiv:2105.06209*, 2021.

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[23] Agustinus Kristiadi, Sina Däubener, and Asja Fischer. Predictive uncertainty quantification with compound density networks. *arXiv preprint arXiv:1902.01080*, 2019.

[24] Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning. *Advances in neural information processing systems*, 36, 2024.

[25] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 608–624. Springer, 2020.

[26] Yawei Li, Wen Li, Martin Danelljan, Kai Zhang, Shuhang Gu, Luc Van Gool, and Radu Timofte. The heterogeneity hypothesis: Finding layer-wise differentiated network architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2144–2153, 2021.

[27] Zexi Li, Lingzhi Gao, and Chao Wu. Text-to-model: Text-conditioned neural network diffusion for train-once-for-all personalization. *arXiv preprint arXiv:2405.14132*, 2024.

[28] Sijia Liu, Yuanshun Yao, Jinghan Jia, Stephen Casper, Nathalie Baracaldo, Peter Hase, Xiaojun Xu, Yuguang Yao, Hang Li, Kush R Varshney, et al. Rethinking machine unlearning for large language models. *arXiv preprint arXiv:2402.08787*, 2024.

[29] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[30] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

[31] Neil G Marchant, Benjamin IP Rubinstein, and Scott Alfeld. Hard to forget: Poisoning attacks on certified machine unlearning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7691–7700, 2022.

[32] Paul Micaelli and Amos J Storkey. Zero-shot knowledge transfer via adversarial belief matching. *Advances in Neural Information Processing Systems*, 32, 2019.

[33] Kou Murayama, Toshiya Miyatsu, Dorothy Buchli, and Benjamin C Storm. Forgetting as a consequence of retrieval: a meta-analytic review of retrieval-induced forgetting. *Psychological bulletin*, 140(5):1383, 2014.

[34] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A. Feder Cooper, Daphne Ippolito, Christopher A. Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. Scalable extraction of training data from (production) language models. *ArXiv*, 2023.

[35] Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

[36] Geunseob Oh and Huei Peng. Cvae-h: Conditionalizing variational autoencoders via hypernetworks and trajectory forecasting for autonomous driving. *arXiv preprint arXiv:2201.09874*, 2022.

[37] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *ArXiv*, 2022.

[38] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[39] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360, 2020.

[40] Geoffrey Roeder, Luke Metz, and Durk Kingma. On linear identifiability of learned representations. In *International Conference on Machine Learning*, pages 9030–9039. PMLR, 2021.

[41] Stefan Schoepf, Jack Foster, and Alexandra Brintrup. Potion: Towards poison unlearning. *arXiv preprint arXiv:2406.09173*, 2024.

[42] Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493, 2021.

[43] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems*, 34:18075–18086, 2021.

[44] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. The seven sins of Personal-Data processing systems under GDPR. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, 2019. USENIX Association.

[45] Alyssa Shuang Sha, Bernardo Pereira Nunes, and Armin Haller. " forgetting" in machine learning and beyond: A survey. *arXiv e-prints*, pages arXiv–2405, 2024.

[46] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[47] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[48] Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[49] Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. Unrolling sgd: Understanding factors influencing machine unlearning. *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 303–319, 2021.

[50] Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. Unrolling sgd: Understanding factors influencing machine unlearning. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 303–319. IEEE, 2022.

[51] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[52] Tomer Volk, Eyal Ben-David, Ohad Amosy, Gal Chechik, and Roi Reichart. Example-based hypernetworks for out-of-distribution generalization. *arXiv preprint arXiv:2203.14276*, 2022.

[53] Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

[54] Kaili Wang, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *ArXiv*, 2024.

[55] Weiqi Wang, Zhiyi Tian, and Shui Yu. Machine unlearning: A comprehensive survey. *arXiv preprint arXiv:2405.07406*, 2024.

[56] Youngsik Yoon, Jinhwan Nam, Hyojeong Yun, Jaeho Lee, Dongwoo Kim, and Jungseul Ok. Few-shot unlearning by model inversion. *arXiv preprint arXiv:2205.15567*, 2022.

[57] Dawen Zhang, Pamela Finckenberg-Broman, Thong Hoang, Shidong Pan, Zhenchang Xing, Mark Staples, and Xiwei Xu. Right to be forgotten in the era of large language models: Implications, challenges, and solutions. *arXiv preprint arXiv:2307.03941*, 2023.

[58] Haibo Zhang, Toru Nakamura, Takamasa Isohara, and Kouichi Sakurai. A review on machine unlearning. *SN Computer Science*, 4(4):337, 2023.

A Appendix / supplemental material

This appendix provides additional information on the technical details and methods behind the proposed hypernetwork-based approach to MU and DiHyFo models. It includes explanations on data generation, experimental results, model definitions, and model evaluations.

The appendix is organized as follows:

- **Section A.1** Additional unlearning evaluation results: This section includes further results and analysis of unlearning capabilities.
- **Section A.2** Additional training results: We provide supplementary training results, discussing model performance, stability, and consistency across experiments to support the findings of the main study.
- **Section A.3** Additional hypernetworks, HyperForget, and DiHyFo details: This section provides details on hypernetworks and how they are used for HyperForget and DiHyFo models.
- **Section A.4** Additional details on evaluation metrics: This section outlines the metrics and evaluation methods used to assess the generative performance and unlearning effectiveness of the models.
- **Section A.5** Additional details on datasets generation: This section describes the methodologies for dataset generation and processing, including parameter checkpoint collection and strategies for class loss tracking across different configurations.
- **Section A.6** Some experimental observations on G.pt: This section presents observations and insights from experiments conducted with the G.pt model, examining performance metrics and prompt alignment under various test conditions.

A.1 Additional unlearning evaluation results

To apply the DiHyFo models for unlearning, we sample multiple parameters prompting different high losses values for different forget sets simultaneously with low losses for the corresponding retain sets. The generated parameters are used to load an instance of the main network, which is then evaluated on a test set. We save the sampled model that obtains the lowest average accuracy on the forget set while obtaining the highest possible average accuracy on the retain set (best unlearned model). We use accuracy for the selection and evaluation as it is a more interpretable performance measure than loss.

We sampled models that forget classes {2} and {2,3} for MNIST-4, and {2}, {2,3,4}, and {0,1,2,3,4} for MNIST. Figures 3 and 4 exemplify this sampling and selection process. Tables 3 and 4 present the evaluation metrics on MNIST-4, showing a close behavior of the sampled unlearned models to the retrained models.

Taking as reference the predictions of the retrained model, we also compare the output spaces using the confusion matrix between the sampled models and the corresponding retrained model on the retain and complete test set. Figures 5, 6, 7, and 8 present the results for DiHyFo-1 and DiHyFo-2 on MNIST-4 and MNIST. The forget set is not included in this comparison as it mainly contains matrices with zeros across all entries. This serves as a visual confirmation of commented findings using the evaluation metrics, obtaining zeroes in the forget set classes.

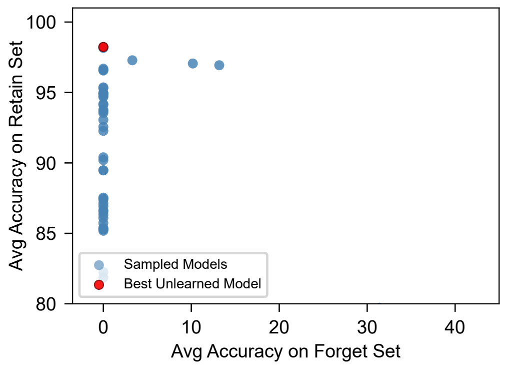
Overall, the evaluations suggest that the presented DiHyFo models accomplish some relevant desired conditions for an unlearning algorithm [35, 58, 14, 8, 48, 19, 56, 32, 17, 21, 2, 45]. They were consistent, timely at sampling unlearned models, provided guarantees and verification of unlearning, but were model intrinsic, with potential recoverability of knowledge, and had issues with scalability.

Table 3: Individual unlearned models performance on MNIST-4 under diverse forgetting scenarios. Average over 5 sampled unlearned models.

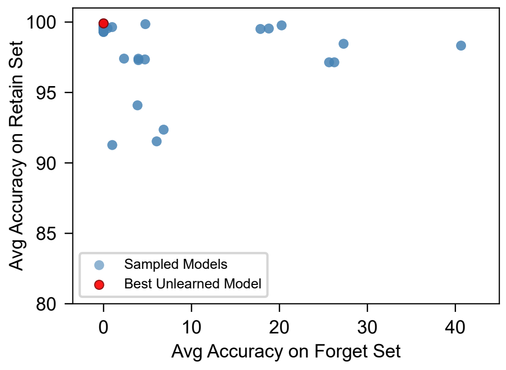
MNIST-4	Sampled with DiHyfo-1		Sampled with DiHyfo-2		Retrained Model	
	D_f={2}	D_f={2,3}	D_f={2}	D_f={2,3}	D_f={2}	D_f={2,3}
Accuracy on D_r (↑)	0.9837	0.9983	0.9834	0.9995	0.9938	0.9991
Accuracy on D_f (↓)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
MIA (↓)	0.6260	0.6484	0.4457	0.3827	0.4171	0.3398

Table 4: Assessment of output alignment and behavioral consistency between the indicated sampled unlearned model and its corresponding retrained model on MNIST-4 under diverse forgetting scenarios. Average over 5 sampled unlearned models.

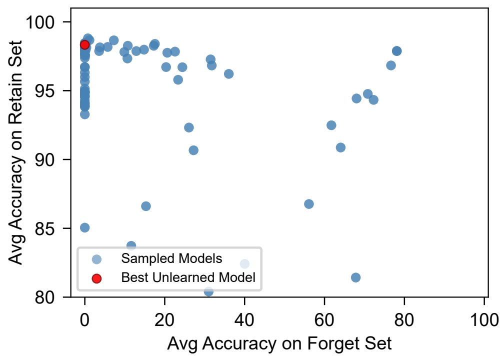
MNIST-4	Sampled with DiHyfo-1		Sampled with DiHyfo-2	
	D_f={2}	D_f={2,3}	D_f={2}	D_f={2,3}
Outputs Overlap on D_r (↑)	0.9851	0.9983	0.9848	0.9995
Outputs Overlap on D_f (−)	0.3473	0.7106	0.6800	0.7751
Outputs Overlap on Test Set (↑)	0.8263	0.8562	0.9094	0.8898
Unlearning Score (φ) (↑)	0.8550	0.9327	0.6874	0.9607



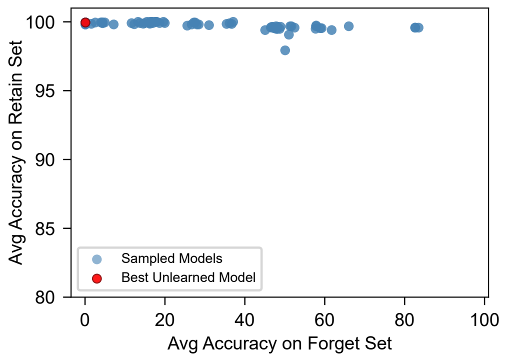
(a) DiHyFo-1 sampled models, $D_f = \{2, 3\}$



(b) DiHyFo-1 sampled models, $D_f = \{2, 3\}$

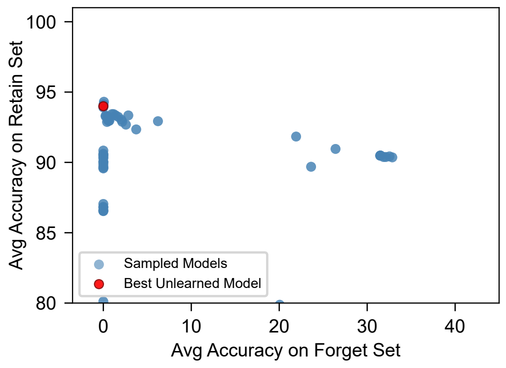


(c) DiHyFo-2 sampled models, $D_f = \{2\}$

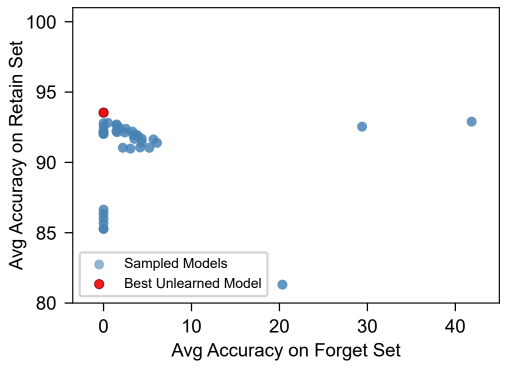


(d) DiHyFo-2 sampled models, $D_f = \{2, 3\}$

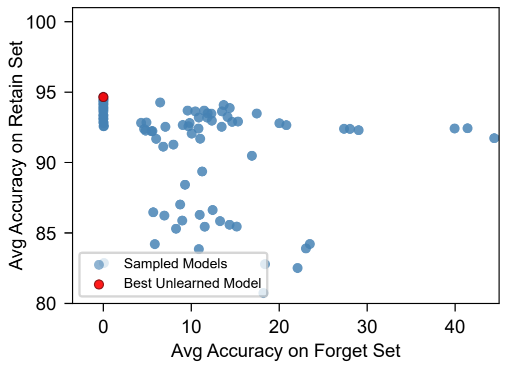
Figure 3: Selection of unlearned models sampled using DiHyFo-1 and DiHyFo-2 on MNIST-4.



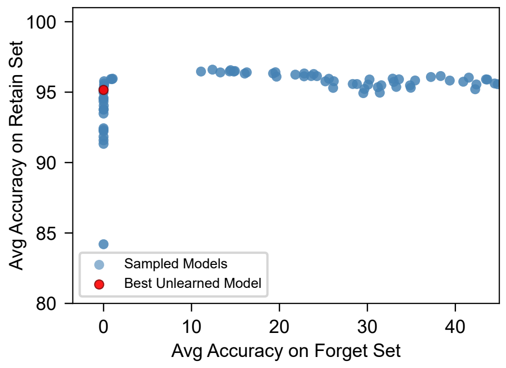
(a) DiHyFo-1 sampled models, $D_f = \{2, 3, 4\}$



(b) DiHyFo-1 sampled models, $D_f = \{0, 1, 2, 3, 4\}$

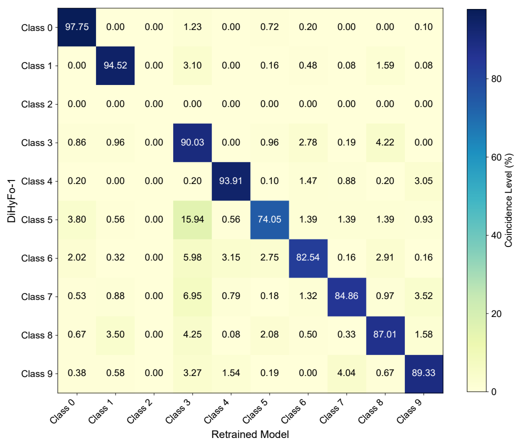


(c) DiHyFo-2 sampled models, $D_f = \{2, 3, 4\}$

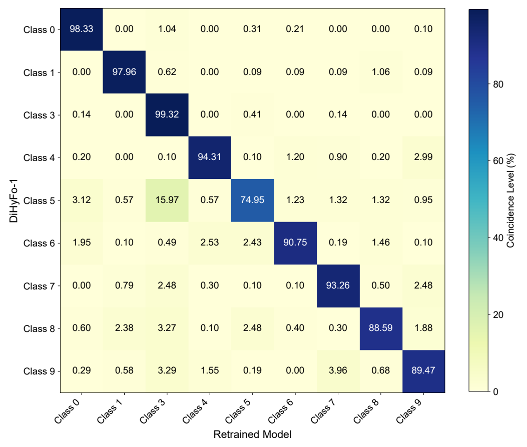


(d) DiHyFo-2 sampled models, $D_f = \{0, 1, 2, 3, 4\}$

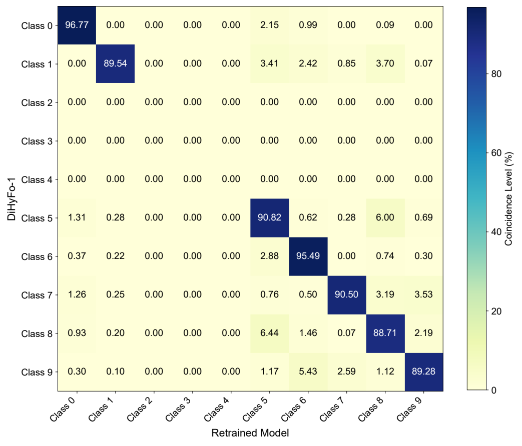
Figure 4: Selection of unlearned models sampled using DiHyFo-1 and DiHyFo-2 on MNIST.



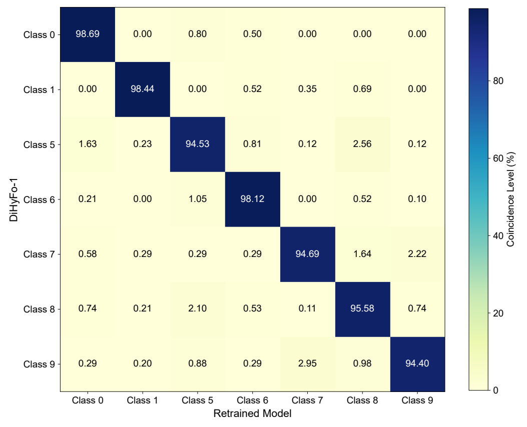
(a) Predictions on test set with $D_f = \{2\}$



(b) Predictions on D_r with $D_f = \{2\}$

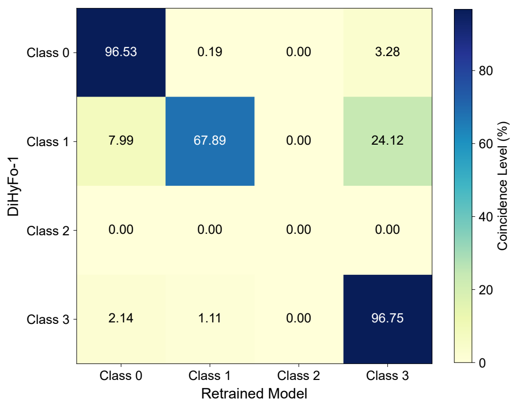


(c) Predictions on test set with $D_f = \{2, 3, 4\}$

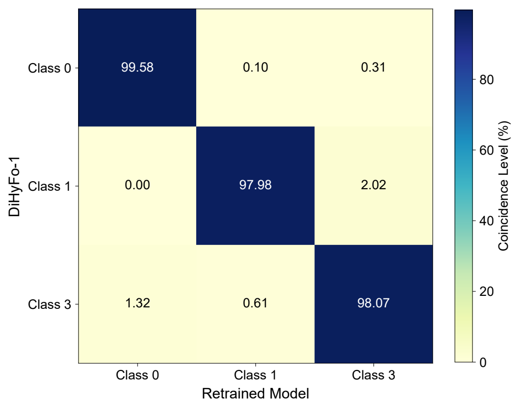


(d) Predictions on D_r with $D_f = \{2, 3, 4\}$

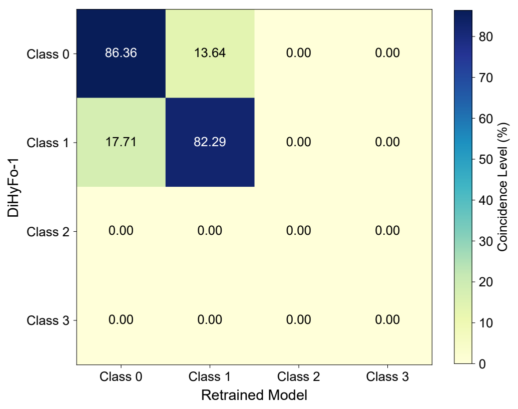
Figure 5: Comparison of predictions between an unlearned model sampled with DiHyFo-1 and the retrained model on MNIST.



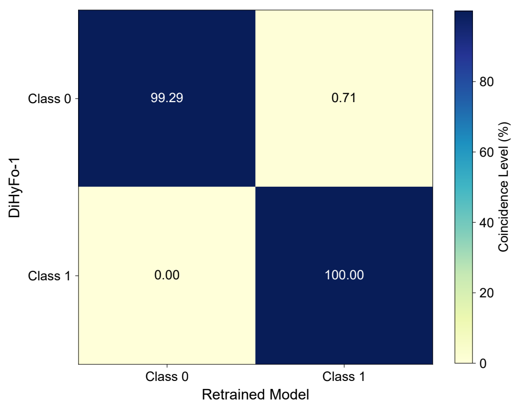
(a) Predictions on test set with $D_f = \{2\}$



(b) Predictions on D_r with $D_f = \{2\}$

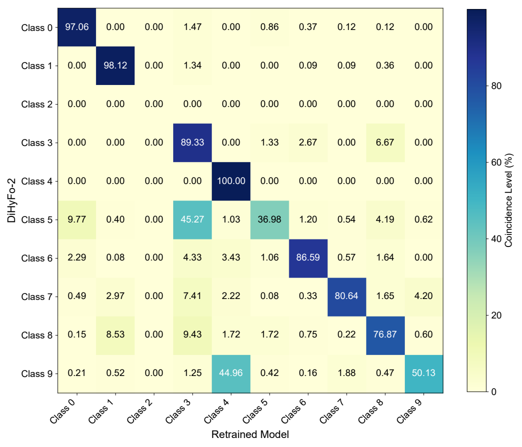


(c) Predictions on test set with $D_f = \{2, 3\}$

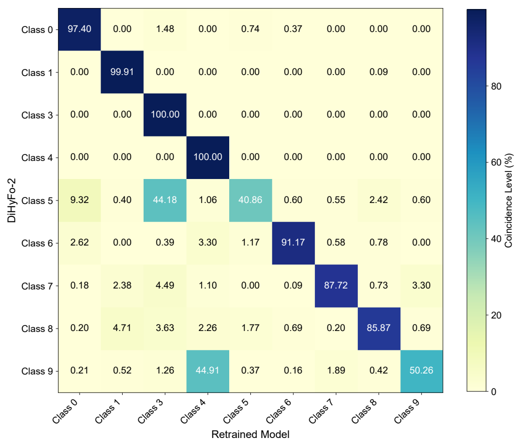


(d) Predictions on D_r with $D_f = \{2, 3\}$

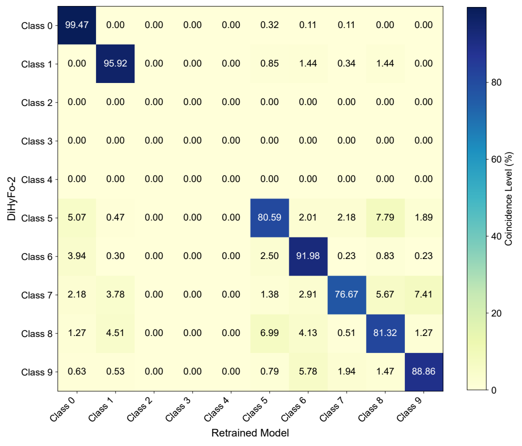
Figure 6: Comparison of predictions between an unlearned model sampled with DiHyFo-1 and the retrained model on MNIST-4.



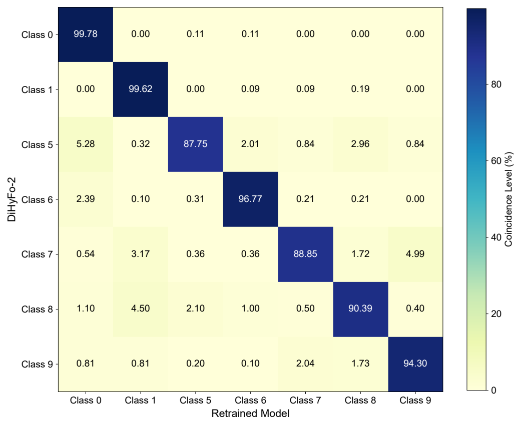
(a) Predictions on test set with $Df = \{2\}$



(b) Predictions on D_r with $D_f = \{2\}$

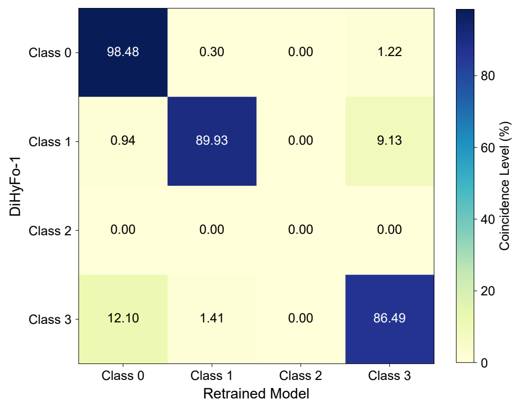


(c) Predictions on test set with $D_f = \{2, 3, 4\}$

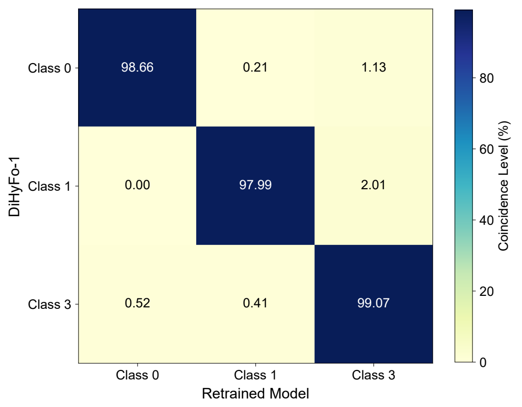


(d) Predictions on D_r with $D_f = \{2, 3, 4\}$

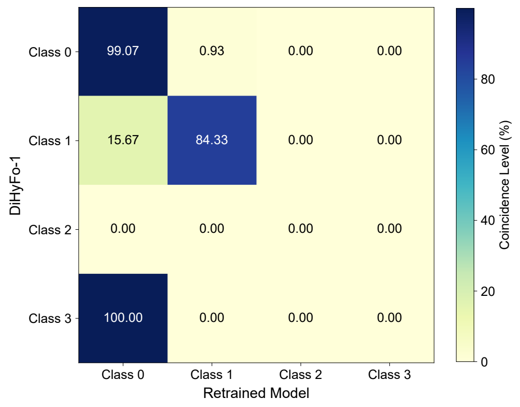
Figure 7: Comparison of predictions between an unlearned model sampled with DiHyFo-2 and the retrained model on MNIST.



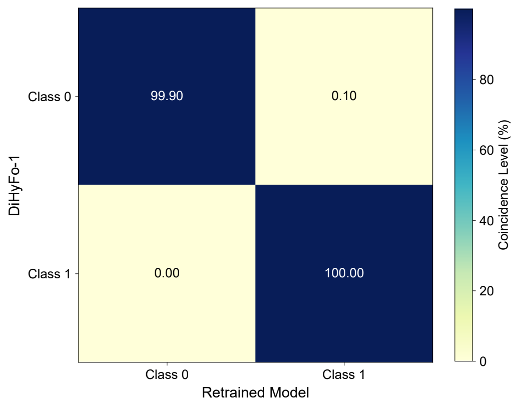
(a) Predictions on test set with $D_f = \{2\}$



(b) Predictions on D_r with $D_f = \{2\}$



(c) Predictions on test set with $D_f = \{2, 3\}$



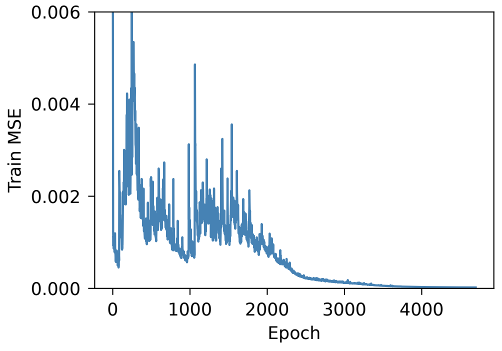
(d) Predictions on D_r with $D_f = \{2, 3\}$

Figure 8: Comparison of predictions between an unlearned model sampled with DiHyFo-2 and the retrained model on MNIST-4.

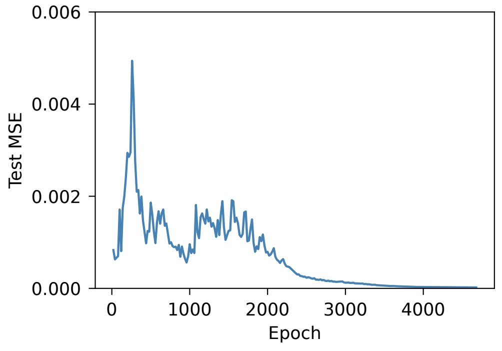
A.2 Additional training results

Learning curves for both models decrease until convergence and show a similar behavior during training and testing, Figures 9 and 10.

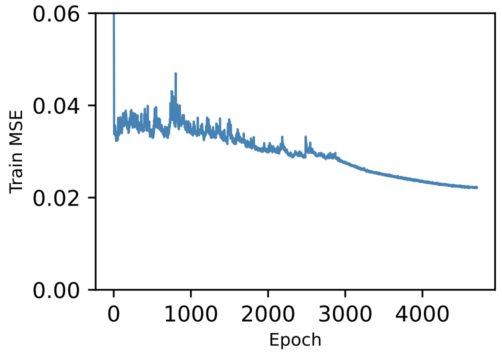
Most experiments presented positive prompt alignment scores, indicating that the generated parameters obtain class losses that closely align with the desired class losses. However, there are cases in which even when the prompt alignment increases and stabilizes it does not achieve to surpass zero, contrasting with what is shown in the correlation and direct comparison plots. As illustrated in Table 5, in several cases this behavior is due to some generated models having high negative scores that pull the average down, suggesting that the performance of the models as a group is quite varied, with most models performing good but some models performing quite poorly in terms of generating parameters for the desired losses.



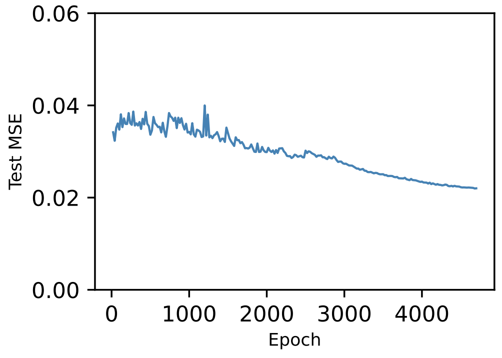
(a) DiHyFo-1, train



(b) DiHyFo-1, test



(c) DiHyFo-2, train

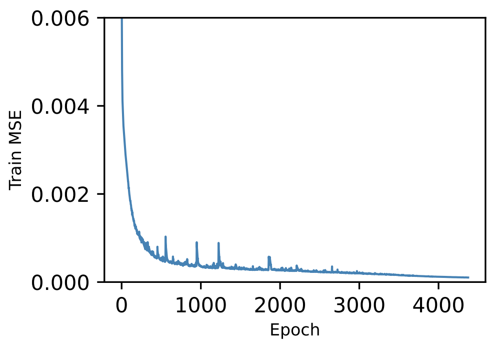


(d) DiHyFo-2, test

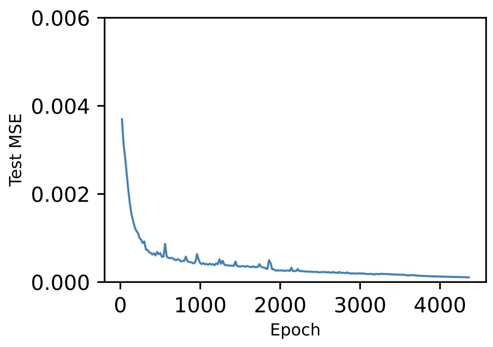
Figure 9: DiHyFo models learning curves for MNIST-4.

On the other hand, most models show high correlation between observed and desired losses, even for models with a low prompt alignment. This suggests that while the models may not always match the exact value of the losses, they generally align with the appropriate loss direction.

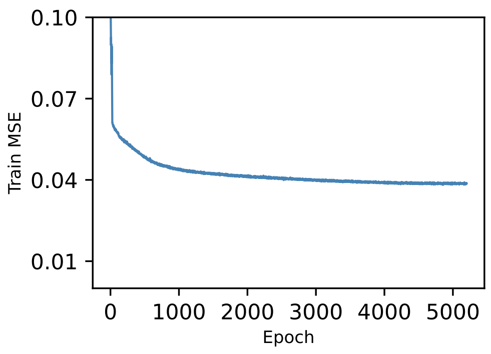
Results in Table 5 were computed using the sampled models with DiHyFo-2 on MNIST-4 during the last epoch, and can be contrasted with the corresponding direct comparison plot. The general loss direction alignment does not always translate into accurate loss matching. Considering the extreme negative values as outliers or setting a lower bound of 0 on the prompt alignment makes it resemble more closely the correlation results. Indeed, the models are capturing the direction of the relationship well but might struggle with the precision of their predictions.



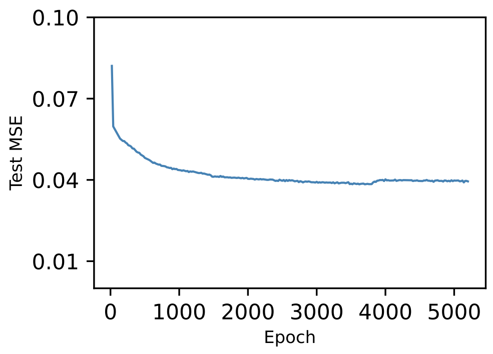
(a) DiHyFo-1, train



(b) DiHyFo-1, test



(c) DiHyFo-2, train

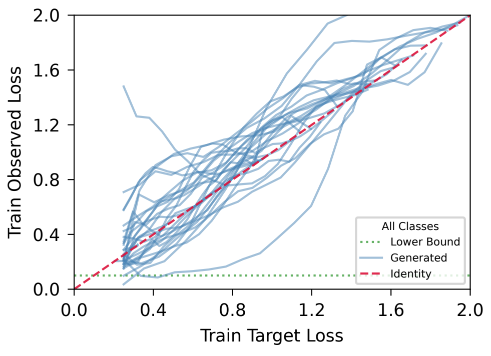


(d) DiHyFo-2, test

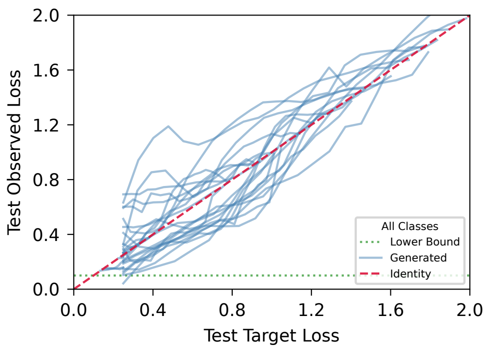
Figure 10: DiHyFo models learning curves for MNIST.

Table 5: Prompt alignment and correlation of losses for class 2 of MNIST-4 obtained by models sampled using DiHyFo-2.

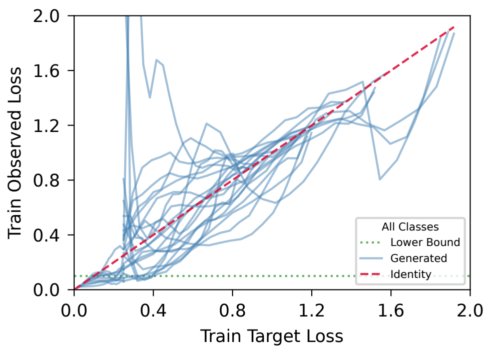
Model Number	**Prompt Alignment**	**Correlation (Pearson)**
0	-6.4499	0.4931
1	0.9443	0.9780
2	-6.8617	0.5607
3	0.8824	0.9448
4	-0.7479	0.8891
5	0.9587	0.9819
6	0.7846	0.9277
7	0.8943	0.9486
8	0.8392	0.9457
9	0.8056	0.9357
10	0.5366	0.8465
11	0.8466	0.9617
12	0.8935	0.9621
13	0.7153	0.9052
14	0.5695	0.8738
15	0.6855	0.8934
16	0.8757	0.9361
17	-1.8833	-0.2664
18	0.5878	0.9455
19	-10.7378	-0.1768
20	0.9206	0.9646
21	0.7484	0.8999
22	0.4875	0.8033
23	0.9187	0.9683
Average	-0.4911	0.7968



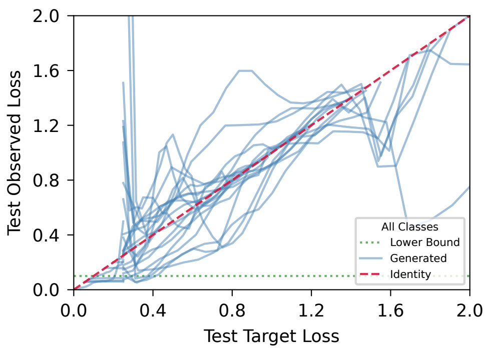
(a) Class 2 - train



(b) Class 2 - test

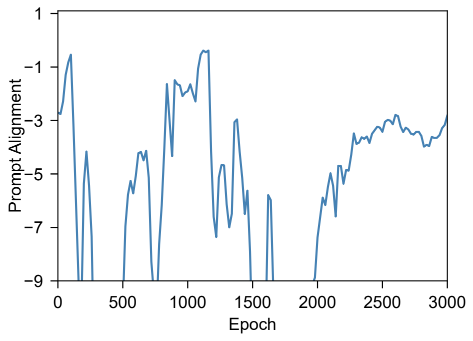


(c) Class 0 (pivot) - train

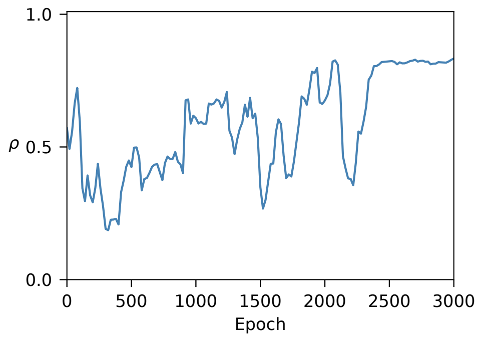


(d) Class 0 (pivot) - test

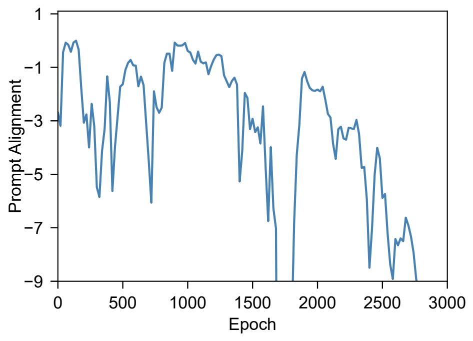
Figure 11: Comparison of losses obtained by the parameters generated with DiHyFo-1 for MNIST-4.



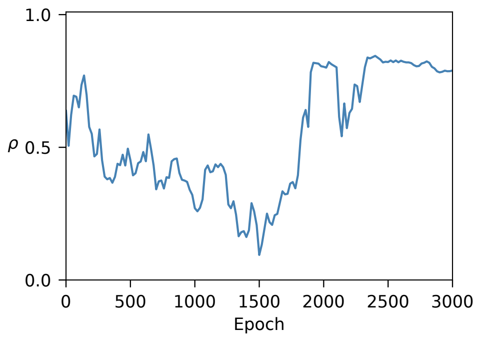
(a) Class 2



(b) Class 2

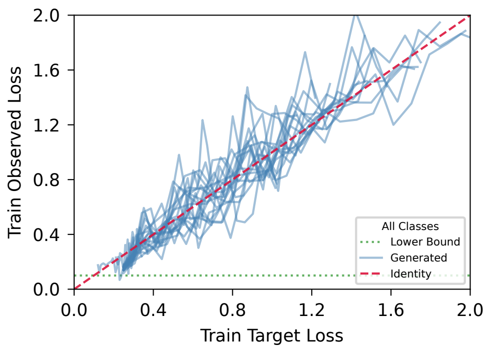


(c) Class 0 (pivot)

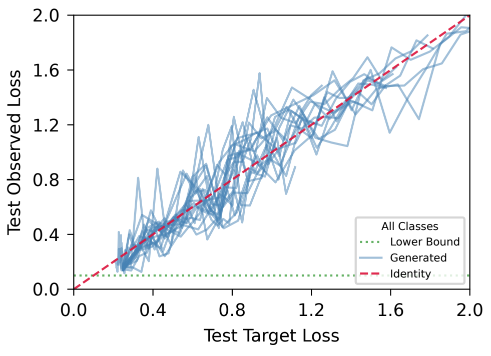


(d) Class 0 (pivot)

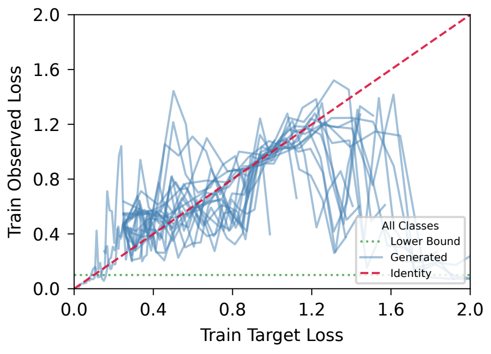
Figure 12: Prompt alignment and correlation of the losses obtained by the parameters generated with DiHyFo-1 for MNIST-4.



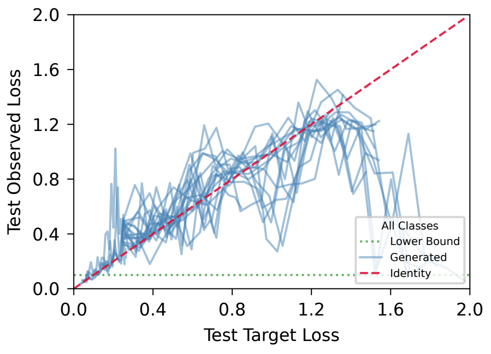
(a) Losses Comparison for class 2 - train



(b) Losses comparison for class 2 - test

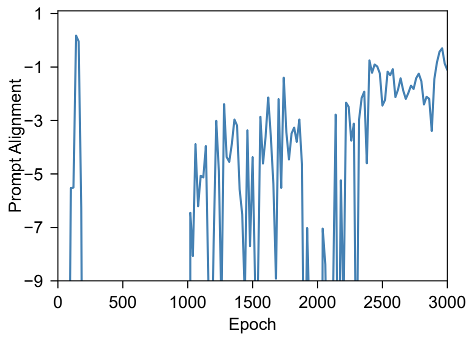


(c) Losses comparison for class 1 (pivot) - train

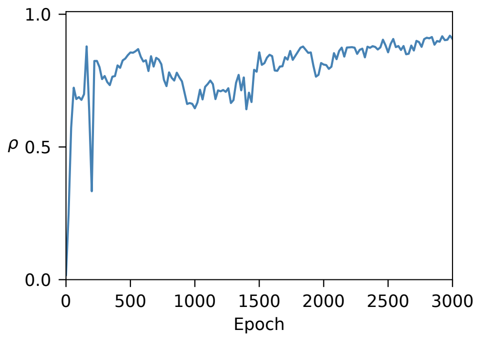


(d) Losses comparison for class 1 (pivot) - test

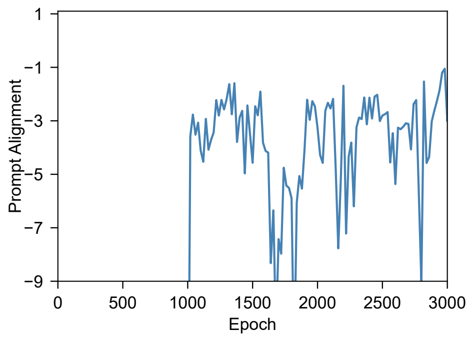
Figure 13: Comparison of losses obtained by the parameters generated with DiHyFo-2 for MNIST-4.



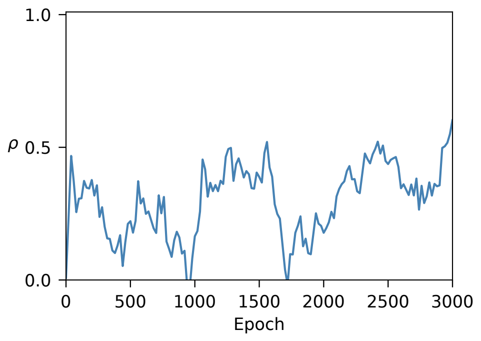
(a) Prompt alignment for class 2



(b) Correlation for class 2

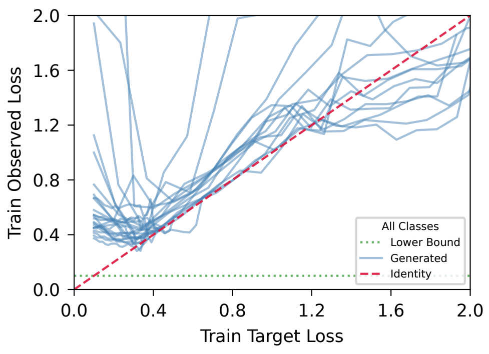


(c) Prompt alignment for class 1 (pivot)

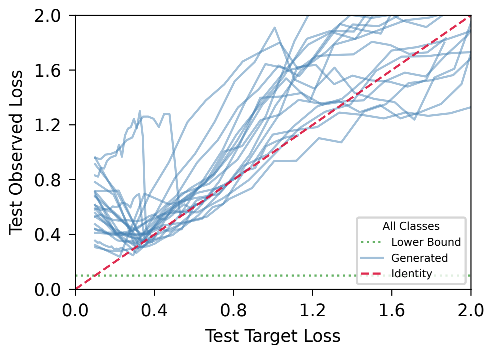


(d) Correlation for class 1 (pivot)

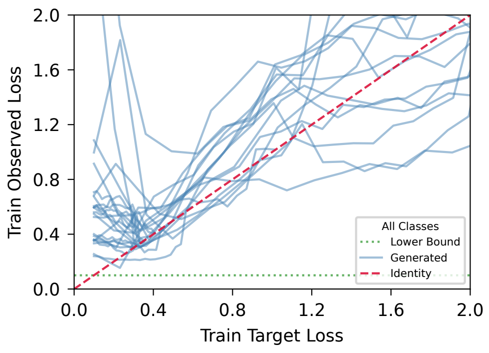
Figure 14: Prompt alignment and correlation of losses obtained by the parameters generated with DiHyFo-2 for MNIST-4.



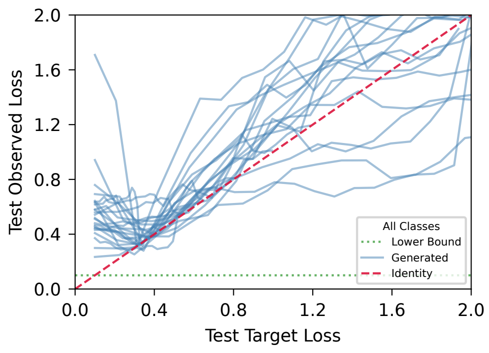
(a) Losses comparison for class 2 - train



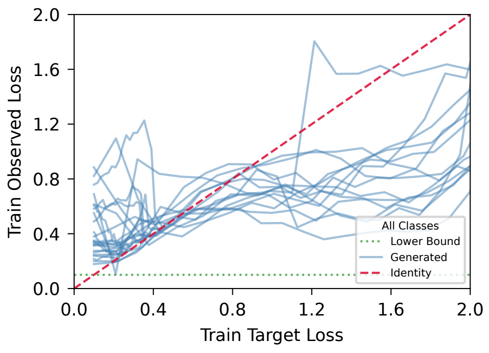
(b) Losses comparison for class 2 - test



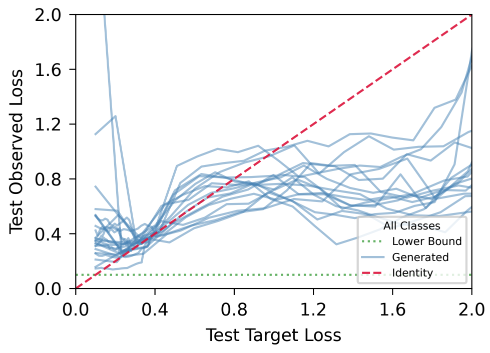
(c) Losses comparison for class 3 - train



(d) Losses comparison for class 3 - test

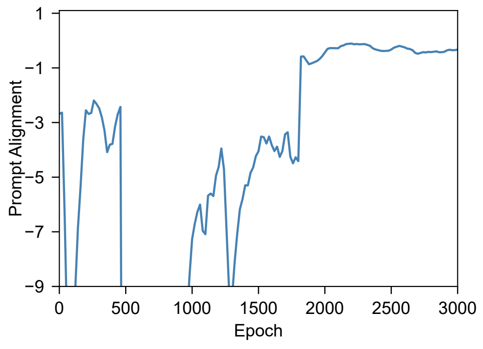


(e) Losses comparison for class 9 (pivot) - train

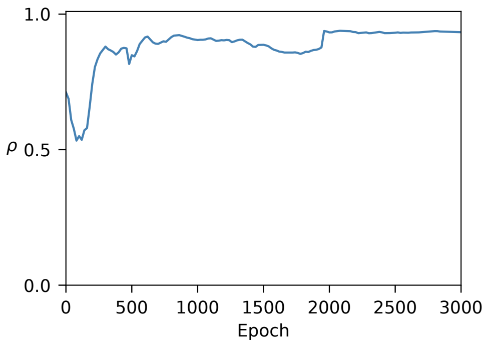


(f) Losses Comparison for class 9 (pivot) - test

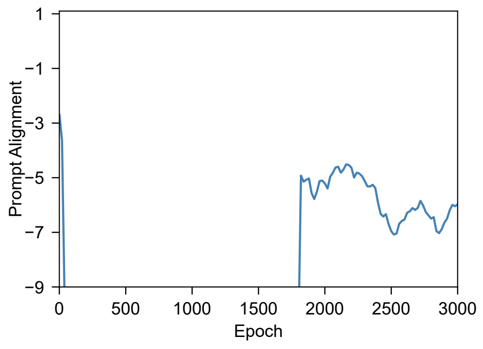
Figure 15: Comparison of losses obtained by the parameters generated with DiHyFo-1 for MNIST.



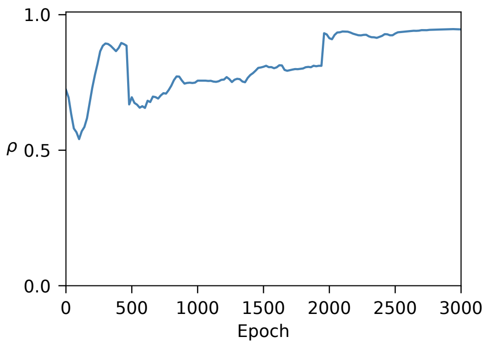
(a) Prompt alignment on class 2



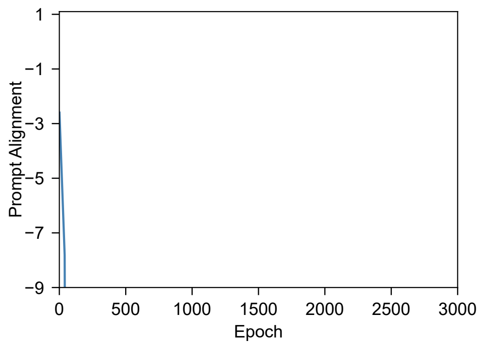
(b) Correlation on class 2



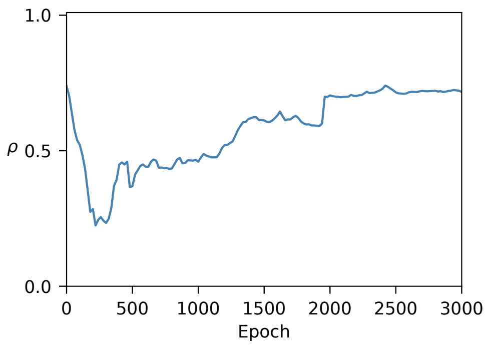
(c) Prompt alignment on class 3



(d) Correlation on class 3

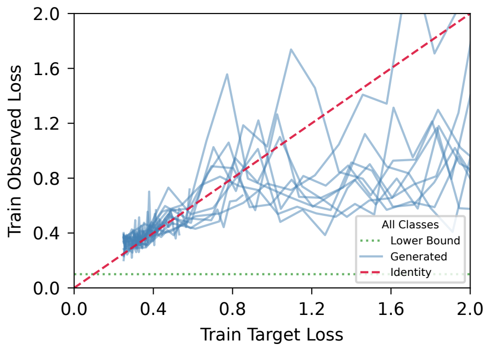


(e) Prompt alignment on class 8 (pivot)

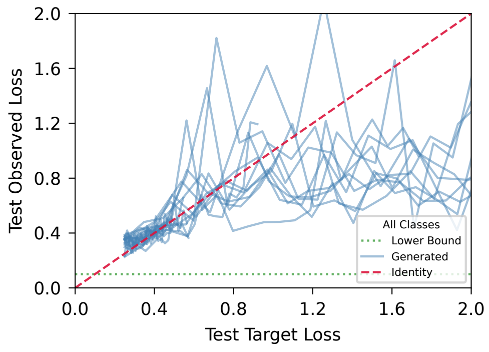


(f) Correlation on class 8 (pivot)

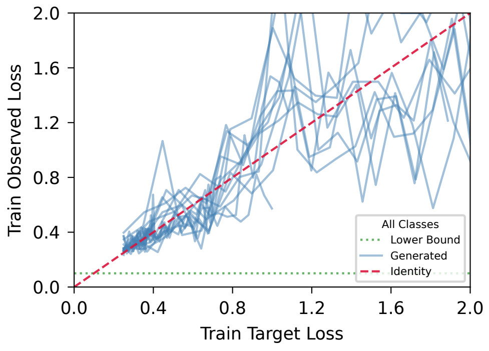
Figure 16: Prompt alignment and correlation of losses obtained by the parameters generated with DiHyFo-1 for MNIST.



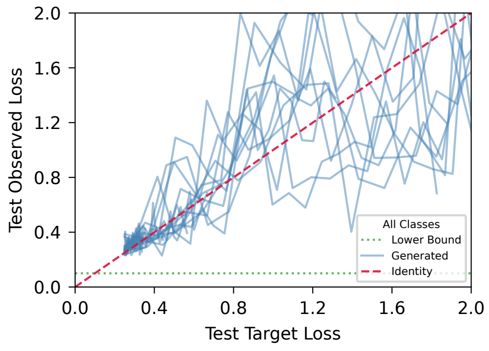
(a) Losses comparison for class 2 - train



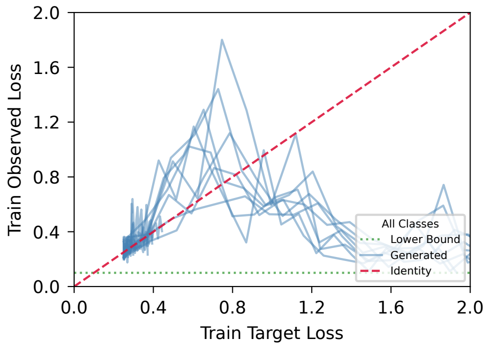
(b) Losses comparison for class 3 - test



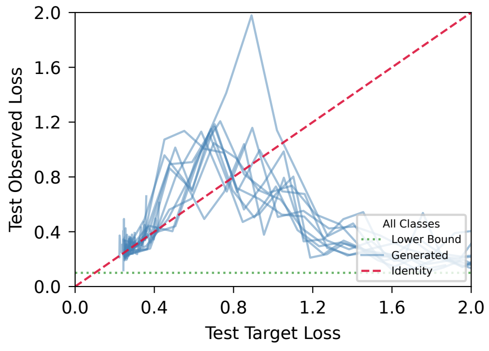
(c) Losses comparison for class 3 - train



(d) Losses comparison for class 3 - test

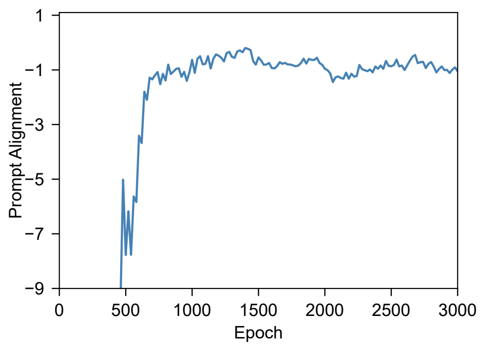


(e) Losses comparison for class 9 (pivot) - train

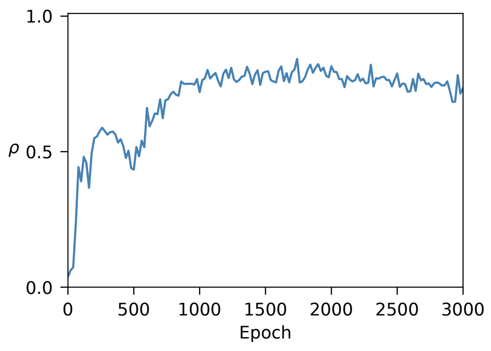


(f) Losses comparison for class 9 (pivot) - test

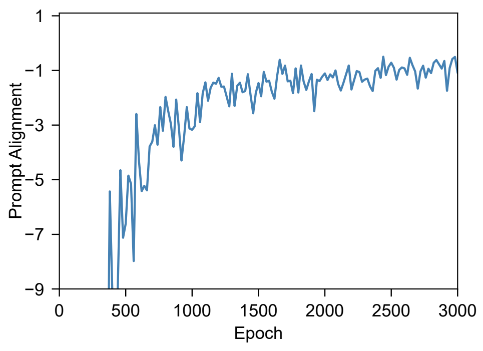
Figure 17: Comparison of losses obtained by the parameters generated with DiHyFo-2 for MNIST.



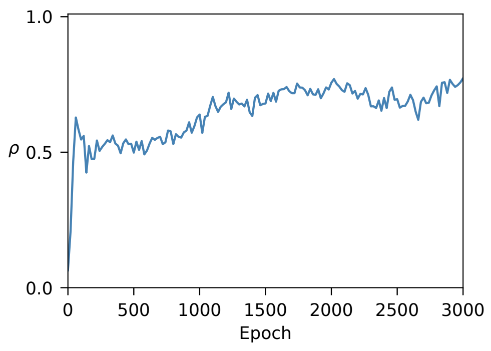
(a) Prompt alignment for class 2



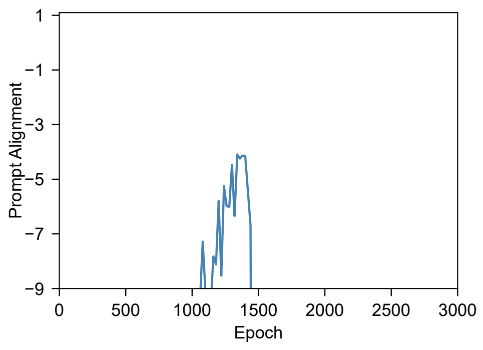
(b) Correlation for class 2



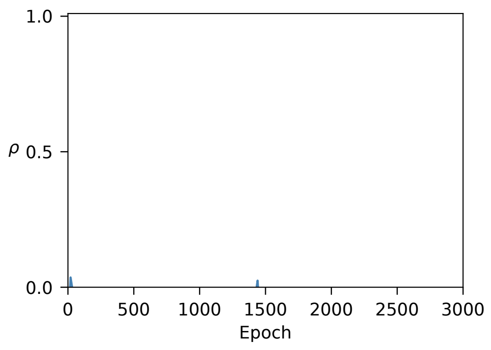
(c) Prompt alignment for class 3



(d) Correlation for class 3



(e) Prompt Alignment for class 9 (pivot)



(f) Correlation for class 9 (pivot)

Figure 18: Prompt alignment and correlation obtained by the parameters generated with DiHyFo-2 for MNIST.

A.3 Additional hypernetworks, HyperForget, and DiHyFo details

Consider a dataset $D = \{X, Y\}$ associated with a task T. In the classical deep learning framework the learnable parameters θ_F of a neural network $F(X; \theta_F)$ are obtained by solving the optimization problem in Equation 1 [6].

The searching for the optimal configuration is performed within large search spaces formed by millions of potential parameters. The input samples $x \in X$ are passed through the layers of F to obtain predictions $y^* \in Y^*$, later compared with the true labels $y \in Y$ using a loss function $L(Y, Y^*)$, which is optimized by updating parameters until convergence.

$$\min_{\theta_F} F(X; \theta_F) \tag{1}$$

Instead of directly optimizing the parameters of the main network, the hypernetwork framework, depicted in Figure 19, uses a separate network to learn to generate the parameters for the main network. Both networks are usually trained in an end-to-end differentiable manner [6, 20].

Definition A.1 Hypernetwork. *A neural network $G(C; \theta_G)$ is called a hypernetwork with learnable parameters θ_G and context input C if its output are parameters for a second neural network $F(X; \theta_F)$ that solves a task T with associated data $D = \{X, Y\}$, i.e. $\theta_F = G(C; \theta_G)$.*

The context input C for the hypernetwork contains information about the structure of the parameters of the main network that enables learning to generate its parameters.

During the forward pass at training time, the parameters θ are generated by passing C through G, and serve as input to F, which then process $x \in X$ and obtains predictions $y^* \in Y^*$. The loss $L(Y, Y^*)$ is then computed and during the backward pass, the error is back-propagated through G with the gradients of L computed with respect to θ_G. Consequently, θ_G are optimized to generate the θ_F that best solves task T. This introduces the optimization problem in Equation 2 [6].

At test time, new parameters θ_F^* can be sampled from the optimized hypernetwork and be used to make predictions with $F(X; \theta_F^*)$ on the test data.

$$\min_{\theta_G} F(X; G(C; \theta_G)) \tag{2}$$

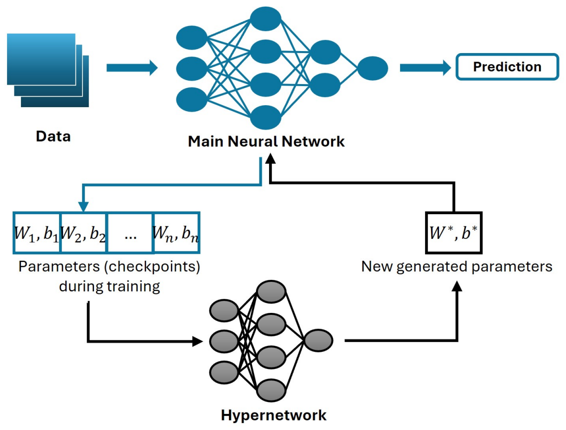


Figure 19: Hypernetwork framework.

For a classification task T with training data $D = \{X, Y\}$, solved with a learning algorithm $F(X; \theta_F)$, the associated unlearning task with forget set $D_f \subseteq D$ and retain set $D_r = D \backslash D_f$ is solved by constructing an unlearned model $U(D, D_f, F)$ that is expected to perform equivalently or similarly to a model that has been trained without the forget set, $F(X \subseteq D \backslash D_f; \theta_F)$. To forget specific subsets of classes, the associated machine unlearning objective can be described by Equation

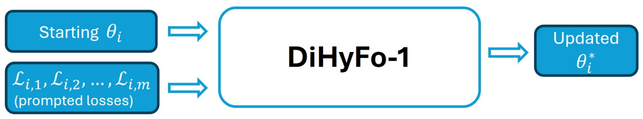
3, where the first term increases the loss on the subset of classes to forget, $\mathcal{C}_{\text{increase}}$, reducing its influence, while the second minimizes the loss on the subset of classes to retain, $\mathcal{C}_{\text{minimize}}$, preserving model performance on it.

$$\max_{\theta} \left[\sum_{c \in \mathcal{C}_{\text{increase}}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}_c(y, \hat{y})\right] - \sum_{c \in \mathcal{C}_{\text{minimize}}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}_c(y, \hat{y})\right] \right], \quad \theta \in \Theta \tag{3}$$

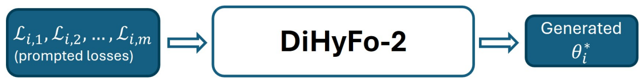
Consequently, the class unlearning task is closely related to control the influence of model parameters in the input-output interactions within the model. Instead of directly solving it, a hypernetwork can be conditioned on the specific class performance metrics, such as class losses, to construct a model capable of generating parameters that yield high-performance on D_r while obtaining low-performance on D_f, effectively forgetting the specified classes, namely, a HyperForget model. Particularly, when a diffusion model is used as hypernetwork for the unlearning model, it receives the name of Diffusion HyperForget Network (DiHyFo). Figure 21 illustrates this procedure.

Notably, unlearned models obtained using HyperForget can be interpreted and evaluated using the two interpretations of the definition of unlearning, either as an approximation to exact unlearning on the parameter space or in the output space [35, 2, 14, 15].

We have presented two mechanisms for constructing a DiHyFo model, depicted in Figure 20. DiHyFo-1 is built on the learning-to-learn framework of [37], extending it to generate parameters conditioned on class losses and to be able to learn to deoptimize, i.e., learning-to-forget. Conversely, DiHyFo-2 uses a diffusion model directly conditioned on the desired class losses.


Starting θ_i
$\mathcal{L}_{i,1}, \mathcal{L}_{i,2}, \ldots, \mathcal{L}_{i,m}$ (prompted losses)
DiHyFo-1
Updated θ_i^*

(a) DiHyFo-1


$\mathcal{L}_{i,1}, \mathcal{L}_{i,2}, \ldots, \mathcal{L}_{i,m}$ (prompted losses)
DiHyFo-2
Generated θ_i^*

(b) DiHyFo-2

Figure 20: Two implementations of DiHyFo.

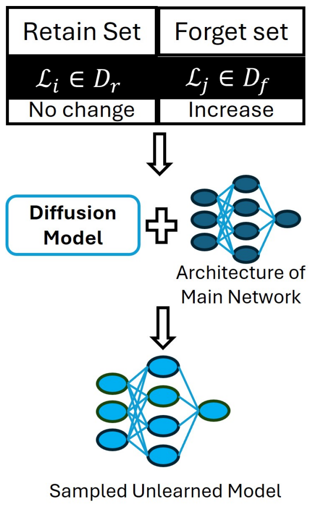


Figure 21: DiHyFo process.

DiHyFo-1's goal is to predict the distribution of updated parameters that achieve the target loss for each class, as detailed in Equation 4. The model's architecture is illustrated in Figure 1a.

$$p_G(\theta^* \mid \theta, \mathcal{L}_1^*, ..., \mathcal{L}_m^*, \mathcal{L}_1, ..., \mathcal{L}_m) \tag{4}$$

The training objective is to minimize the MSE between the original future parameters and the parameters generated by the DiT so that it learns to generate parameters that closely match the original updated parameters conditioned on the input losses, Equation 5.

$$\mathfrak{L}(G) = \mathbb{E}(\|\theta^* - G(\theta_t^*, \theta, \mathcal{L}_1^*, ..., \mathcal{L}_m^*, \mathcal{L}_1, ..., \mathcal{L}_m, t)\|_2^2) \tag{5}$$

The task of DiHyFo-2 is to predict the distribution of parameters that achieve the desired losses conditioned directly on them, reducing all previous expressions to this simpler conditioning.

A.4 Additional details on evaluation metrics

Each model must be evaluated on two key aspects. First, their generative properties as DiHy need to be assessed. This involves verifying whether the models can generate appropriate parameters for the main network, enabling it to approximate targeted performance levels on the classification task for each class.

Both DiHyFo models use MSE as training metric, and we analyze the corresponding learning curves to understand the models' learning behavior and convergence, and identify issues like overfitting or underfitting.

During evaluation, the model is prompted with a value close to the best loss found in the training dataset. It is noted in [37] that using a value slightly above or below the best loss can sometimes yield better results for certain tasks. To evaluate whether the generated parameters effectively approximate prompted loss values, we use the prompt alignment metric defined by [37] as the R^2 score between the obtained loss and the target loss in Equation 6, averaged over the batch size.

$$R^2 = 1 - \frac{\sum_{i=1}^n (\mathcal{L}_i^* - \hat{\mathcal{L}}_i^*)^2}{\sum_{i=1}^n (\mathcal{L}_i^* - \eta)^2 + \epsilon} \tag{6}$$

where ϵ is a small constant to avoid division by zero, and η is the mean of the targets:

$$\eta = \frac{1}{n} \sum_{i=1}^n \mathcal{L}_i^* \tag{7}$$

A score close to 1 indicates that the obtained loss closely aligns with the target loss. Negative values suggest worse alignment than the mean of the target values. We compute this score separately for each class to assess the model's ability to control losses across different classes simultaneously. And it is computed over 20 regularly-sampled prompts and averaged over multiple neural networks sampled with each DiHyFo, using randomly-initialized input parameters.

This metric is effective for evaluating conditional generative tasks, such as those in the learning-to-learn framework ([37]), as it measures how well the model aligns the observed losses with the target losses in terms of both direction and magnitude. However, our training datasets exhibit bidirectional loss movements, which can affect how the model learns to align the losses. In our experiments, we found that observed losses generally aligned correctly with the direction of the prompted losses, but not always in magnitude, and sometimes some generated parameters have drastic undesired performances that significantly drop the prompt alignment in comparison with the common behavior of the generated parameters. Thus, we compute the Pearson correlation between observed and target losses to assess if the model is accurately tracking the target, providing a complementary metric to the prompt alignment score.

Additionally, we directly compare the observed versus target losses by plotting them along the identity line (indicating perfect alignment) and include a lower performance bound—often set to the median or average of losses from checkpoint collection—as a reference for high performance.

The second aspect to evaluate is the DiHyFo models' ability to sample unlearned networks. We assess the unlearning properties of networks sampled with each DiHyFo by comparing them to a network retrained from scratch without the forget targets.

We compare the individually accuracy of each unlearned model and the retrained model on the retain, forget, and complete test sets. Additionally, we calculate the MIA score for each model to assess the likelihood that an adversary could infer information about the forget set from the unlearned model. Ideally, the unlearned model should have MIA scores similar to the retrained model. To compute the MIA score, we follow the implementation of [7, 13], as in Figure 22. The model being evaluated generates prediction probabilities for the retain and test sets, these probabilities are used to calculate entropy for each set, and the resulting entropies serve as features for a logistic regression with labels indicating the origin dataset of each instance. The logistic regression is trained and then used to predict the membership status of instances in the forget set. The average of these predictions provides the MIA score, reflecting how effectively it can infer whether the forget data was part of the training set of the model being evaluated.

```
1: **Input:** Model $F$, Datasets $D_r$ and $D$ (retain and test sets)
2: $P_r \leftarrow F(D_r)$
3: $P_t \leftarrow F(D)$
4: $X_r \leftarrow [\mathcal{H}(P_r), \mathcal{H}(P_t)]$                ▷ $\mathcal{H}$ is the entropy measure
5: $Y_r \leftarrow [1 \text{ (retain)}, 0 \text{ (test)}]$
6: LR $\leftarrow$ Train Logistic Regression with $(X_r, Y_r)$
7: $\hat{Y}_f \leftarrow \text{LR}(\mathcal{H}(P_r))$
8: MIA $\leftarrow \frac{1}{n_f}\sum_{i=1}^{n_f} \hat{Y}_{fi}$
9: **Output:** MIA
```

Figure 22: Pseudocode for computing MIA score.

As we employ a retrained model from scratch without the forget set as baseline, we would like the unlearned models sampled with each DiHyFo to behave as close as possible to the retrained model. Thus, to compare each unlearned model's output space against the retrained model, we measure the overlap in their predictions on each set. For parameter space comparison we compute an *unlearning score* φ using the Jensen-Shannon Divergence (JSD) [7, 35, 12], which presents a symmetric measure constructed using the Kullback–Leibler divergence (KL) between two probability distributions, P and Q. For the unlearning score, the JSD is computed between the softmax of each pair of unlearned models being compared across the entire dataset, Equation 8, in our case the sampled unlearned model using a DiHyFo and the retrained model. The closer the score value to 1, the greater the similarity in behavior between the models for the same inputs.

$$S(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\begin{aligned} \text{JSD}(P \parallel Q) &= \frac{1}{2}\text{KL}\left(P \parallel M\right) + \frac{1}{2}\text{KL}\left(Q \parallel M\right) \\ M &= \frac{P+Q}{2} \end{aligned} \tag{8}$$

$$\varphi = 1 - \text{JSD}(S(\mathbf{z}_{\text{model}_1}) \parallel S(\mathbf{z}_{\text{model}_2})) \tag{9}$$

All these metrics provide a thorough assessment of each DiHyFo model's ability to generate parameters conditioned on class losses and their suitability for unlearning tasks, which is our primary goal.

A.5 Additional details on datasets generation

Checkpoints of parameters and associated class losses are collected during multiple training runs of an MLP on MNIST. Since a simple MLP achieves good results on MNIST early in training, we randomly select a subset of classes to undersample in each run to capture a broader range of loss values. During the MLPs training, a checkpoint is randomly selected and evaluated for potential saving. Permutation augmentation is applied as in [37]. The process is illustrated in Figure 23.

As SGD is an unconstrained optimizer, we make use of heuristics to track the evolution of class losses in a constrained manner, saving checkpoints that include parameters that perform well for some classes and bad for others simultaneously.

Initially, we create bins corresponding to different loss levels and establish a maximum of examples per bin to prevent over-collecting certain loss levels while allowing for more collection on non-frequent loss levels. As the wide range of possible loss values across multiple classes leads to a combinatorial explosion of potential loss-level combinations, we consider a simplified scenario where for a classification task with m classes we use r classes as pivots ($r < m$). We only consider checkpoints when the model achieves high performance on the pivots, using an accuracy threshold $\gamma = 80$. Lowering this threshold increases variability in pivots and expands the combinatorial possibilities. The remaining $m - r$ classes are allowed to vary across the full range of possible loss values. During each training run, if a randomly selected iteration has pivots' performance over γ and the corresponding bin for remaining classes is not full, the checkpoint is saved. In this setup, the forget set can be any subset of the $m - r$ classes, while the retain set must always include the pivots.

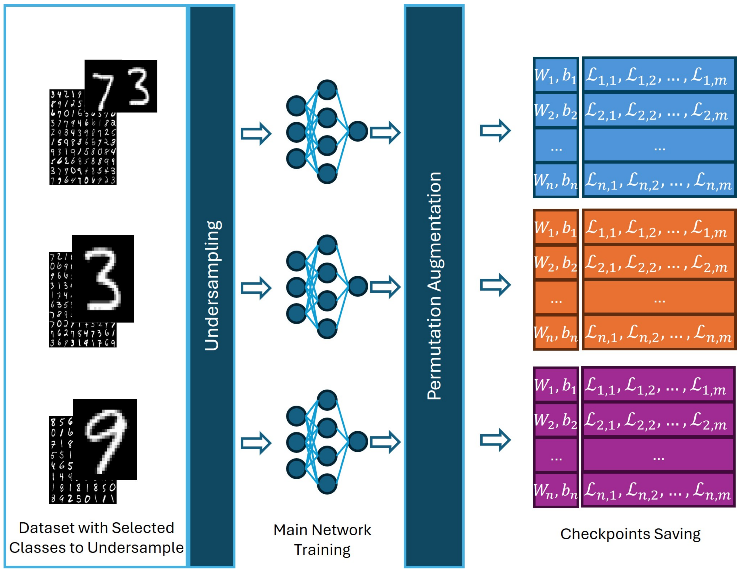
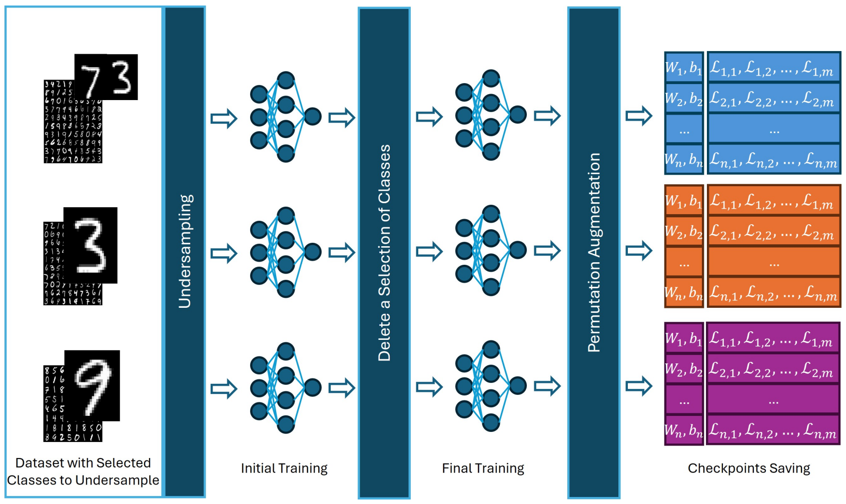

Dataset with Selected Classes to Undersample
Undersampling
Main Network Training
Permutation Augmentation
Checkpoints Saving
W_1, b_1 | $\mathcal{L}_{1,1}, \mathcal{L}_{1,2}, \ldots, \mathcal{L}_{1,m}$
W_2, b_2 | $\mathcal{L}_{2,1}, \mathcal{L}_{2,2}, \ldots, \mathcal{L}_{2,m}$
...
W_n, b_n | $\mathcal{L}_{n,1}, \mathcal{L}_{n,2}, \ldots, \mathcal{L}_{n,m}$

Figure 23: Checkpoints collection for the optimization process.

For more complicated scenarios, we relax the constrains, increasing the number of classes and using fewer pivots or none at all. Instead of using bins to categorize class-level losses, we implement checkpoint-saving conditions to balance the need for diverse loss examples with computational efficiency.

We prioritize capturing high and low performing parameter updates simultaneously, apply diverse undersampling rates and increase the random selection rate during early training epochs, while for later epochs the selection rate is reduced to focus on capturing significant shifts in loss. Also, in some runs we randomly select checkpoints across the training to ensure the models have access to examples of both the general loss evolution process and the particular moments we are interested in for the forgetting task. This checkpoint collection approach has proven effective for conditional parameter generation using DiHy [27, 54], and ensures our dataset includes key learning moments while avoiding redundancy. Figure 25 summarizes this checkpoint-saving strategy.

In the case of DiHyFo-1, it needs samples that enable it to learn bidirectional movements in class losses, either by reversing the order of checkpoint loading during DiT training or by saving checkpoints from a process with incremental losses. We chose the latter approach so that the model can learn directly from a forgetting process. To collect examples from the forgetting process, we follow a similar training procedure as described before, but at a certain point during training, we randomly delete a selection of classes to capture the associated increase in losses. This is similar to forgetting by fine-tuning the main network without the classes to be forgotten, Figure 24.



Dataset with Selected Classes to Undersample
Undersampling
Initial Training
Delete a Selection of Classes
Final Training
Permutation Augmentation
Checkpoints Saving
W_1, b_1 $\mathcal{L}_{1,1}, \mathcal{L}_{1,2}, \dots, \mathcal{L}_{1,m}$
W_2, b_2 $\mathcal{L}_{2,1}, \mathcal{L}_{2,2}, \dots, \mathcal{L}_{2,m}$
W_n, b_n $\mathcal{L}_{n,1}, \mathcal{L}_{n,2}, \dots, \mathcal{L}_{n,m}$

Figure 24: Checkpoints collection for de-optimization process.

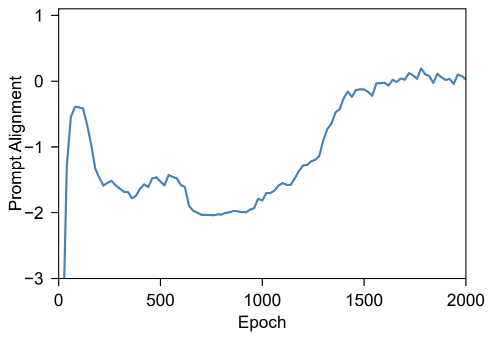
DiHyFo-2 does not requires to see a process of increments in class losses, just a variety of combinations of losses across classes, which can be done by collecting a vast amount of checkpoints from runs with different configurations.

```
1: **Input:** $\gamma$ (performance threshold), $F$ (main network), $D_T$ (task data), $\beta$ (checking threshold), $bin_1, ..., bin_r$ (loss bins), $b_1, ..., b_r$ (max bin sizes), $Y_p$ (pivots), max_checkpoints
2: $D_1 \leftarrow [X_{\text{training}}, Y_{\text{training}}]$ from $D$
3: $D_2 \leftarrow [X_{\text{test}}, Y_{\text{test}}]$ from $D$
4: $J \leftarrow$ Random sample from unique$(Y_{\text{training}})$
5: $D_1 \leftarrow$ subsample$(D_1, J)$
6: $n_{\text{checkpoints}} \leftarrow 0$
7: for epoch in $N_{\text{epochs}}$ do
8:     if $n_{\text{checkpoints}} <$ max_checkpoints then
9:         training Step $F(D_1)$
10:        $\mathbf{L_1}, ..., \mathbf{L_m} \leftarrow$ Compute class losses of $F(D_1)$ on $D_2$
11:        if $\beta <$ sampled random value then
12:            if every $L_j$ associated with $Y_p > \gamma$ then
13:                if $L_u$ in $bin_u$, not $L_u$ in $Y_p$ then
14:                    if count($bin_u$) < $b_u$ then
15:                        save checkpoint
16:                        $n_{\text{checkpoints}} \leftarrow n_{\text{checkpoints}} + 1$
17:                    end if
18:                end if
19:            end if
20:        end if
21:    end if
22: end for
```

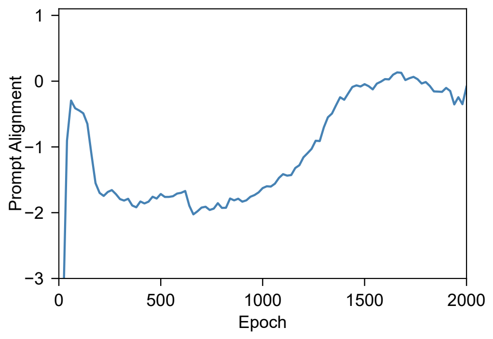
Figure 25: Pseudocode for collecting checkpoints with bins.

A.6 Some experimental observations on G.pt

Our experiments suggest that, while G.pt can learn to generate parameters using loss, prediction error, or accuracy, it is generally easier to learn with loss. Losses values typically span a narrower range than prediction error or accuracy, making them more sensitive to parameter changes. Therefore, we decided to use loss as our conditional performance metric.

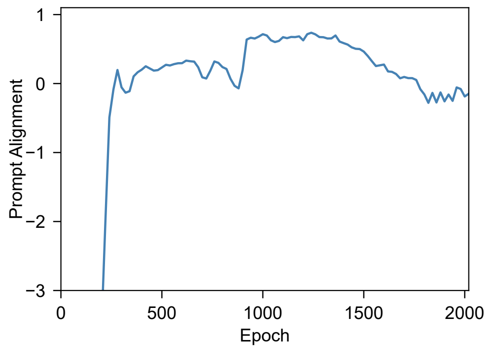


(a) Train

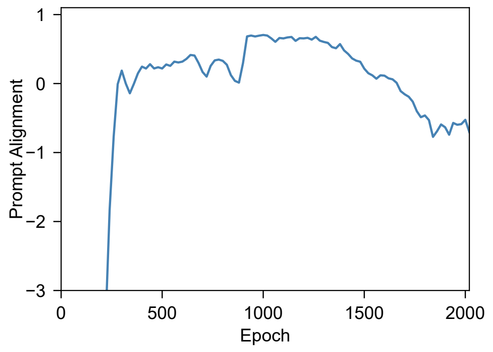


(b) Test

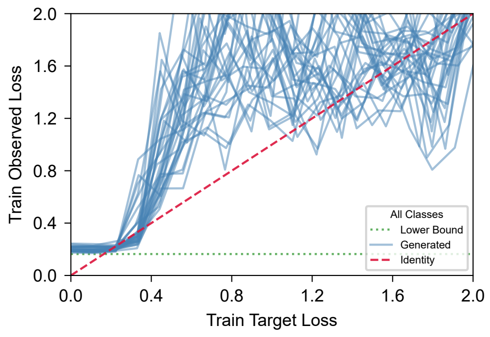
Figure 26: Evolution of G.pt prompt alignment with prediction error as metric.



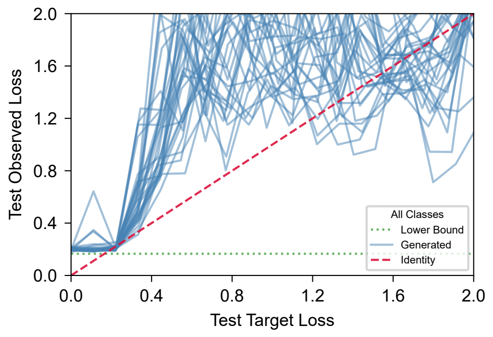
(a) Train prompt alignment



(b) Test prompt alignment



(c) Train observed vs target loss

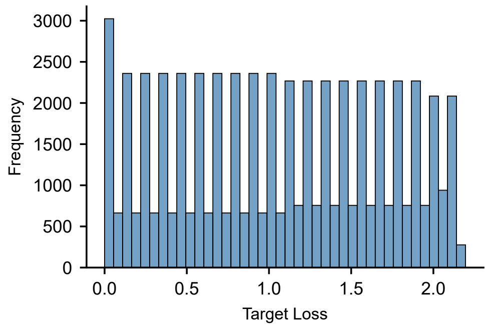


(d) Test observed vs target loss

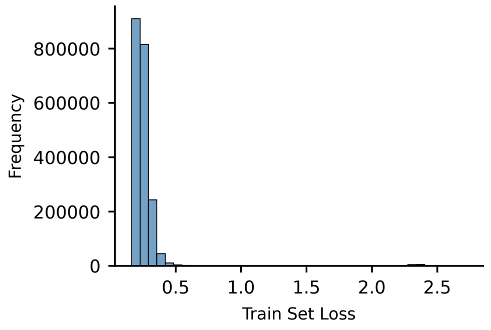
Figure 27: G.pt training results.

CIFAR-10 experimental results showed in [37] used prediction error as a conditional metric with positive results. However, for this particular experiment it was observed that the range of prediction error values obtained by the target model is contained in a relatively narrower range than usual. When comparing performance across different metrics on the same set, as shown in Figure 26, the instances with more samples and a narrower value range, typically loss, performed better.

Additionally, Figure 28 highlights a disparity between the distribution of losses found during testing and those in the training datasets used to produce the results in Figure 27. During testing, metric values tend to be more uniformly distributed, thus, effective learning requires varied examples across the entire testing range. By resampling the training set to cover a wider range of losses, Figure 29, G.pt's performance improves significantly, as shown in Figure 30. Indeed, experimental observations indicate that G.pt, while capable of generating neural network parameters for diverse losses, relies heavily on well-composed training data to perform effectively within the target metric space.

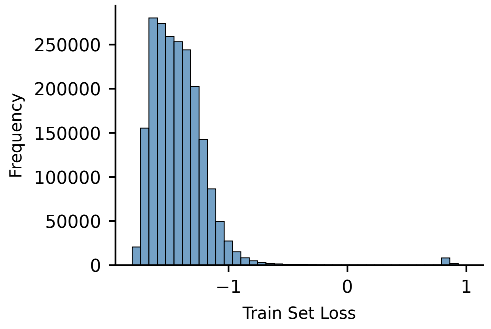


(a) Target losses

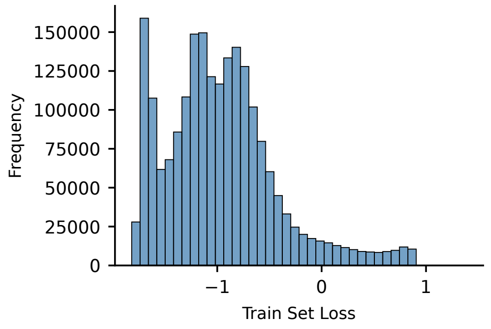


(b) Losses in train set

Figure 28: Comparison of distribution of test losses and losses in train set for G.pt.

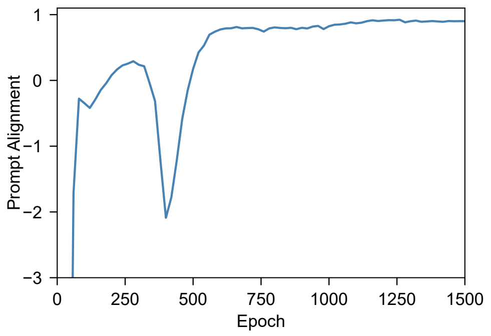


(a) Losses in train set

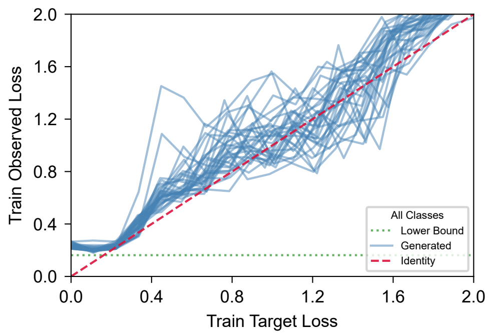


(b) Losses in resampled train set

Figure 29: Comparison of distribution of losses in train set and resampled train set for G.pt in log-scale.



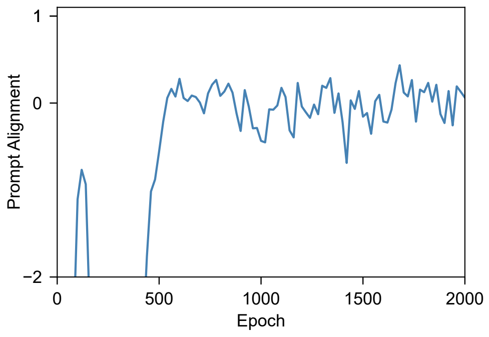
(a) Prompt alignment - train



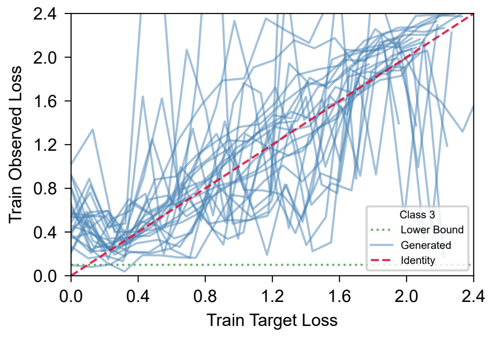
(b) Observed vs target loss - train

Figure 30: Behavior of G.pt with re-balanced loss data.

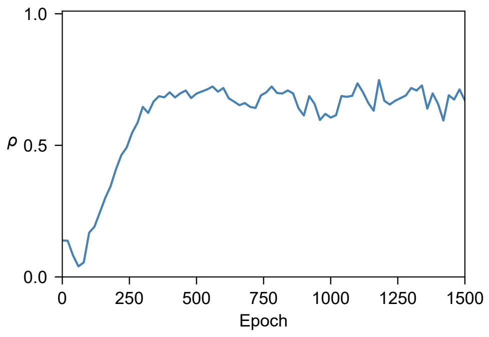
On the other hand, we evaluated G.pt capabilities when conditioned on individual classes. Figure 31 illustrates that while G.pt had potential to learn from conditioning on one class loss, it still needs additional considerations to be able to learn from multiple classes simultaneously and more to learn to forget.



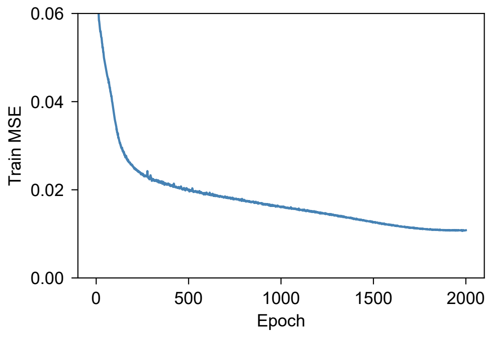
(a) Prompt alignment - train.



(b) Loss comparison - train.



(c) Correlation - train.



(d) Training learning curve.

Figure 31: Behavior of G.pt when trained conditioned on one class loss.