

InTact: Interval-based Task Activation Consolidation for Continual Learning

Patryk Krukowski
Jagiellonian University
IDEAS NCBR

patryk.krukowski@doctoral.uj.edu.pl

Jan Miksa
Jagiellonian University

Piotr Helm
Jagiellonian University

Jacek Tabor
Jagiellonian University

Paweł Wawrzyński
IDEAS Research Institute

Przemysław Spurek
Jagiellonian University
IDEAS Research Institute

Abstract

Continual learning aims to enable neural networks to acquire new knowledge without forgetting previously learned information. While recent prompt-based methods perform strongly in class-incremental settings, they remain vulnerable under domain shifts, where the input distribution changes but the label space remains fixed. This exposes a persistent problem known as representation drift. Shared representations evolve in ways that overwrite previously useful features and cause forgetting even when prompts isolate task-specific parameters. To address this issue, we introduce InTact, a method that preserves functional behavior in shared layers without freezing parameters or storing past data. InTact captures the characteristic activation ranges associated with previously learned tasks and constrains updates to ensure the network remains consistent within these regions, while still allowing for flexible adaptation elsewhere. In doing so, InTact stabilizes the functional role of important neurons rather than directly restricting parameter values. The approach is architecture-agnostic and integrates seamlessly into existing prompt-based continual learning frameworks. By regulating representation changes where past knowledge is encoded, InTact achieves a principled balance between stability and plasticity. Across diverse domain-incremental benchmarks, including DomainNet and ImageNet-R, InTact consistently reduces representation drift and improves performance, increasing Average Accuracy by up to 8 percentage points over state-of-the-art baselines.

1. Introduction

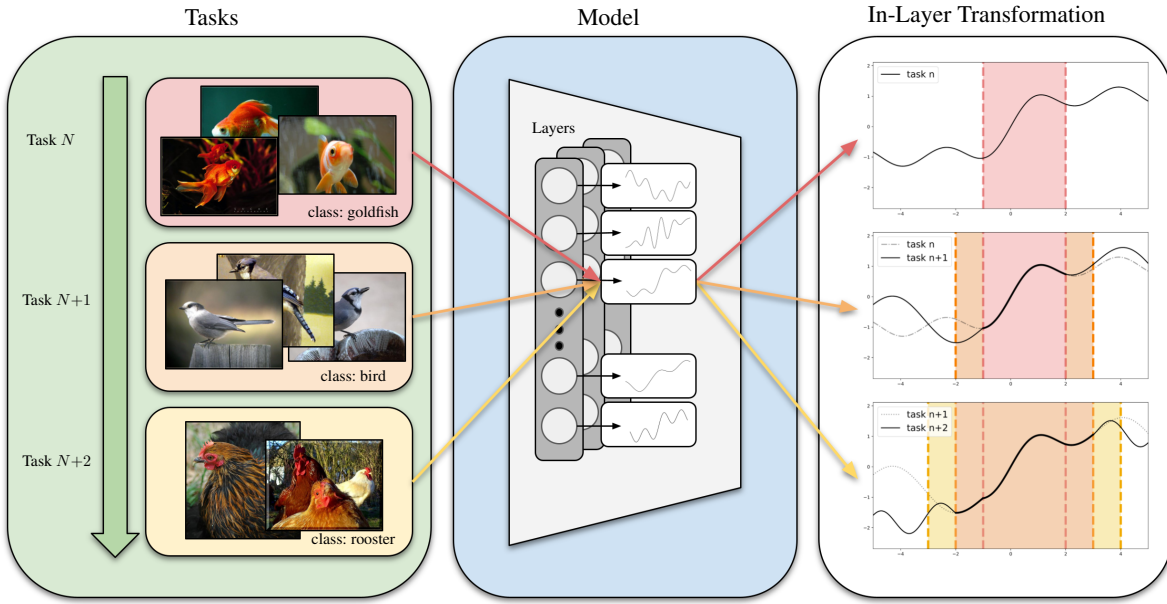
Continual learning enables neural networks to acquire new knowledge over time while retaining previously learned information. Human beings achieve this naturally by balan-

cing *stability*, the preservation of past knowledge, and *plasticity*, the ability to learn new tasks. Deep neural networks, however, struggle to maintain this balance [21]. When adapting to new data, their internal representations often *drift*, altering the learned features that were important for earlier tasks and ultimately leading to *catastrophic forgetting* [34]. Many existing approaches mitigate forgetting by constraining parameter updates, yet few explicitly control how the network’s function itself evolves across tasks. Uncontrolled changes in these representations remain a key source of instability.

This challenge of internal representation drift is especially evident in *prompt-based continual learning* [11, 47, 53, 54]. These methods currently achieve state-of-the-art performance in both *class-incremental* and *domain-incremental* settings, but their reliance on shared components makes them vulnerable to representational drift. In class-incremental learning (CIL), where tasks introduce new classes, prompt-based models approach optimal performance. Contrary, in domain-incremental learning (DIL), where the label space is fixed but input distributions shift, these models often degrade as internal features drift. Although prompting reduces interference at the embedding level, it does not explicitly regulate how shared representations evolve within trainable modules, leaving the model’s internal function unconstrained.

Recent work such as the Kolmogorov–Arnold Classifier (KAC) [17] introduces the notion of *functional locality*, constraining how output heads change across tasks. Yet, this principle has not been extended to deeper representations, leaving open the fundamental question: **how can we control where and how much a network’s internal function should adapt during continual learning?**

We address this challenge with *Interval-based Task Activation Consolidation* (InTact), a framework that enforces functional stability across all layers of the model (see



Rysunek 1. During training on task $(N+1)$, InTact preserves the functional region established by task N (pink) while allowing the model to learn new knowledge in a distinct region (orange). After training, these regions merge into an expanded protected region (yellow), defining where the layer’s transformation should remain stable in future updates. The dashed curves illustrate transformations from earlier tasks, highlighting that their behaviors remain preserved within their respective regions, while adaptation is freely allowed outside them.

Fig. 1). Rather than typical approaches, which operate in the parameters’ space, InTact regularizes activations directly, which preserves network behaviour. After learning each task, it summarizes the stable activation patterns as bounded regions, *activation intervals*, and aggregates them into multidimensional *activation hypercubes*. An interval here is a one-dimensional range capturing the lower and upper bounds of a neuron’s typical activation values, representing the region within which its response remains functionally consistent. A hypercube generalizes this concept to multiple neurons within a layer, jointly describing where the layer’s transformation is expected to be stable.

When learning new tasks, these hypercubes act as soft functional constraints: the model is encouraged to keep activations within established stable regions when appropriate, yet retains flexibility to explore new regions for adaptation. This mechanism enables continual learning that is both robust and adaptive, preserving prior functionality while supporting new knowledge acquisition. By operating at the representation level, InTact directly controls how the model’s function evolves, instead of only constraining parameters.

When integrated into state-of-the-art prompt-based frameworks, InTact improves Average Accuracy (AA) by up to 8 percentage points (p.p.) on challenging domain-incremental benchmarks such as DomainNet [37] and ImageNet-R [15]. Moreover, InTact remains effective even in settings without pretrained backbones, demonstrating its versatility as a general continual learning regularization strategy. These results show that activation-level consolidation provides a general and effective approach for robust and adaptive continual learning systems.

The main contributions of this work are:

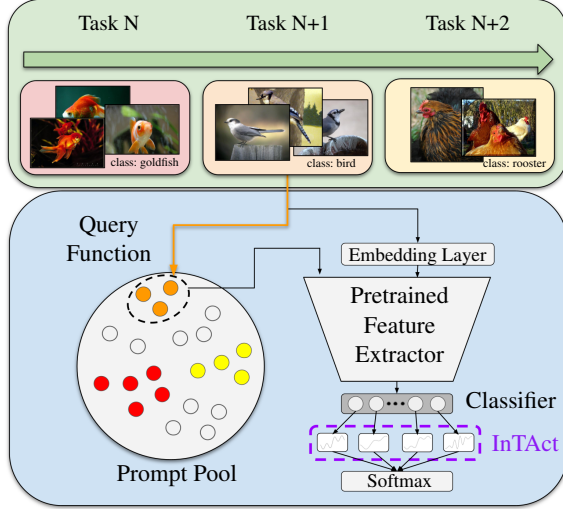
1. We propose InTact¹, a continual learning method that preserves stable functional behavior through a novel regularization guided by activation hypercubes.
2. InTact achieves a principled balance between stability and plasticity, maintaining the model’s functionality on past tasks while enabling flexible adaptation to new ones.
3. We demonstrate that InTact consistently improves state-of-the-art prompt-based methods by up to 8 p.p. in AA on challenging continual learning benchmarks.

2. Related Works

For an extended literature overview, see Supplementary Materials (SM) 6.

Continual Learning Continual learning methods [8, 33, 36] fall into replay-based, regularization-based, and parameter-isolation-based categories. Replay-based approaches store exemplars [2–4, 27, 29, 39, 50] or generate synthetic data [46, 57] to revisit past tasks. Regularization-based methods add loss terms to constrain parameter changes, leveraging current [20, 26] or prior data [1, 5, 21, 60], or null-space projections [22, 48, 51, 52]. Parameter-isolation techniques allocate task-specific subnetworks, e.g., Progressive Neural Networks [42, 43], Piggyback [32], PackNet [31], HAT [44], and SupSup [56], but require task IDs at inference.

¹Code available at: <https://github.com/pkrukowski1/InTact>



Rysunek 2. Integration of InTact with prompt-based methods. InTact stabilizes features within the pretrained extractor and constrains activation changes in the classifier, mitigating representation drift across tasks.

Prompt-Based Continual Learning Prompt-based methods leverage pretrained models for rehearsal-free continual learning. L2P [11] and DualPrompt [47] optimize adaptive prompts to guide frozen backbones. CoDA-Prompt [53] decomposes attention for modularity, and C-Prompt [54] ensures prompt stability across tasks. While prompt-based approaches reduce forgetting, they still suffer from drift in shared parameters, such as classifiers or learnable prompts, especially in DIL scenarios. InTact addresses this by regularizing these shared parameters to prevent drift of important features, maintaining stable representations across tasks. Figure 2 illustrates how InTact integrates with these prompt-based methods, stabilizing shared parameters and reducing representation drift across multiple tasks in continual learning.

Interval Arithmetic in Continual Learning Interval arithmetic [6, 35] has been applied in continual learning, for example, in InterContiNet [55], which constrains weights using intervals to ensure multi-task performance, and HINT [24], which maps intervals in the embedding space to the network via hypernetworks. In contrast, we use interval arithmetic to define activation hypercubes that capture prior task data, without propagating intervals and avoiding the wrapping effect. Each layer is then regularized to keep its transformations within these hypercubes, ensuring stable representations across tasks.

3. Method

At its core, continual learning should allow a network to evolve gradually, adjusting its learned representations lo-

cally to accommodate new knowledge while leaving its functional structure intact. In practice, however, most approaches either constrain parameters globally or rely on replaying past data, both of which limit scalability and efficiency. What is missing is a mechanism that can selectively stabilize regions of the network’s function that matter, without halting its capacity to adapt elsewhere.

Goal. Our method addresses this gap by preserving previously learned internal representations without freezing parameters or using data replay. We summarize the activation distribution of each layer l from past tasks and define a protected region, a hypercube \mathcal{H}_l capturing the central $p\%$ of neuron activations. This region represents the critical activation manifold to preserve. During new task learning, all parameters remain fully trainable, but the transformation of layer $l+1$ is regularized to stay stable for any input within \mathcal{H}_l , while regions outside it remain free to adapt. This targeted constraint maintains prior functionality and enables new knowledge acquisition, achieving a principled balance between stability and plasticity.

Idea Visualization. To illustrate the concept, Fig. 3 shows how InTact incrementally learns three segments of a Gaussian function. In the first task, the network learns to approximate the first segment and identifies the corresponding activation hypercubes. During the second task, the model learns the next segment while InTact actively protects the activations established in the first task (Fig. 3a) from drifting. This exact process continues for subsequent tasks, demonstrating how InTact successfully balances stability (preserving old segments) and plasticity (learning new ones).

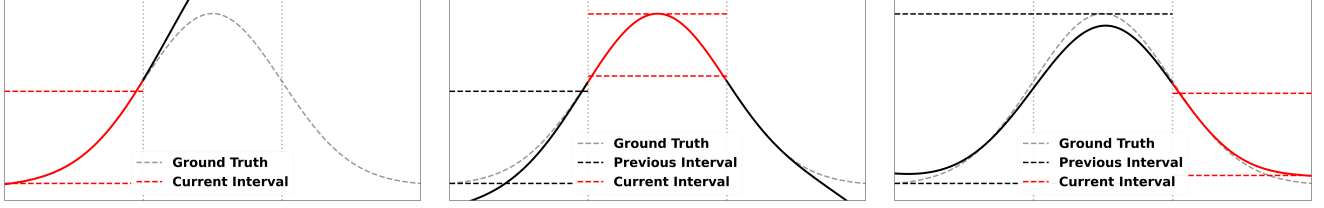
The following sections describe how these activation hypercubes are computed and updated throughout the continual learning process.

3.1. Activation Interval Representation

We aim to summarize activation distributions from past tasks without storing all raw activations. Let $\mathcal{S}_{\mathcal{H}}$ be the set of layer indices for which a hypercube from the preceding layer is available. For each task t and each layer $l \in \mathcal{S}_{\mathcal{H}}$, we build a hypercube $\mathcal{H}_{l,t}$ that captures the stable activation region at that layer. Importantly, InTact need not be applied to every layer.

Let $x_{l,t}$ denote the activation vector of the l -th layer index in $\mathcal{S}_{\mathcal{H}}$ after learning task t . To define the hypercube boundaries, we capture the central $p\%$ of the activation distribution by excluding the extreme $\alpha = \frac{100-p}{2}\%$ tails, for example, $\alpha = 5$ for $p = 90$.

For each neuron j in the selected layer, we determine the per-task activation range by computing its lower and upper



(a) Task 1: learning the first segment of the Gaussian.

(b) Task 2: adapting to the second segment while preserving the first.

(c) Task 3: completing the Gaussian while retaining prior mappings.

Rysunek 3. The model learns a Gaussian function in three sequential tasks. At each stage, InTact constrains activations within previously established hypercubes, allowing new segments to be learned without overwriting prior knowledge.

bounds, which together define the characteristic hypercube:

$$\mathcal{H}_{l,t} = [\underline{x}_{l,t}, \bar{x}_{l,t}], \quad (1)$$

where

$$\underline{x}_{l,t}[j] = \text{percentile}_{\alpha\%}(x_{l,t}[j]), \quad (2)$$

$$\bar{x}_{l,t}[j] = \text{percentile}_{(100-\alpha)\%}(x_{l,t}[j]). \quad (3)$$

Importantly, these hypercubes are computed only *after* a task has been fully learned, ensuring they reflect stable activation patterns rather than transient learning dynamics.

Cumulative Hypercube Update. As the model learns sequentially, we must preserve the functional regions from *all* previous tasks. We define a cumulative hypercube, $\mathcal{H}_l^{(t)}$, which represents the total protected region for layer index $l \in \mathcal{S}_{\mathcal{H}}$ after learning t tasks.

After the new task t is learned and its hypercube $\mathcal{H}_{l,t}$ is computed, we merge it with the previous cumulative hypercube $\mathcal{H}_l^{(t-1)}$. The update rule is a simple elementwise expansion:

$$\mathcal{H}_l^{(t)} = [\min(\underline{x}_l^{(t-1)}, \underline{x}_{l,t}), \max(\bar{x}_l^{(t-1)}, \bar{x}_{l,t})], \quad (4)$$

where $\underline{x}_l^{(t-1)}$ and $\bar{x}_l^{(t-1)}$ are the bounds of the cumulative hypercube from the previous step, and the $\min(\cdot)$ and $\max(\cdot)$ operations are applied elementwise.

As new tasks arrive, these cumulative hypercubes gradually expand to cover the full range of activations observed across all tasks. This compact representation ensures that each layer retains the functional regions essential for solving past tasks while leaving room for adaptation.

3.2. Functional Preservation via Regularization

In this subsection, we describe our regularization method. The central idea is to permit parameter updates ($\Delta\theta$) *only* if they do not alter the network’s output function for inputs corresponding to previous tasks. Our approach targets this goal directly in the activation space.

Internal Representation Drift Loss. When training on a new task t , parameter updates $\Delta\theta$ can alter the output of each layer. This representation drift changes the overall function and degrades performance on past tasks.

Our method constrains the transformation $f(\cdot; \theta_l)$ of each layer index $l \in \mathcal{S}_{\mathcal{H}}$. We require that for any input x drawn from the protected cumulative hypercube of the previous layer index, $\mathcal{H}_{l-1}^{(t-1)}$, the layer’s output remains invariant to the parameter update $\Delta\theta_l$. This condition is formalized as:

$$f(x; \theta_l + \Delta\theta_l) = f(x; \theta_l), \quad \forall x \in \mathcal{H}_{l-1}^{(t-1)}. \quad (5)$$

Consider an affine layer followed by an activation: $f(x; \theta_l) = \sigma(W_l x + b_l)$, where $\theta_l = \{W_l, b_l\}$, and W_l and b_l are learnable parameters. A direct and robust way to satisfy Eq. (5) is to enforce invariance at the pre-activation level. If the pre-activation remains constant, the output of standard activation functions (e.g., ReLU) will also be invariant. This yields a simpler, stricter constraint:

$$(W_l + \Delta W_l)x + (b_l + \Delta b_l) = W_l x + b_l, \quad \forall x \in \mathcal{H}_{l-1}^{(t-1)},$$

which simplifies to:

$$\Delta W_l x + \Delta b_l = 0, \quad \forall x \in \mathcal{H}_{l-1}^{(t-1)}. \quad (6)$$

While formulated for affine layers, this principle is readily extended to other types, such as convolutional layers (see SM 13).

Enforcing this pointwise constraint over all $x \in \mathcal{H}_{l-1}^{(t-1)}$ is intractable. Our key insight is to leverage interval arithmetic to enforce the constraint over the *entire hypercube* $\mathcal{H}_{l-1}^{(t-1)}$ simultaneously. We represent the hypercube as $[\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}]$ and reformulate the constraint (Eq. (6)) as:

$$\Delta W_l [\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}] + \Delta b_l = [0, 0]. \quad (7)$$

To translate this constraint into a differentiable loss, we compute the bounds of the resulting hypercube. Using standard interval arithmetic (detailed in SM 12), the output in-

terval for the i -th neuron is $[\tilde{x}_i + \Delta b_{l,i}, \tilde{x}_i + \Delta b_{l,i}]$, where:

$$\tilde{x}_i = (\Delta w_{l,i}^\top)^+ \underline{x}_{l-1}^{(t-1)} - (\Delta w_{l,i}^\top)^- \bar{x}_{l-1}^{(t-1)}, \quad (8)$$

$$\tilde{x}_i = (\Delta w_{l,i}^\top)^+ \bar{x}_{l-1}^{(t-1)} - (\Delta w_{l,i}^\top)^- \underline{x}_{l-1}^{(t-1)}. \quad (9)$$

Here, $(\Delta w_{l,i}^\top)^+$ and $(\Delta w_{l,i}^\top)^-$ denote the element-wise positive and negative parts of the i -th row of ΔW_l . The loss is applied only to layers for which an input hypercube is available. Let l_1 denote the first such layer. Earlier layers cannot be regularized because no input hypercube exists for them. For each $l \in \mathcal{S}_{\mathcal{H}} \setminus l_1$, we use the following penalty:

$$\mathcal{L}_{\text{IntDrift}} = \lambda_{\text{IntDrift}} \sum_{l \in \mathcal{S}_{\mathcal{H}} \setminus l_1} \sum_{i=1}^{n_l} [(\tilde{x}_i + \Delta b_{l,i})^2 + (\tilde{x}_i + \Delta b_{l,i})^2]. \quad (10)$$

This constrains the network within preserved activation regions while allowing updates outside them.

Remaining Challenges. The proposed $\mathcal{L}_{\text{IntDrift}}$ loss effectively stabilizes internal representations operating within the protected activation hypercubes of preceding layers. However, two key challenges remain. First, early feature layers, those before the first defined hypercube \mathcal{H}_{l_1} , are not constrained by this mechanism. To ensure full feature stability, a complementary regularization strategy is required, as introduced in the next section. Second, as tasks accumulate, cumulative hypercubes (Eq. (4)) may expand into empty regions between disjoint activation spaces, potentially over-constraining the model and reducing plasticity.

How InTact Differs from Existing Regularizers. Unlike parameter-based regularizers (e.g., EWC, SI) that penalize weight changes, $\mathcal{L}_{\text{IntDrift}}$ directly constrains functional change in activation space, ensuring that the network’s behavior, not just its parameters, remains consistent. In contrast to output-level methods like LwF or those relying on discrete point sampling [7], our interval-based formulation enforces stability across continuous activation regions, offering a principled and tractable guarantee of functional preservation. Hypercubes act as compact, data-free summaries of past tasks, avoiding replay or architectural overhead and making InTact lightweight, privacy-preserving, and easily integrable into existing continual learning frameworks.

Activation Compactness Regularization. To prevent the excessive growth of activation hypercubes and improve representational efficiency, we introduce a compactness regularization term. This loss, \mathcal{L}_{Var} , encourages activations from the current task to remain concentrated in a smaller, denser region of the feature space.

For a layer index $l \in \mathcal{S}_{\mathcal{H}}$ and the N_t samples of the current task t , let $x_{i,l-1}$ be the input to the layer with index l for sample i (i.e., the activation from the layer with index $l-1$). We compute the empirical mean activation $\bar{f}_l^{(t)}$ using the current model $f(\cdot; \theta_l^{(t)})$:

$$\bar{f}_l^{(t)} = \frac{1}{N_t} \sum_{i=1}^{N_t} f(x_{i,l-1}; \theta_l^{(t)}). \quad (11)$$

The activation dispersion $\mathcal{V}_l^{(t)}$ is the mean squared deviation from this mean:

$$\mathcal{V}_l^{(t)} = \frac{1}{N_t} \sum_{i=1}^{N_t} \|f(x_{i,l-1}; \theta_l^{(t)}) - \bar{f}_l^{(t)}\|_2^2. \quad (12)$$

The compactness regularization is the sum of these dispersions over the relevant layer indices:

$$\mathcal{L}_{\text{Var}} = \lambda_{\text{Var}} \sum_{l \in \mathcal{S}_{\mathcal{H}}} \mathcal{V}_l^{(t)}, \quad (13)$$

where $\lambda_{\text{Var}} > 0$ controls the strength of the penalty. Minimizing \mathcal{L}_{Var} forces tighter clustering of activations, resulting in more compact hypercubes. This ensures our functional regularization ($\mathcal{L}_{\text{IntDrift}}$) operates over smaller, more meaningful regions, mitigating hypercube over-expansion and preserving model plasticity for future tasks.

Inter-Task Alignment Regularization. The cumulative hypercube update rule (Eq. (4)) is prone to over-regularizing inactive space, particularly when activation regions from sequential tasks are disjoint. If a new task’s hypercube is far from the previous cumulative region, the rule expands to cover the wide gap between them, unnecessarily constraining a "free" region that contains no meaningful activations. This severely restricts plasticity, especially in deeper layers.

To prevent this and promote smoother transitions, we introduce an inter-task alignment regularization term, $\mathcal{L}_{\text{Align}}$. This loss encourages the centers of the newly learned activation hypercubes ($\mathcal{H}_{l,t}$) to remain close to the centers of the hypercubes calculated from preceding tasks ($\mathcal{H}_{l,t-1}$).

We define the center of a hypercube \mathcal{H}_l as $c_l = (\bar{x}_l + \underline{x}_l)/2$, and the radius as $r_l = (\bar{x}_l - \underline{x}_l)/2$. The alignment loss penalizes the squared distance between the new center $c_{l,t}$ and the previous center $c_{l,t-1}$:

$$\mathcal{L}_{\text{Align}} = \lambda_{\text{Align}} \sum_{l \in \mathcal{S}_{\mathcal{H}}} \frac{\|c_{l,t} - c_{l,t-1}\|_2^2}{r_{l,t-1}^{\text{mean}} + \varepsilon}. \quad (14)$$

Here, ε is a small constant for numerical stability, and the denominator provides an adaptive scaling mechanism. We use the mean radius of the previous task’s hypercube across its d_l dimensions, $r_{l,t-1}^{\text{mean}} = \frac{1}{d_l} \sum_{j=1}^{d_l} r_{l,t-1}^{(j)}$.

Intuitively, the radius $r_{l,t-1}^{\text{mean}}$ reflects the spread of past representations. A small radius indicates that past tasks occupied a narrow, critical region, thus scaling the penalty up and enforcing stricter alignment. Conversely, a large radius implies widely spread representations, weakening the constraint and granting the model more freedom to adapt. This adaptive scaling efficiently stabilizes the model’s representational space while preserving flexibility.

Feature Distillation Loss. To stabilize the early feature layers (those with indices $l < l_1$), we introduce a feature distillation loss, $\mathcal{L}_{\text{Feat}}$. This loss is applied at the layer with index l_{start} , where the first hypercube $\mathcal{H}_{l_{\text{start}}}$ is defined. For each input sample x_i , we compare its feature representation from the current model $f_{\text{Feat}}^{(t)}(x_i)$ against the frozen representation from the previous model $f_{\text{Feat}}^{(t-1)}(x_i)$. Crucially, we do this only on a subset of feature dimensions indicated by a binary mask M :

$$\mathcal{L}_{\text{Feat}} = \frac{\lambda_{\text{Feat}}}{N_t} \sum_{i=1}^{N_t} \frac{\left\| (f_{\text{Feat}}^{(t)}(x_i) - f_{\text{Feat}}^{(t-1)}(x_i)) \odot M \right\|_2^2}{s + \varepsilon}, \quad (15)$$

where s is the number of active entries (1s) in M , and $s + \varepsilon$ normalizes the loss magnitude.

Unlike standard knowledge distillation, which often over-restricts the model by constraining the entire feature vector, $\mathcal{L}_{\text{Feat}}$ implements a *selective stabilization*. It preserves only a controlled subset of features deemed important from past tasks, leaving the remaining dimensions free to adapt to the new task. This targeted approach preserves learned structure while maintaining the flexibility required to acquire new information.

Final Regularization Objective. Our full training objective combines the base continual learning loss $\mathcal{L}_{\text{Task}}$ with the proposed regularization terms:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{Task}} + \mathcal{L}_{\text{IntDrift}} + \mathcal{L}_{\text{Var}} + \mathcal{L}_{\text{Align}} + \mathcal{L}_{\text{Feat}}. \quad (16)$$

Each component targets a different aspect of the stability–plasticity trade-off: $\mathcal{L}_{\text{IntDrift}}$ constrains changes in the model’s internal function across tasks, \mathcal{L}_{Var} keeps activation distributions compact to prevent representational over-expansion, $\mathcal{L}_{\text{Align}}$ regulates the growth of protected hypercubes over time, and $\mathcal{L}_{\text{Feat}}$ stabilizes early feature layers while allowing adaptation in later ones. Their effects are illustrated in Fig. 4.

Because these regularizers act directly on activations, InTAct can be seamlessly integrated into diverse continual learning methods. An algorithm of the training procedure appears in SM 14, and ablations isolating each term are reported in Section 4.

4. Experiments

We evaluate the effectiveness of InTAct across multiple benchmarks under both CIL and DIL scenarios. Our experiments compare InTAct with regularization-based and prompt-based continual learning methods, demonstrating consistent performance gains across different architectures and learning paradigms. Additional results showing the effectiveness of InTAct without pretrained backbones are provided in SM 15. Moreover, SM 16 presents a comparison with a broader range of methods for the CIL scenario.

4.1. Experimental Setup

Datasets. We evaluate the effectiveness of InTAct on several datasets. (1) For regularization-based approaches: Split MNIST, Split FMNIST, and Split CIFAR-10; (2) For prompt-based approaches: Split CIFAR-100, DomainNet, and ImageNet-R. Except for Split CIFAR-100, we evaluate performance under both CIL and DIL scenarios. Full dataset and task details are provided in SM 8.

Architectures. For the Split MNIST and Split FMNIST datasets, we use a multilayer perceptron (MLP) with three hidden layers of 400 neurons each. For Split CIFAR-10, we employ a pretrained ResNet-18 [14] as the feature extractor, where the last residual block is unfrozen and adapted during continual learning. For Split CIFAR-100, ImageNet-R, and DomainNet, we follow the CODA-Prompt, using a ViT-B/16 encoder [9] pretrained on ImageNet-1K [41]. We apply our method to activations of the last layer before Softmax as presented on Figure 2. Additional architectural and implementation details are provided in SM 8.

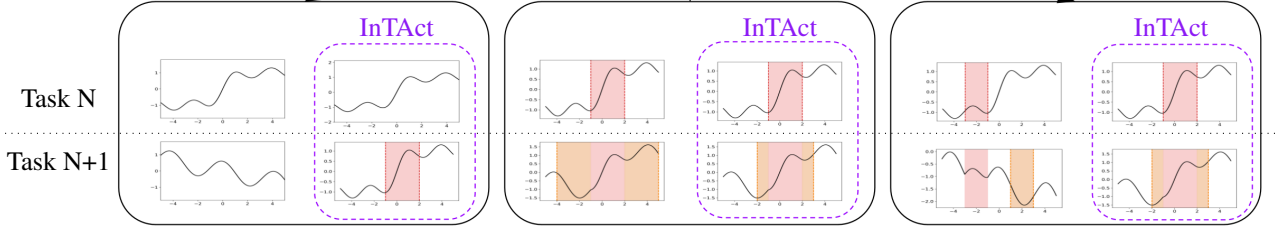
Metrics. We evaluate the performance of our method using two standard continual learning metrics: Average Accuracy (AA) and Average Forgetting (AF). Formal definitions of these metrics are provided in SM 11.

4.2. Results

This subsection discusses evaluation results of InTAct across diverse continual learning benchmarks. Overall, the results highlight InTAct’s ability to enhance stability while preserving plasticity across varied scenarios.

Results for Prompt-based Methods (DIL). Overall, InTAct consistently delivers the strongest performance across all prompt-based methods in the DIL setting, substantially improving both AA and reducing AF. The detailed results are reported in Tab. 1 for the 6-task DomainNet benchmark, where tasks share labels but differ in visual domains. Integrating InTAct yields gains of 4–6 percentage points in AA and decreases AF by up to 60%, with the largest improvements observed when combined with DualPrompt (AA:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{Task}} + \mathcal{L}_{\text{IntDrift}} + \mathcal{L}_{\text{Feat}} + \mathcal{L}_{\text{Var}} + \mathcal{L}_{\text{Align}}$$



Rysunek 4. Illustration of how the InTact loss components (Eq. 23) regulate representation updates between consecutive tasks. $\mathcal{L}_{\text{IntDrift}}$ and $\mathcal{L}_{\text{Feat}}$ preserve previously learned in-layer transformations within protected activation hypercubes (pink), maintaining consistency of past feature space while allowing adaptation in new regions. \mathcal{L}_{Var} constrains the expansion of activation hypercubes so that new regions (orange) remain compact and balanced across tasks, preventing uncontrolled growth. $\mathcal{L}_{\text{Align}}$ enforces smooth transitions by aligning consecutive hypercubes and avoiding fragmentation of functional regions. Purple outlines denote the InTact-regularized case, where the representation remains stable and well-aligned across tasks.

50.87% \rightarrow 56.83%, AF: 12.13 \rightarrow 4.66). Similar trends for L2P and CODA-Prompt confirm that InTact stabilizes shared prompt representations effectively across different prompting strategies without additional replay mechanisms or architectural overhead.

Tabela 1. Results on the 6-task (345 classes each) DomainNet benchmark (DIL setting). We report the mean and standard deviation of AA and AF averaged over 3 random seeds.

Method	AA (\uparrow)	AF (\downarrow)
Upper-Bound	79.65	—
L2P w/ InTact	48.44 \pm 0.09	14.56 \pm 0.05
	53.44 \pm 0.02	6.95 \pm 0.99
DualPrompt w/ InTact	50.87 \pm 0.31	12.13 \pm 0.2
	56.83 \pm 0.28	4.66 \pm 0.11
CODA-P w/ InTact	52.52 \pm 0.22	11.54 \pm 0.16
	57.36 \pm 0.25	6.49 \pm 0.11

Tab. 2 further shows consistent benefits on the 15-task ImageNet-R benchmark, a more challenging scenario with greater domain diversity and more incremental steps. Note that integrating InTact not only preserves prior knowledge but can also enhance performance on earlier tasks (e.g., AF drops from 1.01 to -0.36 with DualPrompt), highlighting the method’s ability to stabilize representations in prompt-based architectures. Across both benchmarks, InTact improves stability while maintaining plasticity, all without replay buffers, replay, or architectural changes.

Across both DomainNet and ImageNet-R in the DIL setting, InTact consistently enhances stability without hindering plasticity. These gains are achieved without additional replay mechanisms or architectural modifications.

Tabela 2. Results on the 15-task (200 classes each) ImageNet-R benchmark (DIL setting). We report the mean and standard deviation of AA and AF averaged over 5 random seeds.

Method	AA (\uparrow)	AF (\downarrow)
Upper-Bound	77.13	—
L2P w/ InTact	50.34 \pm 0.26	1.46 \pm 0.19
	58.34 \pm 0.24	0.02 \pm 0.1
DualPrompt w/ InTact	53.31 \pm 0.36	1.01 \pm 0.17
	61.85 \pm 0.45	−0.36 \pm 0.17
CODA-P w/ InTact	61.21 \pm 0.49	0.68 \pm 0.11
	66.11 \pm 0.24	0.35 \pm 0.07

Results for Prompt-based Methods (CIL). Tab. 3 reports AA and AF on ImageNet-R for 10- and 20-step CIL scenarios. Across all baselines, integrating InTact consistently improves or matches performance, confirming its robustness and plug-and-play compatibility with diverse prompt-based continual learning frameworks. For L2P, InTact delivers clear AA gains and lower standard deviations, indicating that our regularization stabilizes optimization and promotes smoother learning. DualPrompt also benefits, maintaining high accuracy with reduced forgetting and improved consistency. The largest improvements appear for CODA-Prompt, where InTact boosts AA by nearly one point in both settings, showing that functional preservation effectively complements prompt-based methods.

This improvement is most pronounced in CODA-Prompt due to its shared, trainable prompt pool, composed of prompt tokens (P_m), keys (K_m), and attention vectors (A_m), which are jointly updated across tasks. During continual learning, this shared structure can easily drift, causing instability in both the generated prompts and the clas-

Tabela 3. Results on the ImageNet-R dataset (CIL setting). We report the mean and standard deviation of AA and AF averaged over 5 random seeds.

Method	10 steps		20 steps	
	AA (\uparrow)	AF (\downarrow)	AA (\uparrow)	AF (\downarrow)
Upper-Bound	77.13	–	77.13	–
L2P w/ InTAct	69.29 \pm 0.73 2.03 \pm 0.19	2.89 \pm 0.46	65.89 \pm 1.30 1.24 \pm 0.14	1.30 \pm 0.13
DualPrompt w/ InTAct	71.32 \pm 0.62 1.71 \pm 0.24	1.68 \pm 0.07	67.87 \pm 1.39 1.07 \pm 0.14	1.19 \pm 0.18
CODA-P w/ InTAct	75.45 \pm 0.56 1.60 \pm 0.20	2.07 \pm 0.16	72.37 \pm 1.19 1.00 \pm 0.15	1.56 \pm 0.26

sifier head. InTAct counteracts this by jointly stabilizing the prompt-generation process and the classifier’s decision boundaries through its regularization losses ($\mathcal{L}_{\text{IntDrift}}$, $\mathcal{L}_{\text{Feat}}$, $\mathcal{L}_{\text{Align}}$). This coordinated stabilization reduces internal representation drift and prevents interference between evolving prompts and classification layers.

Overall, InTAct enhances the stability and consistency of CODA-Prompt, demonstrating that even advanced prompt-based approaches remain vulnerable to internal functional drift. By precisely regulating how and where a model’s representations evolve during continual learning, InTAct provides a principled and broadly applicable solution. Consistent findings are observed on the CIFAR-100 dataset, with additional results reported in SM 15.

4.3. Ablations

In Tab. 4, we report an ablation study analyzing the contribution of each component of InTAct. The experiment is conducted on the Split CIFAR-10 benchmark under the DIL setting. We report the AA averaged over five seeds for the full model and three seeds for each ablated variant, where a single component is removed at a time. The results show that excluding any term consistently degrades performance, demonstrating that all components of InTAct contribute in a complementary manner. The most significant drop occurs when \mathcal{L}_{Var} is removed, highlighting its importance in controlling the compactness of activation intervals. Without this term, the network’s representations are regularized over excessively large hypercubes, as described in Eq. (5), which weakens the effectiveness of our regularization. Moreover, we also investigate how different values of λ_{Var} affect AA, and present these results in SM 15.

We also observe that removing $\mathcal{L}_{\text{Align}}$ slightly decreases AA. This component prevents distant activation regions from merging into excessively large hypercubes when tasks are related (see Eq. (4)). Without it, two small but separate regions can combine into a much larger one, causing the model to regularize a substantial amount of irrelevant space

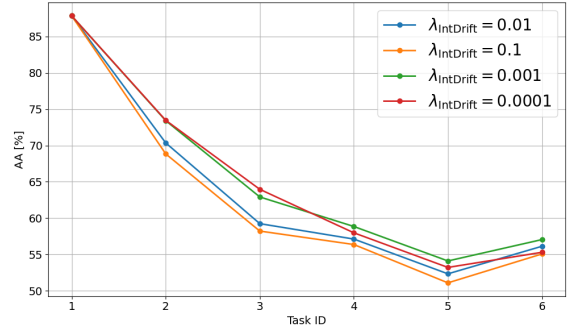
Tabela 4. Ablation study of different components in InTAct. The results are averaged over 3 random seeds for ablations.

Method	AA
InTAct	75.84 \pm 0.63
Ablate $\mathcal{L}_{\text{IntDrift}}$	71.63 \pm 1.34
Ablate \mathcal{L}_{Var}	67.43 \pm 1.32
Ablate $\mathcal{L}_{\text{Align}}$	74.37 \pm 0.46
Ablate $\mathcal{L}_{\text{Feat}}$	71.04 \pm 0.54

and limiting its ability to adapt effectively to new tasks.

4.4. Additional Experiments

Impact of $\lambda_{\text{IntDrift}}$ on AA in DIL setting. Fig. 5 illustrates how different values of $\lambda_{\text{IntDrift}}$ affect the stability–plasticity trade-off when InTAct is integrated into CODA-Prompt under the DIL scenario on the DomainNet dataset. While accuracy naturally decreases as new tasks are introduced, the strength of the regularization plays a key role in controlling this decline. A significant value ($\lambda_{\text{IntDrift}} = 0.1$) overconstrains the model, hindering adaptation to new tasks, whereas a very small value ($\lambda_{\text{IntDrift}} = 0.0001$) provides insufficient functional stability. The best performance is obtained for $\lambda_{\text{IntDrift}} = 0.001$, which yields the highest AA across tasks by achieving a balanced trade-off.



Rysunek 5. Impact of the $\lambda_{\text{IntDrift}}$ hyperparameter on AA for the CODA-Prompt method. Results are averaged over 2 random seeds.

These results confirm that our regularization does not overconstrain the model and effectively mediates the stability–plasticity balance. All other hyperparameters are fixed to the optimal configuration previously determined for CODA-Prompt on DomainNet under the DIL setting to isolate the effect of $\lambda_{\text{IntDrift}}$. For clarity, the reported values correspond to the scaled coefficient $\lambda'_{\text{IntDrift}} = \frac{\lambda_{\text{IntDrift}}}{C}$ where C is the total number of classes in the benchmark (here, $C = 345$).

5. Conclusions

We introduced InTAct, a continual learning method that preserves knowledge at the activation level. Rather than constraining parameters or replaying data, InTAct protects key activation ranges that encode past knowledge, keeping transformations consistent within these regions while allowing flexibility elsewhere. This prevents representation drift without reducing adaptability. InTAct is lightweight, model-agnostic, and integrates easily with existing architectures, improving stability and accuracy across benchmarks such as DomainNet and ImageNet-R, and narrowing the gap to the multi-task upper bound.

Limitations. Current activation regions are represented as hypercubes, which may not capture the full complexity of activation geometries. Future work will explore more flexible region representations and improved mechanisms for consolidating knowledge across tasks.

Literatura

- [1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget, 2018. 2, 1
- [2] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *Advances in neural information processing systems*, pages 15920–15930, 2020. 2, 1
- [3] Arslan Chaudhry, Marc’Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2018. 7
- [4] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc’Aurelio Ranzato. On tiny episodic memories in continual learning, 2019. arXiv preprint arXiv:1902.10486. 2, 1
- [5] Zhikang Chen, Abudukelimu Wuerkaixi, Sen Cui, Haoxuan Li, Ding Li, Jingfeng Zhang, Bo Han, Gang Niu, Houfang Liu, Yi Yang, Sifan Yang, Changshui Zhang, and Tianling Ren. Learning without isolation: Pathway protection for continual learning. In *International Conference on Machine Learning*, 2025. 2, 1
- [6] Germund Dahlquist and Åke Björck. *Numerical methods in scientific computing, volume I*. SIAM, 2008. 3, 2
- [7] MohammadReza Davari, Nader Asadi, Sudhir Mudur, Rahaf Aljundi, and Eugene Belilovsky. Probing representation forgetting in supervised and unsupervised continual learning, 2022. 5
- [8] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021. 2, 1
- [9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 6, 2, 4
- [10] Arthur Douillard and Timothée Lesort. Continuum: Simple management of complex continual learning scenarios, 2021. 4
- [11] Zhanxin Gao, Jun Cen, and Xiaobin Chang. Consistent prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28463–28473, 2024. 1, 3
- [12] Dipam Goswami, Yuyang Liu, Bartłomiej Twardowski, and Joost van de Weijer. Fecam: Exploiting the heterogeneity of class distributions in exemplar-free continual learning, 2024. 1
- [13] Dipam Goswami, Albin Soutif-Cormerais, Yuyang Liu, Sandesh Kamath, Bart Twardowski, and Joost van de Weijer. Resurrecting old classes with new data for exemplar-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28525–28534, 2024. 2
- [14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 6, 4
- [15] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. *ICCV*, 2021. 2, 4
- [16] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines, 2019. 4
- [17] Yusong Hu, Zichen Liang, Fei Yang, Qibin Hou, Xialei Liu, and Ming-Ming Cheng. Kac: Kolmogorov-arnold classifier for continual learning, 2025. 1
- [18] Paul Janson, Wenxuan Zhang, Rahaf Aljundi, and Mohamed Elhoseiny. A simple baseline that questions the use of pretrained-models in continual learning. In *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2022. 1
- [19] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust and Control*. Springer, London, 2001. 6, 7
- [20] Sanghwan Kim, Lorenzo Noci, Antonio Orvieto, and Thomas Hofmann. Achieving a better stability-plasticity trade-off via auxiliary networks in continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11930–11939, 2023. 2, 1
- [21] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. 1, 2

- [22] Yajing Kong, Liu Liu, Zhen Wang, and Dacheng Tao. Balancing stability and plasticity through advanced null space in continual learning. In *European Conference on Computer Vision*, pages 219–236, 2022. [2](#), [1](#)
- [23] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012. [4](#)
- [24] Patryk Krukowski, Anna Bielawska, Kamil Książek, Paweł Wawrzyński, Paweł Batorski, and Przemysław Spurek. Hint: Hypernetwork approach to training weight interval regions in continual learning. *Information Sciences*, 717:122261, 2025. [3](#), [2](#)
- [25] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021. [2](#)
- [26] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. [2](#), [1](#), [11](#)
- [27] Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23638–23647, 2024. [2](#), [1](#)
- [28] Xuan Liu and Xiaobin Chang. Lora subtraction for drift-resistant space in exemplar-free continual learning. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 15308–15318, 2025. [2](#)
- [29] David Lopez-Paz and Marc’Aurelio Ranzato. Gradient episodic memory for continual learning, 2022. [2](#), [1](#)
- [30] Tamasha Malepathirana, Damith Senanayake, and Saman Halgamuge. Napa-vq: Neighborhood-aware prototype augmentation with vector quantization for continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11674–11684, 2023. [1](#)
- [31] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. [2](#), [1](#)
- [32] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018. [2](#), [1](#)
- [33] Marc Masana, Xialei Liu, Bartłomiej Twardowski, Mikel Menta, Andrew D. Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5513–5533, 2022. [2](#), [1](#)
- [34] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. pages 109–165. Academic Press, 1989. [1](#)
- [35] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to interval analysis*. SIAM, 2009. [3](#), [2](#), [5](#), [7](#), [8](#)
- [36] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:475–480, 2019. [2](#), [1](#)
- [37] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1406–1415, 2019. [2](#), [4](#)
- [38] Grégoire Petit, Adrian Popescu, Hugo Schindler, David Picard, and Bertrand Delezoide. Fetrl: Feature translation for exemplar-free class-incremental learning. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3911–3920, 2023. [2](#)
- [39] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018. [2](#), [1](#)
- [40] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning, 2019. [11](#)
- [41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015. [6](#)
- [42] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016. arXiv preprint arXiv:1606.04671. [2](#), [1](#)
- [43] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2022. [2](#), [1](#)
- [44] Joan Serra, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557, 2018. [2](#), [1](#)
- [45] Wuxuan Shi and Mang Ye. Prototype reminiscence and augmented asymmetric knowledge aggregation for non-exemplar class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1772–1781, 2023. [1](#)
- [46] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in neural information processing systems*, 2017. [2](#), [1](#)
- [47] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning, 2023. [1](#), [3](#), [7](#), [10](#), [11](#), [12](#)
- [48] Shixiang Tang, Dapeng Chen, Jinguo Zhu, Shijie Yu, and Wanli Ouyang. Layerwise optimization by gradient decomposition for continual learning. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 9634–9643, 2021. [2](#), [1](#)
- [49] Marco Toldo and Mete Ozay. Bring evanescent representations to life in lifelong class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16732–16741, 2022. [1](#)
- [50] Edoardo Uretini and Antonio Carta. Online curvature-aware replay: Leveraging 2nd order information for online continual learning. In *International Conference on Machine Learning*, 2025. [2](#), [1](#)

- [51] Shipeng Wang, Xiaorong Li, Jian Sun, and Zongben Xu. Training networks in null space of feature covariance for continual learning. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 184–193, 2021. [2](#), [1](#)
- [52] Zhen Wang, Liu Liu, Yiqun Duan, Yajing Kong, and Dacheng Tao. Continual learning with lifelong vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 171–181, 2022. [2](#), [1](#)
- [53] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Dualprompt: Complementary prompting for rehearsal-free continual learning, 2022. [1](#), [3](#)
- [54] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning, 2022. [1](#), [3](#)
- [55] Maciej Wołczyk, Karol Piczak, Bartosz Wójcik, Lukasz Pustelnik, Paweł Morawiecki, Jacek Tabor, Tomasz Trzcinski, and Przemysław Spurek. Continual learning with guarantees via weight interval constraints. In *International Conference on Machine Learning*, pages 23897–23911. PMLR, 2022. [3](#), [2](#)
- [56] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Advances in Neural Information Processing Systems*, pages 15173–15184, 2020. [2](#), [1](#)
- [57] Chenshen Wu, Luis Herranz, Xialei Liu, Yaxing Wang, Joost van de Weijer, and Bogdan Raducanu. Memory replay gans: Learning to generate new categories without forgetting. In *Advances in Neural Information Processing Systems*, 2018. [2](#), [1](#)
- [58] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. [4](#)
- [59] Lu Yu, Bartłomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6982–6991, 2020. [1](#)
- [60] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence, 2017. [2](#), [1](#)
- [61] Da-Wei Zhou, Hai-Long Sun, Han-Jia Ye, and De-Chuan Zhan. Expandable subspace ensemble for pre-trained model-based class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23554–23564, 2024. [2](#)

InTact: Interval-based Task Activation Consolidation for Continual Learning

Supplementary Material

6. Extended Overview of Related Works

This section provides an extended literature review of continual learning methods. It first categorizes classical approaches into replay-based, regularization-based, and parameter-isolation techniques. Subsequently, the focus shifts to modern parameter-efficient methods like prompt-based continual learning (e.g., L2P, DualPrompt, CODA-Prompt) and contemporary strategies for managing feature drift in the representation space. Finally, we review recent applications of interval arithmetic in continual learning, contrasting these ideas with our proposed approach of regularizing functional transformations across model layers.

Continual Learning. Continual learning methods [8, 33, 36] can generally be divided into three categories: replay-based, regularization-based, and parameter-isolation-based. *Replay-based* methods employ memory or rehearsal mechanisms to recall past tasks during training, thereby maintaining low loss on those tasks. Two primary strategies are exemplar replay, which stores selected training samples [2–4, 27, 29, 39, 50], and generative replay, where models synthesize previous data using generative models [46, 57]. *Regularization-based* methods typically introduce a regularization term into the loss function to constrain parameter changes for previously learned tasks. This regularization may be defined by the current task data [20, 26] or previous tasks data [1, 5, 21, 60] variants. Recent methods limit weight updates to null space of previous data feature covariance [22, 48, 51, 52]. Liang and Li [27] leverage gradient information from old tasks to construct a subspace for LoRA’s dimensionality reduction matrix, thereby reducing interference between the current task and the previous ones. *Parameter-isolation* methods learn task-specific subnetworks within the model. Techniques such as Progressive Neural Networks [42, 43], Piggyback [32], PackNet [31], HAT [44] and SupSup [56] allocate and combine parameters for individual tasks. While effective in task-aware settings, these methods are based on assigning test samples to tasks, which may be problematic in some continual learning scenarios.

Prompt-Based Continual Learning. These methods are philosophically rooted in parameter-efficient learning, where the vast majority of the model (the backbone) remains unchanged to prevent catastrophic forgetting. The core mechanism involves introducing a small set of new, learnable parameters called *prompts* for each task. These prompts are typically vectors that are prepended to the input sequ-

ence embeddings or inserted into intermediate layers. This effectively steers the frozen model’s behavior towards the new task’s objective without overwriting knowledge from previous tasks.

A key innovation in this area is the concept of a *prompt pool*. Rather than learning a single, monolithic prompt for each task, methods like L2P [11] learn a collection of shared prompt vectors. During inference, the model uses a query-based mechanism (e.g., matching input features to prompt keys) to select a sparse combination of prompts from this pool that are most relevant to the current input instance. This approach not only adapts to the current task but also allows the model to potentially recognize and handle inputs from previous tasks by selecting the appropriate "old" prompts, enabling a form of rehearsal-free task routing.

Further refinements focus on the *structure* and *function* of the prompts themselves. For instance, DualPrompt [47] introduced the idea of learning two distinct sets of prompts: a *general* prompt shared across all tasks to capture common knowledge, and a set of *task-specific* prompts to capture task-unique features. CoDA-Prompt [53] builds on this by using a decomposed attention mechanism, allowing the model to explicitly query general versus task-specific knowledge encoded in the prompts, thereby improving modularity and reducing interference between task-specific instructions.

Despite their success in parameter-efficient, rehearsal-free learning, these methods face a fundamental limitation, particularly in domain-incremental learning (DIL): they struggle to adapt the classifier to distributional shifts between tasks. While the backbone features remain largely fixed, the learned prompts or adapters often fail to capture semantic drift in task labels or decision boundaries. Consequently, when a new task comes from a domain substantially different from previous ones, the classifier may misrepresent the task despite informative backbone features. This highlights the need for future approaches that strategically update classifier parameters or the backbone itself, or develop more flexible prompt-tuning strategies capable of handling evolving task distributions.

Addressing Feature Drift in Continual Learning. A critical challenge in continual learning is the feature drift of old classes when new tasks are learned without access to previous samples. Methods often define prototypes of classes [12, 18] in the feature space and prevent their drift. SDC [59] and [49] estimate and compensate for feature drift with current-task data following each training phase. NAPA-VQ [30] and Prototype Reminiscence [45] reshape old prototy-

pes through topological information with current-data samples. FeTrIL [38] introduces a feature translation strategy that aligns new and old class feature distributions. ADC [13] generates adversarial pseudo-exemplars of new classes to adjust prototypes of earlier classes. EASE [61] introduces a semantic-guided prototype complement strategy that reshapes prototypes of old classes in the feature space adjusted by the current task’s data. Liu and Chang [28] defines a drift-resistant space in which the model weights can be adjusted to the new task without interfering with old data features. In this work, we do not refer to the concept of prototype. Instead, we mitigate the feature drift by enforcing each layer to preserve its performed transformation on its subdomain defined by the previous tasks’ data.

Interval Arithmetic in Continual Learning. Interval arithmetic [6, 35] has been applied to continual learning in InterContiNet [55], where the key idea is to employ interval constraints on the weights associated with successive tasks. The intersection of these intervals defines the subset of weights that yield satisfactory performance across all tasks. HINT [24] employs intervals in the embedding space and leverages a hypernetwork to map them to the weight space of the target network. In contrast, we propose a novel approach, also grounded in interval arithmetic: each layer of the neural network is regularized to preserve its transformation within a subdomain delineated by a hyper-interval, which encapsulates data from previously learned tasks.

Parameter-Isolation Approaches. Another strategy is to prevent interference across tasks by isolating parameters. Progressive Neural Networks [43] expand the architecture with task-specific columns and lateral connections, reusing prior knowledge without overwriting it. While this eliminates forgetting by construction, network growth scales linearly with the number of tasks, making these approaches impractical for long task sequences or large-scale settings.

Regularization-Based Methods. Regularization-based approaches constrain weight updates to preserve information from earlier tasks. Elastic Weight Consolidation (EWC) [60] introduces a Fisher information-based penalty to protect important parameters, while Synaptic Intelligence (SI) [26] and Memory Aware Synapses (MAS) [21] estimate weight importance through gradient flow. Learning Without Forgetting (LwF) [1] further anchors predictions of past tasks using knowledge distillation. Despite their success, these methods focus primarily on the parameter space, leaving internal representations vulnerable to drift. As a result, preserving weight importance does not necessarily maintain the functional behavior of learned features.

Representation-Level Stability. A complementary direction emphasizes stabilizing learned representations rather than weights. Approaches inspired by interval analysis [35] and activation regularization aim to limit representation drift by constraining activations to remain within task-specific bounds. However, these ideas have seen limited adoption in continual learning due to the difficulty of summarizing activation distributions efficiently.

7. Preliminaries

This section presents the key technical concepts underlying prompt-based continual learning. We begin by explaining how parameter-efficient prefix tuning allows a frozen pre-trained feature extractor to be adapted without modifying its weights. We then provide an overview of three representative prompt-management frameworks, L2P, DualPrompt, and CODA-Prompt, which illustrate how modern methods organize and select prompts to support rehearsal-free learning while minimizing interference across tasks.

Prompt-Based Continual Learning. Recent work explores continual learning through prompts, leveraging the representational capacity of pretrained vision models. The core idea is to introduce a small set of learnable prompt tokens, which guide a frozen network backbone to adapt to new tasks, significantly reducing catastrophic forgetting without requiring rehearsal of old data. We now detail the mechanics and principal architectures of these methods.

Let f_θ denote a pretrained encoder (e.g., a Vision Transformer [9]) with frozen parameters θ . We denote the input token embeddings to the l -th layer as $h^{(l)} \in \mathbb{R}^{L \times D}$, where L is the sequence length (tokens) and D is the embedding dimension. In prompt-based continual learning, each task is associated with a set of learnable prompt vectors, $p \in \mathbb{R}^{L_p \times D}$, where L_p is the prompt length. These prompts serve as additional tokens that condition the model’s self-attention, allowing it to adapt to new tasks without modifying the shared backbone parameters θ .

A common and widely used implementation of prompt injection is prefix-tuning [25]. In a standard multi-head self-attention (MSA) layer, the input $h^{(l)}$ is linearly projected into Query (Q), Key (K), and Value (V) matrices using frozen projection matrices W_Q, W_K, W_V . Prefix-tuning introduces a layer-specific prompt $p^{(l)}$ composed of two sets of vectors, $p_K^{(l)}, p_V^{(l)} \in \mathbb{R}^{L_p \times D}$. These vectors are prepended (concatenated) to the K and V matrices, respectively, replacing the standard MSA operation with a prompted one:

$$\begin{aligned} Q &= h^{(l)} W_Q, \\ K' &= [p_K^{(l)}; h^{(l)} W_K], \\ V' &= [p_V^{(l)}; h^{(l)} W_V], \\ h^{(l+1)} &= \text{MSA}(Q, K', V'), \end{aligned} \tag{17}$$

Where $\text{MSA}(\cdot)$ denotes the standard multi-head self-attention operation, $[\cdot]$ is concatenation along the sequence dimension, and W_Q, W_K, W_V are the frozen projection matrices. K' and V' are the augmented Key and Value matrices incorporating the prompt components $p_K^{(l)}$ and $p_V^{(l)}$, where $[\cdot]$ denotes concatenation along the sequence length dimension. This design enables task-specific adaptation purely through the small set of learnable prompts $p^{(l)}$ while keeping the entire backbone f_θ frozen. Additionally, the model includes a trainable classifier head (a linear layer) whose weights are optimized alongside the prompts to map the prompt-adapted feature representations to the specific class logits.

Prompt Selection and Memory Structure. A central challenge is managing and selecting the correct prompts when the task identity is unknown at test time (task-agnostic continual learning). Learning to Prompt (L2P) [54] addresses this by storing prompts in a shared prompt pool (or memory) $\mathcal{P} = \{(k_i, p_i)\}_{i=1}^M$, where M is the pool size. Each prompt p_i is associated with a corresponding learnable key $k_i \in \mathbb{R}^D$.

During training on each task, a subset of the key-prompt pairs is optimized via backpropagation to capture task-relevant knowledge, while f_θ remains frozen. At inference time, L2P uses a two-stage mechanism to process an input x :

1. **Query Pass (Prompt Selection):** The input x is first passed through the frozen, unprompted encoder f_θ . The resulting output embedding of the $[\text{CLS}]$ token, $q(x) = f_\theta(x)_{[\text{CLS}]}$, is used as a query. This query is compared to all keys $k_i \in \mathcal{P}$ using a similarity function (e.g., cosine similarity) to select a subset containing the N most relevant prompts.
2. **Execution Pass (Task Processing):** The N selected prompts $\{p_j\}_{j \in S}$, with $S \subset \{1, \dots, M\}$, are first aggregated into a single final prompt, typically via averaging: $p_{\text{final}} = \frac{1}{|S|} \sum_{j \in S} p_j$, where $|S|$ denotes cardinality of S . The input x is then processed by the encoder f_θ a second time, where p_{final} is injected into the self-attention layers (e.g., using Eq. 17).

This mechanism allows the model to retrieve the most relevant knowledge from memory based on the input features. During training, only the N selected key-prompt pairs are updated, while the rest of the pool remains unchanged.

Optimization Objective. Since the prompt retrieval mechanism (e.g., Top-K selection) is typically discrete and non-differentiable, gradients from the primary classification objective cannot flow back to update the keys $\{k_i\}$. To address this, prompt-based methods employ a composite loss function. The learnable prompt vectors p and the classifier head are optimized via the standard task loss $\mathcal{L}_{\text{task}}$

(e.g., Cross-Entropy). Simultaneously, the keys k are optimized via an auxiliary surrogate loss \mathcal{L}_{aux} (e.g., a matching loss) that minimizes the distance between the query and the selected keys. The total objective is formulated as $\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{task}} + \lambda \mathcal{L}_{\text{aux}}$, ensuring that the keys learn to capture the input distribution while the prompts learn to minimize the prediction error.

Complementary and Decomposed Prompting. DualPrompt [53] extends L2P by dividing the prompt space into two functional parts: a shared **G-Prompt** (g), which captures task-invariant knowledge, and a task-specific **E-Prompt** (e_t), which captures task-unique features. DualPrompt inserts E-Prompts $e_t = \{e_t^{(l)}\}$ into a set of shallow layers (e.g., layers $[1, \dots, L_e]$) and the G-Prompt $g = \{g^{(l)}\}$ into deeper layers (e.g., layers $[L_e + 1, \dots, L_g]$). The forward pass is a sequential application of prompted blocks:

$$h^{(l+1)} = \begin{cases} \text{Block}(h^{(l)}, e_t^{(l)}) & \text{for } l \in [1, L_e] \\ \text{Block}(h^{(l)}, g^{(l)}) & \text{for } l \in [L_e + 1, L_g] \end{cases} \quad (18)$$

where $\text{Block}(h, p)$ denotes a Transformer block using prompt p in its MSA (via Eq. 17). Unlike L2P's shared pool, DualPrompt learns a discrete set of key-prompt pairs $\{(k_t, e_t)\}$ (one for each task t). At inference, the query $q(x)$ is used to find the best-matching key k_t via cosine similarity, and its corresponding E-Prompt e_t is selected.

CODA-Prompt [47] generalizes this mechanism by composing the final prompt from a weighted sum of M learnable prompt components $\mathcal{P} = \{P_m\}_{m=1}^M$. The query $q(x)$ computes a set of attention weights $\alpha(x) = \{\alpha_m(x)\}_{m=1}^M$ over the components, representing a continuous composition over the memory:

$$\alpha_m(x) = \frac{\exp(\langle q(x) \odot A_m, K_m \rangle / \tau)}{\sum_{j=1}^M \exp(\langle q(x) \odot A_j, K_j \rangle / \tau)}, \quad (19)$$

where $K_m, A_m \in \mathbb{R}^D$ are learnable keys and query-modulation vectors, \odot is the Hadamard product, and τ is a temperature parameter. The final prompt $p(x)$ is then composed as:

$$p(x) = \sum_{m=1}^M \alpha_m(x) P_m. \quad (20)$$

This continuous, differentiable composition increases flexibility, allows capacity to scale by adding new components when needed, and uses orthogonality regularization to reduce interference.

8. Implementation Details

This section provides all necessary implementation and technical details. We discuss the architectures utilized, the construction of the dataset tasks, and the GPU resources used for the computations.

Architectures. We perform experiments using three widely used neural architectures:

- **MLP**: A three-layer fully connected network with 400 hidden units per layer, used for low-dimensional datasets (Split MNIST and Split FMNIST). This serves as a lightweight baseline.
- **ResNet-18** [14]: A convolutional network with residual connections, used for medium-scale datasets such as Split CIFAR-10. We use it as a feature extractor and unfreeze only the final residual block to allow limited adaptation. As a head, we employ a lightweight MLP classifier consisting of a single hidden layer with 400 units.
- **ViT** [9]: A Vision Transformer that operates on patch tokens with global self-attention. It is used for larger and more diverse benchmarks (DomainNet and ImageNet-R), and also for Split CIFAR-100. In this setup, ViT acts as both the feature extractor and the query function in prompt-based frameworks, and is kept fully frozen during training.

The classifier head is configured according to the continual learning setting. In the CIL scenario, where each task introduces new, non-overlapping classes, the classifier head is expanded by adding output neurons for the new labels. In the DIL scenario, the label set is shared across tasks, so the classifier head remains fixed and only the input distribution changes.

For prompt-based methods under CIL, the CODA-Prompt formulation applies an output masking mechanism that restricts predictions to classes observed so far. To ensure fair comparison, we adopt the same masking for all prompt-based baselines and for InTact in CIL experiments, including Split CIFAR-10. We do not apply masking in the MLP experiments, as the model is intentionally capacity-limited and we aim to evaluate each method’s ability to mitigate forgetting without additional constraints.

Datasets. We evaluate InTact on six widely used continual learning benchmarks that vary in scale, visual complexity, and type of distribution shift. All datasets are normalized to the $[0, 1]$ range. Unless stated otherwise, we consider both the CIL and DIL scenarios.

- **Split MNIST** [16]: The MNIST dataset is partitioned into five sequential tasks, each forming a binary classification problem from two non-overlapping digit classes.
- **Split FMNIST** [58]: Constructed analogously to Split MNIST using Fashion-MNIST, providing higher visual diversity and texture across tasks.
- **Split CIFAR-10** [23]: CIFAR-10 is divided into five tasks, each containing two distinct object categories. This benchmark introduces natural image variability, increasing the difficulty of representation stability.
- **Split CIFAR-100** [23]: CIFAR-100 is split into ten tasks, each containing ten classes. This benchmark is evaluated

only under the CIL setting.

- **DomainNet** [37]: A large-scale multi-domain benchmark with six visually distinct domains (*clipart, painting, real, sketch, quickdraw, infograph*). In the DIL setting, each domain forms one task (6 tasks total). In the CIL setting, we split classes into 5 tasks, with all domains present in each task. This results in domain-mixed CIL, where semantic novelty, not visual style, drives task progression.
- **ImageNet-R** [15]: A distribution shift benchmark of 200 ImageNet classes represented as renditions (e.g., drawings, sculptures, cartoons). In the DIL setting, we construct sequential tasks by randomly partitioning the 200 classes into 15 or 5 partitions, while keeping the label space constant. In the CIL setting, we evaluate two configurations with 10 and 20 tasks, where each task contains a disjoint subset of classes, but all domains remain present in every task.

All datasets are normalized to the $[0, 1]$ pixel range. For ImageNet-R, images are resized such that the shorter side is 256px, then during training, randomly cropped to 224×224 with random horizontal flip, and normalized using the ImageNet mean and standard deviation. At evaluation, the random crop is replaced with a center crop of the same size.

Code and Reproducibility. For experiments involving prompt-based architectures, we build directly on the official implementation of CODA-Prompt, and we adapt the DomainNet DIL task handlers from the KAC GitHub repository. To ensure strict reproducibility and a fair comparison with reported baselines, we match the software environment used in CODA-Prompt, including the same `timm` version (0.4.12), which is essential to ensure identical pre-trained ViT weights. Our method InTact is integrated into this codebase without modifying the underlying prompt selection or training routines.

For all remaining experiments (i.e., non-prompt-based settings), we implement InTact along with other referenced methods within a custom continual learning framework, using the following package versions: `pyrootutils 1.0.1`, `hydra-core 1.3.2`, `omegaconf 2.3.0`, `torch 2.5.1`, `torchvision 0.20.1`, `lightning 2.5.0`, `lightning-fabric 2.5.0`, `wandb 0.19.1`, and `continuum` [10].

All experiments were conducted on NVIDIA A100 (80GB) and V100 (32GB) GPUs within a DGX cluster, as well as on standalone RTX 4090 machines. Full training pipelines, configuration files, and scripts for reproducing all reported results are available in our GitHub repository.

9. Selected Hyperparameters of InTact

This section summarizes the key hyperparameters and training procedures used for InTact across all benchmarks. For

transparency, we report both the selected values and the corresponding search ranges.

Optimization Procedure. For Split MNIST, Split FMNIST, and Split CIFAR-10 (under both CIL and DIL), we select hyperparameters using a Bayesian optimization framework. The following search ranges are used consistently:

- Batch size (grid): {512, 256, 128}.
- Number of epochs (grid): {15, 10, 5}.
- Learning rate: log-uniform in $[10^{-4}, 10^{-2}]$.
- λ_{Feat} : uniform in $[0, 1000]$.
- $\lambda_{\text{IntDrift}}$: uniform in $[0, 1000]$.
- λ_{Var} : uniform in $[0, 10]$.

We emphasize that the baseline results reported in Table 9 and Table 10 are also obtained using Bayesian search. For readability, the hyperparameter values reported in the tables are rounded; the exact values used in experiments are provided in the code. All training scripts and the final selected hyperparameters are included in the codebase.

For Split CIFAR-100, DomainNet, and ImageNet-R (prompt-based experiments), we adopt the hyperparameter configurations reported in CODA-Prompt wherever available. CODA-Prompt does not provide results for DomainNet under DIL, nor for ImageNet-R under DIL with 5- or 15-task settings. In these cases, we transfer the best configuration identified in the corresponding CIL setting for DomainNet, and for ImageNet-R we reuse the configuration tuned for the 5-task CIL scenario. Training schedules and learning-rate schedulers remain unchanged. To accommodate GPUs with lower memory capacity, we reduce the batch size from 128 to 64 in these experiments.

For InTact integrated into prompt-based architectures under the CIL setting, we additionally perform a grid search over:

- λ_{Var} : {0.001, 0.01, 0.1, 1.0},
- $\lambda_{\text{IntDrift}}$: {0.0001, 0.001, 0.01, 0.1},
- λ_{Feat} : {0.0001, 0.001, 0.01, 0.1}.

In the DIL setting, we use the same grid search, but additionally scale $\lambda_{\text{IntDrift}}$ by the total number of classes encountered throughout training. We found that the classifier head to which this regularization is applied is sensitive to the absolute magnitude of $\lambda_{\text{IntDrift}}$, making this scaling beneficial for stable optimization.

The configuration achieving the highest validation AA is used for all final results reported in the main paper.

Best Hyperparameters. The selected hyperparameters for InTact on Split MNIST, Split FMNIST, and Split CIFAR-10 are shown in Table 5. Table 6 reports the hyperparameters used when integrating InTact with prompt-based methods. Note that in the CIL setting, we set $\lambda_{\text{IntDrift}}$ to zero because the outputs corresponding to

previously seen classes are masked in the classifier at the top of the ViT architecture. In this case, the loss component defined in Eq. (10) does not apply.

10. Interval Arithmetic

Interval arithmetic, introduced in its modern form by Ramon E. Moore [35], provides a method for performing computations on ranges of real numbers rather than single point values. An interval x is a closed, bounded set of real numbers defined by its endpoints, $x = [x_l, x_u]$, such that $x_l \leq x_u$. This framework is essential for validated numerics, as it allows for the rigorous containment of numerical errors, including rounding errors and uncertainties in initial data.

10.1. Fundamental Operations

The fundamental principle of interval arithmetic is the *inclusion property*: the resulting interval of an operation must contain all possible results from applying the same operation to any real numbers within the operand intervals. Let $x = [x_l, x_u]$ and $y = [y_l, y_u]$ be two intervals. The elementary arithmetic operations are defined as follows:

Addition. The sum is obtained by adding the respective endpoints:

$$x + y = [x_l + y_l, x_u + y_u] \quad (21)$$

Subtraction. The difference is computed by cross-adding the endpoints:

$$x - y = [x_l - y_u, x_u - y_l] \quad (22)$$

Multiplication. The product is the interval spanning the minimum and maximum of the four products of the endpoints.

$$x \cdot y = [z_l, z_u], \quad (23)$$

where

$$z_l = \min(x_l y_l, x_l y_u, x_u y_l, x_u y_u), \quad (24)$$

$$z_u = \max(x_l y_l, x_l y_u, x_u y_l, x_u y_u). \quad (25)$$

Division. Division is defined as multiplication by the reciprocal of y , provided that $0 \notin y$.

$$\frac{x}{y} = x \cdot \left[\frac{1}{y_u}, \frac{1}{y_l} \right], \quad \text{if } 0 \notin [y_l, y_u]$$

Suppose the divisor interval y contains zero. In that case, the operation is typically undefined or results in extended intervals (e.g., a union of two intervals), depending on the specific framework being used [35].

Tabela 5. Selected hyperparameters used for InTAct across the Split MNIST, Split FMNIST, and Split CIFAR-10 benchmarks under both CIL and DIL settings.

Dataset	Scenario	Batch size	Epochs	Learning rate	λ_{Feat}	$\lambda_{\text{IntDrift}}$	λ_{Var}
Split MNIST	CIL	512	5	2×10^{-4}	779	6	4.2
	DIL	512	5	1×10^{-4}	602	634	4.0
Split FMNIST	CIL	128	10	4×10^{-4}	227	731	0.8
	DIL	512	5	1×10^{-4}	927	196	1.6
Split CIFAR-10	CIL	32	15	1×9^{-3}	391	792	0.2
	DIL	64	5	6×10^{-4}	239	96	0.7

Tabela 6. Selected regularization hyperparameters for InTAct integrated into prompt-based methods. Rows correspond to datasets and learning scenarios; columns list the regularization coefficients used for each base method.

Dataset	Scenario	L2P			DualPrompt			CODA-Prompt		
		λ_{Feat}	$\lambda_{\text{IntDrift}}$	λ_{Var}	λ_{Feat}	$\lambda_{\text{IntDrift}}$	λ_{Var}	λ_{Feat}	$\lambda_{\text{IntDrift}}$	λ_{Var}
Split CIFAR-100	CIL	0.0001	0.0	0.001	0.1	0.0	0.001	0.01	0.0	0.001
DomainNet	CIL	0.1	0.0	0.001	0.001	0.0	0.01	0.0001	0.0	1.0
DomainNet	DIL	0.1	$\frac{0.0001}{345}$	0.001	0.001	$\frac{0.001}{345}$	0.01	0.0001	$\frac{0.001}{345}$	1.0
ImageNet-R	CIL (10 tasks)	0.0001	0.0	0.01	0.001	0.0	0.001	0.1	0.0	0.1
ImageNet-R	CIL (20 tasks)	0.001	0.0	0.01	0.0001	0.0	0.001	0.1	0.0	0.1
ImageNet-R	DIL (5 tasks)	0.01	$\frac{0.1}{200}$	0.01	0.01	$\frac{0.0001}{200}$	0.1	0.001	$\frac{0.1}{200}$	0.1
ImageNet-R	DIL (15 tasks)	0.1	$\frac{0.01}{200}$	0.01	0.0001	$\frac{0.01}{200}$	0.1	0.1	$\frac{0.001}{200}$	0.1

10.2. Interval Matrix Multiplication

The operations of interval arithmetic extend naturally to linear algebra, enabling computations with interval matrices. An interval matrix A is a matrix whose elements are intervals. Given two compatible interval matrices, A and B , their product $C = AB$ is an interval matrix where each element c_{ij} the interval extension of the standard dot product defines:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}. \quad (26)$$

Here, each multiplication $a_{ik} b_{kj}$ is an interval multiplication, and the summation \sum represents a sequence of interval additions. This operation is fundamental for solving interval linear systems, which are crucial for analyzing the effects of bounded uncertainties in models. However, this definition is highly susceptible to the dependency problem; if the same interval variable appears in multiple entries of A or B , the resulting bounds on C can be much wider than the true range.

10.3. Interval Convolutions

Interval arithmetic can be extended to operations like convolutions, which are fundamental in areas such as signal processing and image analysis. By representing an input signal X and a filter kernel W as intervals, one can analyze systems where inputs or parameters are subject to bounded

uncertainties.

A key operation is the interval convolution. For a 2D input X , an interval kernel (filter) W , and an optional interval bias b , the resulting output feature map Y is computed as:

$$Y_{ij} = \left(\sum_{k,l} X_{i+k,j+l} \cdot W_{k,l} \right) + b. \quad (27)$$

This computation propagates the interval bounds through the filtering operation, producing a rigorous enclosure for the true output. This is particularly useful in robust state estimation and control, where system parameters or measurements are known to lie within certain bounds [19].

10.4. Key Properties and Considerations

While interval arithmetic provides rigorous bounds, it has two important characteristics.

First, to be implemented correctly on a computer, interval arithmetic must use *directed rounding* (or *outward rounding*). For any operation, the computed lower bound must be rounded down (towards $-\infty$) and the computed upper bound must be rounded up (towards $+\infty$). This practice, supported by standards like IEEE 754, ensures that the resulting floating-point interval strictly encloses the true (and often unknowable) real-number interval. It is critical to note, however, that standard computing environments and programming languages, such as Python or MATLAB, do

not satisfy this requirement by default for general floating-point operations. Specialized interval arithmetic libraries (e.g., `python-intervals`, `JuliaIntervals`) or tools that explicitly manage floating-point control registers are necessary to ensure correct outward rounding and, thus, the validity of the computed bounds.

Second, interval arithmetic is subject to the *dependency problem*. When a variable appears multiple times in an expression, interval arithmetic treats each occurrence independently, which can lead to a significant overestimation of the true range. A classic example is computing $x - x$ with $x = [0, 1]$. The result is:

$$[0, 1] - [0, 1] = [0 - 1, 1 - 0] = [-1, 1]$$

This is a valid enclosure, but the true range for the function $f(x) = x - x$ over the interval $[0, 1]$ is simply $\{0\}$. This overestimation, also known as the *wrapping effect*, is a central challenge in interval-based methods, as seen in both matrix multiplication and complex dynamic systems [19, 35].

11. Metrics

We evaluate continual learning performance using two standard metrics: the Average Accuracy (AA) and the Average Forgetting (AF). These metrics assess how well the model retains knowledge across tasks and how much it forgets after learning new ones.

Let there be N tasks in sequence. Let $R_{i,j}$ denote the test accuracy on task i after the model has been trained on task j . In particular, $R_{i,N}$ represents the final accuracy on task i after training on all N tasks.

Average Accuracy (AA). The AA measures the mean performance over all tasks at the end of training:

$$AA = \frac{1}{N} \sum_{i=1}^N R_{i,N}. \quad (28)$$

A higher A_N indicates better overall continual learning performance and knowledge retention across all tasks.

Average Forgetting (AF). The standard definition of AF, as used for example in [3], measures for each task the gap between its best performance and its final performance:

$$\tilde{AF} = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(\max_{j \in \{i, \dots, N\}} R_{i,j} - R_{i,N} \right), \quad (29)$$

where $R_{i,j}$ denotes the accuracy on task i after training up to task j .

However, following the evaluation protocol used in the CODA-Prompt paper [47], and in order to ensure comparability with prompt-based baselines, we adopt the forgetting formulation implemented in their official codebase. For

each task t and each earlier task $i < t$, forgetting is accumulated as the incremental reduction in performance from step $t-1$ to t , averaged over all earlier tasks:

$$AF = \frac{1}{N-1} \sum_{t=2}^N \left(\frac{1}{t-1} \sum_{i=1}^{t-1} (R_{i,t-1} - R_{i,t}) \right). \quad (30)$$

Here, $R_{i,t}$ denotes the accuracy on task i after completing training on task t . A lower value of F_N indicates better retention of previously learned knowledge.

This corresponds exactly to the computation used in the CODA-Prompt evaluation script, ensuring direct comparability across all reported results.

12. Detailed Derivation of Internal Representation Drift Loss

Here, we provide the complete derivation for the $\mathcal{L}_{\text{IntDrift}}$ regularization term introduced in the main paper. Our objective is to ensure that the *preactivations* of the l -th layer, $l \in \mathcal{S}_H$, remain unchanged for all inputs x belonging to the hypercube $\mathcal{H}_{l-1}^{(t-1)}$. Formally, consider the affine layer

$$f(x; \theta_l) = W_l x + b_l. \quad (31)$$

After applying the parameter update $(\Delta W_l, \Delta b_l)$, the new preactivation becomes

$$f(x; \theta_l + \Delta \theta_l) = (W_l + \Delta W_l)x + (b_l + \Delta b_l). \quad (32)$$

To ensure that the preactivations of the layer remain unchanged for every $x \in \mathcal{H}_{l-1}^{(t-1)}$, we require that

$$f(x; \theta_l + \Delta \theta_l) - f(x; \theta_l) = 0, \quad \forall x \in \mathcal{H}_{l-1}^{(t-1)}. \quad (33)$$

Substituting the expressions above yields the condition

$$\Delta W_l x + \Delta b_l = 0, \quad \forall x \in \mathcal{H}_{l-1}^{(t-1)}. \quad (34)$$

Using interval arithmetic, we extend this constraint over the full hypercube $\mathcal{H}_{l-1}^{(t-1)} = [\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}]$. The interval version of the preactivation-stability condition becomes

$$\Delta W_l [\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}] + \Delta b_l = [0, 0]. \quad (35)$$

This condition must hold for each output coordinate. Let $w_{l,i}^\top$ denote the i -th row of the weight update matrix ΔW_l . We therefore compute the interval bounds of the linear mapping $w_{l,i}^\top \mathcal{H}_{l-1}^{(t-1)}$. For each coordinate i , interval arithmetic yields:

$$w_{l,i}^\top \mathcal{H}_{l-1}^{(t-1)} = \left[\sum_j \min\{w_{l,ij} \underline{x}_{l-1,j}^{(t-1)}, w_{l,ij} \bar{x}_{l-1,j}^{(t-1)}\}, \right. \quad (36)$$

$$\left. \sum_j \max\{w_{l,ij} \underline{x}_{l-1,j}^{(t-1)}, w_{l,ij} \bar{x}_{l-1,j}^{(t-1)}\} \right]. \quad (37)$$

To obtain a differentiable form of the bounds, we decompose $w_{l,i}^\top$ into its positive and negative parts:

$$(w_{l,i}^\top)^+ = \max(0, w_{l,i}^\top), \quad (38)$$

$$(w_{l,i}^\top)^- = \max(0, -w_{l,i}^\top), \quad (39)$$

so that $w_{l,i}^\top = (w_{l,i}^\top)^+ - (w_{l,i}^\top)^-$. The interval product $w_{l,i}^\top \mathcal{H}_{l-1}^{(t-1)}$ is then computed as [35]:

$$w_{l,i}^\top \mathcal{H}_{l-1}^{(t-1)} = \left((w_{l,i}^\top)^+ - (w_{l,i}^\top)^- \right) [\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}] \quad (40)$$

$$= (w_{l,i}^\top)^+ [\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}] \quad (41)$$

$$- (w_{l,i}^\top)^- [\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}] \quad (42)$$

$$= [(w_{l,i}^\top)^+ \underline{x}_{l-1}^{(t-1)}, (w_{l,i}^\top)^+ \bar{x}_{l-1}^{(t-1)}] \quad (43)$$

$$- [(w_{l,i}^\top)^- \underline{x}_{l-1}^{(t-1)}, (w_{l,i}^\top)^- \bar{x}_{l-1}^{(t-1)}]. \quad (44)$$

Using interval subtraction $[a, b] - [c, d] = [a-d, b-c]$, we obtain the final hypercube $[\tilde{x}_{l,i}, \tilde{\bar{x}}_{l,i}]$:

$$\tilde{x}_{l,i} = (w_{l,i}^\top)^+ \underline{x}_{l-1}^{(t-1)} - (w_{l,i}^\top)^- \bar{x}_{l-1}^{(t-1)}, \quad (45)$$

$$\tilde{\bar{x}}_{l,i} = (w_{l,i}^\top)^+ \bar{x}_{l-1}^{(t-1)} - (w_{l,i}^\top)^- \underline{x}_{l-1}^{(t-1)}. \quad (46)$$

Adding the bias update $\Delta b_{l,i}$, the constraint from Eq. (35) becomes:

$$[\tilde{x}_{l,i} + \Delta b_{l,i}, \tilde{\bar{x}}_{l,i} + \Delta b_{l,i}] = [0, 0]. \quad (47)$$

To enforce this condition during training, our regularizer $\mathcal{L}_{\text{IntDrift}}$ penalizes the squared L_2 norm of these hypercube endpoint bounds, driving both toward zero, which yields Equation (10).

13. Internal Representation Drift Loss for Other Layers

This section presents the full derivation of the interval representation drift loss for convolutional and batch normalization layers.

Convolutional Layer. Here, we adapt the $\mathcal{L}_{\text{IntDrift}}$ derivation from Appendix 12 for a 2D convolutional layer. All notation (such as l , t , and \mathcal{H}) follows the definitions in the previous section.

A convolutional layer $f(x; \theta_l) = \text{Conv2d}(x, W_l) + b_l$ is a linear operator that applies a shared filter W_l across all spatial locations of the input x . A key insight is that this operation is a form of affine transformation (like a fully-connected layer) applied to local patches of the input.

Let ΔW_l and Δb_l be the updates to the layer's parameters. Our stability objective is to ensure that the preactivation change is zero for any input x within the prior task's

hypercube $\mathcal{H}_{l-1}^{(t-1)}$:

$$\text{Conv2d}(x, \Delta W_l) + \Delta b_l = 0, \quad \forall x \in \mathcal{H}_{l-1}^{(t-1)}. \quad (48)$$

We can analyze this by focusing on a single preactivation y_i in the i -th output channel. This value is computed by a dot product between the i -th filter of the kernel update, $\Delta w_{l,i}$, and the corresponding input patch x_p :

$$y_i = \Delta w_{l,i}^\top x_p + \Delta b_{l,i}. \quad (49)$$

Here, $\Delta w_{l,i}$ and x_p are the flattened vector representations of the i -th filter and the input patch, respectively. This formulation is mathematically identical to the affine layer case presented in Appendix 12. The input x_p is drawn from a hypercube $\mathcal{H}_{\text{patch}}$ defined by the interval bounds $[\underline{x}_{l-1}^{(t-1)}, \bar{x}_{l-1}^{(t-1)}]$ of the corresponding input elements.

Therefore, we can directly apply the same interval arithmetic derivation. The interval bounds on the output drift for the i -th channel, $[\tilde{x}_{l,i}, \tilde{\bar{x}}_{l,i}]$, are computed analogously. Let $\Delta w_{l,i}$ be the flattened i -th filter update, and let \underline{x}_p and \bar{x}_p be the corresponding flattened lower and upper bounds of the input patch. The bounds are:

$$\tilde{x}_{l,i} = (\Delta w_{l,i}^\top)^+ \underline{x}_p - (\Delta w_{l,i}^\top)^- \bar{x}_p, \quad (50)$$

$$\tilde{\bar{x}}_{l,i} = (\Delta w_{l,i}^\top)^+ \bar{x}_p - (\Delta w_{l,i}^\top)^- \underline{x}_p. \quad (51)$$

Note that $(\Delta w_{l,i}^\top)^+$ and $(\Delta w_{l,i}^\top)^-$ are the positive and negative parts of the filter update, and the products are element-wise dot products. The constraint from Eq. (35) becomes:

$$[\tilde{x}_{l,i} + \Delta b_{l,i}, \tilde{\bar{x}}_{l,i} + \Delta b_{l,i}] = [0, 0]. \quad (52)$$

Our regularizer $\mathcal{L}_{\text{IntDrift}}$ penalizes the squared L_2 norm of these hypercube endpoints, just as in the affine case, driving the representation drift within the defined input domain to zero.

BatchNorm Layer. The same principle applies to Batch-Norm (BN) layers. In inference, a BN layer computes an affine transformation:

$$f(x; \theta_l) = \gamma_l \left(\frac{x - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \right) + \beta_l, \quad (53)$$

where the learnable parameters are $\theta_l = (\gamma_l, \beta_l)$, and the fixed (non-learnable) parameters are the running statistics μ_l and σ_l^2 . We can rewrite this as a standard affine layer $f(x) = W_l^{\text{eff}} x + b_l^{\text{eff}}$, where:

$$W_l^{\text{eff}} = \frac{\gamma_l}{\sqrt{\sigma_l^2 + \epsilon}} \quad (54)$$

$$b_l^{\text{eff}} = \beta_l - \frac{\gamma_l \mu_l}{\sqrt{\sigma_l^2 + \epsilon}}. \quad (55)$$

When the parameters are updated by $(\Delta\gamma_l, \Delta\beta_l)$, the *drift* in the output, $\Delta f(x)$, is:

$$\Delta f(x) = f(x; \theta_l + \Delta\theta_l) - f(x; \theta_l) \quad (56)$$

$$= \left(\frac{\Delta\gamma_l}{\sqrt{\sigma_l^2 + \epsilon}} \right) x + \left(\Delta\beta_l - \frac{\Delta\gamma_l \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \right) \quad (57)$$

$$= \Delta W_l^{\text{eff}} x + \Delta b_l^{\text{eff}}. \quad (58)$$

This drift $\Delta f(x)$ is itself an affine transformation of x . We want to enforce $\Delta f(x) = 0$ for all $x \in \mathcal{H}_{l-1}^{(t-1)}$. This is the exact same problem as in Appendix 12.

Therefore, we can compute $\mathcal{L}_{\text{IntDrift}}$ for a BN layer by applying the derivation from Eq. (45) and Eq. (46), simply by replacing the weight update ΔW_l with the effective weight update ΔW_l^{eff} and the bias update Δb_l with the effective bias update Δb_l^{eff} .

14. Training Algorithm

The complete training procedure for InTact is provided in Algorithm 1. To make this procedure applicable to a wide range of continual learning setups, whether using standard architectures or prompt-based extensions, we adopt a unified formulation for both input handling and parameter optimization.

Unified Notation. Let Ψ denote the set of **all learnable parameters** in the current training configuration. This set may include model parameters, prompt parameters, or any additional task-specific components that are updated during training. Before describing the two possible settings, we introduce the procedure **QUERYPROMPTS**: this procedure is responsible for processing an input sample x and returning (i) a possibly modified input x_p , where prompt or context tokens may be injected, and (ii) an auxiliary loss term \mathcal{L}_{aux} that arises when prompt mechanisms impose additional learning objectives. Using this unified interface for input processing allows Algorithm 1 to operate consistently across different architectures and training strategies.

- **When prompt-based methods are not applied:** In this case, no prompt parameters are present ($\phi = \emptyset$), and the only learnable parameters are those inherent to the model, denoted by θ . Thus, the optimization target reduces to $\Psi = \theta$. Since no prompt mechanism is used, **QUERYPROMPTS** becomes an identity operation: it returns the original input ($x_p = x$) and contributes no auxiliary loss ($\mathcal{L}_{\text{aux}} = 0$).
- **When prompt-based methods are applied:** Here, the system includes a set of learnable prompt parameters ϕ , and may also include other components that remain trainable, depending on the chosen configuration. To describe this generally, we write $\Psi \subseteq \{\theta, \phi\}$. In this setting, **QUERYPROMPTS** uses the input x to retrieve prompt

or context tokens, producing a modified input $x_p \neq x$ and, when applicable, generating a prompt-related auxiliary loss \mathcal{L}_{aux} .

This unified formulation ensures that Algorithm 1 handles input transformation, gradient propagation, and all associated loss terms in a consistent manner across diverse continual learning configurations.

Algorithm 1 InTact Training Algorithm

Require: Number of tasks T ; Sequential tasks $\{\mathcal{D}_t\}_{t=1}^T$; Neural network f with parameters θ ; Optional prompt module parameters ϕ .
Require: Set of layer indices $\mathcal{S}_{\mathcal{H}}$ for which activation hypercubes are computed.
Require: Hyperparameters: λ_{Var} , $\lambda_{\text{IntDrift}}$, λ_{Feat} , λ_{Prompt} , α ; Training epochs E ; Learning rate η .
Ensure: Optimized parameters Ψ .

- 1: **Initialize:** Parameters θ, ϕ ; Hypercubes $\mathcal{H}_l^{(0)} \leftarrow \emptyset$ for all $l \in \mathcal{S}_{\mathcal{H}}$.
- 2: **Define Active Parameters:** $\Psi \subseteq \{\theta, \phi\}$.
- 3: **for** task $t = 1$ to T **do**
- 4: **for** epoch $e = 1$ to E **do**
- 5: **for** batch $(x, y) \sim \mathcal{D}_t$ **do**
- 6: {**Step 1: Prompt Query & Pre-processing**}
- 7: $x_p, \mathcal{L}_{\text{aux}} \leftarrow \text{QUERYPROMPTS}(x)$
- 8: {**Step 2: Forward Pass**}
- 9: $\hat{y} \leftarrow f(x_p; \theta)$
- 10: {**Step 3: Loss Calculation**}
- 11: $\mathcal{L}_{\text{Total}} \leftarrow \mathcal{L}_{\text{CE}} + \lambda_{\text{Prompt}} \mathcal{L}_{\text{aux}}$
- 12: Calculate variance loss \mathcal{L}_{Var} (Eq. (13))
- 13: $\mathcal{L}_{\text{Total}} \leftarrow \mathcal{L}_{\text{Total}} + \lambda_{\text{Var}} \mathcal{L}_{\text{Var}}$
- 14: **if** $t \geq 2$ **then**
- 15: Calculate feature distillation loss $\mathcal{L}_{\text{Feat}}$ (Eq. (15))
- 16: Calculate internal representation drift loss (Eq. (10))
- 17: Calculate inter-task alignment loss $\mathcal{L}_{\text{Align}}$ (Eq. 14)
- 18: $\mathcal{L}_{\text{Total}} \leftarrow \mathcal{L}_{\text{Total}} + \lambda_{\text{Feat}} \mathcal{L}_{\text{Feat}} + \lambda_{\text{IntDrift}} \mathcal{L}_{\text{IntDrift}} + \mathcal{L}_{\text{Align}}$
- 19: **end if**
- 20: {**Step 4: Optimization**}
- 21: Update $\Psi \leftarrow \Psi - \eta \nabla_{\Psi} \mathcal{L}_{\text{Total}}$
- 22: **end for**
- 23: **end for**
- 24: {**Step 5: Update Activation Hypercubes**}
- 25: Collect activations $A_l = \{h_l(x_p; \theta) \mid x \in \mathcal{D}_t\}$ for all $l \in \mathcal{S}_{\mathcal{H}}$
- 26: **for** each layer index $l \in \mathcal{S}_{\mathcal{H}}$ **do**
- 27: $\underline{v} \leftarrow \text{percentile}_{\alpha}(A_l)$; $\bar{v} \leftarrow \text{percentile}_{100-\alpha}(A_l)$
- 28: **if** $t = 1$ **then**
- 29: $\mathcal{H}_l^{(t)} \leftarrow [\underline{v}, \bar{v}]$
- 30: **else**
- 31: Let $\mathcal{H}_l^{(t-1)} = [\underline{h}, \bar{h}]$
- 32: $\mathcal{H}_l^{(t)} \leftarrow [\min(\underline{h}, \underline{v}), \max(\bar{h}, \bar{v})]$
- 33: **end if**
- 34: **end for**
- 35: **end for**

15. Additional Results

In this section, we report additional experiments and analysis that further demonstrate the effectiveness of InTact.

Results on CIFAR-100 under the CIL scenario. Tab. 7 shows results on the 10-task CIFAR-100. Across all methods, InTact maintains or improves accuracy while sub-

stantially reducing variance, confirming its stabilizing effect. For L2P and DualPrompt, InTact matches baseline accuracy but halves standard deviation, preserving functional stability without limiting adaptability. The largest gain occurs for CODA-Prompt, where AA rises from 86.25% to 87.31% and variance drops sharply, approaching the upper bound.

Tabela 7. Results on the 10-task (10 classes each) CIFAR-100 benchmark (CIL setting). We report the mean and standard deviation of AA and AF averaged over 5 random seeds.

Method	AA (\uparrow)	AF (\downarrow)
Upper-Bound	89.30	–
L2P	82.50 \pm 1.10	1.75 \pm 0.42
w/ InTact	82.40 \pm 0.30	1.75 \pm 0.18
DualPrompt	83.05 \pm 1.16	1.72 \pm 0.40
w/ InTact	82.71 \pm 0.51	1.66 \pm 0.33
CODA-P	86.25 \pm 0.74	1.67 \pm 0.26
w/ InTact	87.31 \pm 0.17	1.77 \pm 0.2

Results on DomainNet under the CIL scenario. Table 8 summarizes performance on the 5-task DomainNet benchmark (CIL). Adding InTact to L2P and DualPrompt improves AA while maintaining competitive AF. Notably, InTact reduces variance across random seeds, indicating increased training stability on this challenging multi-domain setting.

Tabela 8. Results on the 5-task (69 classes each) DomainNet benchmark (CIL setting). We report the mean and standard deviation of AA and AF calculated over 3 random seeds. Results from baselines are obtained from the paper [47].

Method	AA (\uparrow)	AF (\downarrow)
Upper-Bound	79.65	–
L2P	69.58 \pm 0.39	2.25 \pm 0.08
w/ InTact	70.02 \pm 0.17	3.13 \pm 0.34
DualPrompt	70.73 \pm 0.49	2.03 \pm 0.22
w/ InTact	71.33 \pm 0.13	3.87 \pm 0.59
CODA-P	73.24 \pm 0.59	3.46 \pm 0.09
w/ InTact	72.12 \pm 0.10	3.59 \pm 0.13

Results for Regularization-based Methods (CIL). InTact achieves the best performance among all regularization-based methods on the CIL benchmarks, consistently improving AA across tasks. The detailed results are shown in Table 9 for Split MNIST, Split FMNIST, and Split CIFAR-10. In particular, InTact outperforms

the strongest baseline (LwF) by more than six percentage points on Split MNIST and Split FMNIST, demonstrating its ability to mitigate catastrophic forgetting and stabilize internal representations across sequential tasks. Note that these gains are achieved without any task-specific architectural modifications or rehearsal buffers, highlighting the robustness and generality of the proposed interval-based regularization.

Tabela 9. Comparison of regularization-based continual learning baselines and InTact across multiple datasets (CIL setting). Results report the AA metric values and are averaged over 5 random seeds.

Method	Split MNIST	Split FMNIST	Split CIFAR-10
EWC	20.57 \pm 0.78	19.92 \pm 0.02	30.43 \pm 1.69
LwF	28.63 \pm 0.61	25.92 \pm 0.84	24.26 \pm 0.75
MAS	17.91 \pm 2.93	18.74 \pm 3.40	21.75 \pm 0.67
SI	18.40 \pm 0.33	17.30 \pm 0.52	20.18 \pm 0.27
InTact	35.27 \pm 2.27	34.25 \pm 2.91	31.28 \pm 0.77

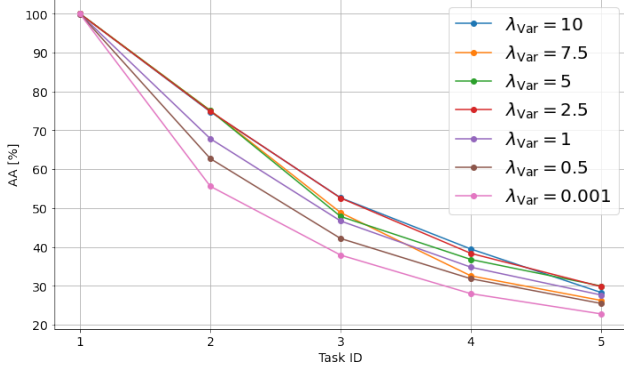
Results for Regularization-Based Methods (DIL). InTact achieves also the strongest performance among regularization-based methods on the DIL benchmarks. The detailed results are reported in Table 10 for Split MNIST, Split FMNIST, and Split CIFAR-10. On Split MNIST, InTact clearly outperforms EWC, MAS, and SI, achieving the highest AA. For Split FMNIST, all methods perform competitively, with InTact and EWC obtaining similar results. On Split CIFAR-10, InTact matches the best-performing baseline (LwF), confirming that the proposed regularization is effective even in more complex visual tasks.

Tabela 10. Comparison of regularization-based continual learning baselines and InTact across multiple datasets (DIL setting). Results report the AA metric values and are averaged over 5 random seeds.

Method	Split MNIST	Split FMNIST	Split CIFAR-10
EWC	78.90 \pm 2.28	92.30 \pm 0.80	68.93 \pm 1.13
LwF	75.54 \pm 0.37	84.74 \pm 0.76	76.45 \pm 0.63
MAS	77.59 \pm 2.84	91.16 \pm 2.56	69.35 \pm 3.40
SI	69.84 \pm 10.15	85.03 \pm 3.98	67.66 \pm 0.38
InTact	82.51 \pm 0.83	92.57 \pm 0.63	75.84 \pm 0.63

These gains are achieved without task-specific knowledge distillation or memory buffers, highlighting the general applicability of InTact in DIL scenarios.

Impact of λ_{var} on AA. In Fig. 6, we illustrate the effect of varying λ_{var} on AA for the Split MNIST dataset under the CIL setting.



Rysunek 6. Impact of the λ_{var} hyperparameter on AA for the Split MNIST dataset under the CIL scenario. Results are averaged over 5 seeds.

We observe that increasing the value of λ_{var} generally leads to improved AA. This is because stronger representation variance regularization (Eq. (13)) encourages the hidden representations to become more compact and structured. In turn, this results in activation hypercubes of smaller volume. Since our regularization mechanism constrains in-layer transformations within these hypercubes, operating on a smaller and more coherent region of the activation space yields a more effective and stable control signal. All other hyperparameters are fixed to their respective optimal values to isolate the effect of λ_{var} .

16. Extended Table Results in CIL Scenario

In this section, we present extended results on the CIFAR-100, DomainNet, and ImageNet-R benchmarks (Tables 11, 12, and 13). Overall, InTact consistently enhances the performance of existing prompt-based methods (L2P, DualPrompt, CODA-P), improving AA while achieving competitive or lower AF. Beyond prompt-based approaches, InTact also surpasses non-prompt continual learning baselines such as ER [40], sequential fine-tuning (FT), its improved variant FT++, which uses the same classifier implementation as prompt-based methods in the CIL scenario, and LwF [26] on most benchmarks. The notable exception is DomainNet, where LwF surprisingly achieves the highest AA. These results demonstrate that InTact effectively boosts prompt-based methods and remains competitive against traditional continual learning strategies. In all tables, the best results in each column are highlighted in bold.

17. Additional Visualization

Here, we present additional visualizations of representation drift in InTact and LwF, and provide analysis based on the Split MNIST dataset in the DIL scenario.

Tabela 11. Extended results on the 10-task (10 classes each) CIFAR-100 benchmark (CIL setting). We report the mean and standard deviation of AA and AF averaged over 5 random seeds. Best results in each column are highlighted in bold. Results from baselines are obtained from [47].

Method	AA (\uparrow)	AF (\downarrow)
Upper-Bound	89.30	—
ER (5000)	76.20 \pm 1.04	8.50 \pm 0.37
FT	9.92 \pm 0.27	29.21 \pm 0.18
FT++	49.91 \pm 0.42	12.30 \pm 0.23
LwF.MC	64.83 \pm 1.03	5.27 \pm 0.39
L2P	82.50 \pm 1.10	1.75 \pm 0.42
w/ InTact	82.40 \pm 0.30	1.75 \pm 0.18
DualPrompt	83.05 \pm 1.16	1.72 \pm 0.40
w/ InTact	82.71 \pm 0.51	1.66 \pm 0.33
CODA-P	86.25 \pm 0.74	1.67 \pm 0.26
w/ InTact	87.31 \pm 0.17	1.77 \pm 0.20

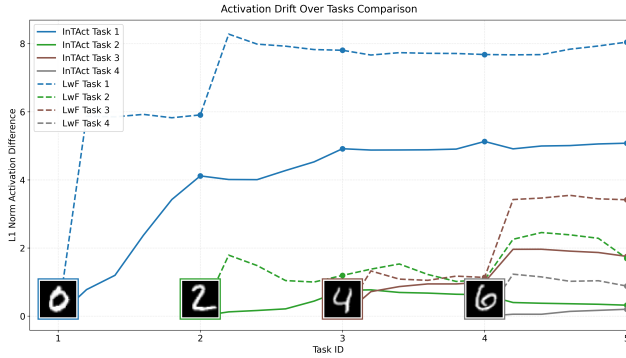
Tabela 12. Extended results on the 5-task (69 classes each) DomainNet benchmark (CIL setting). We report the mean and standard deviation of AA and AF calculated over 3 random seeds. Best results in each column are highlighted in bold. Results from baselines are obtained from [47].

Method	AA (\uparrow)	AF (\downarrow)
Upper-Bound	79.65	—
ER (5000)	58.32 \pm 0.47	26.25 \pm 0.24
FT	18.00 \pm 0.26	43.55 \pm 0.27
FT++	39.28 \pm 0.21	44.39 \pm 0.31
LwF.MC	74.78 \pm 0.43	5.01 \pm 0.14
L2P	69.58 \pm 0.39	2.25 \pm 0.08
w/ InTact	70.02 \pm 0.17	3.13 \pm 0.34
DualPrompt	70.73 \pm 0.49	2.03 \pm 0.22
w/ InTact	71.33 \pm 0.13	3.87 \pm 0.59
CODA-P	73.24 \pm 0.59	3.46 \pm 0.09
w/ InTact	72.12 \pm 0.10	3.59 \pm 0.13

Analysis of Internal Representation Drift. We provide a qualitative analysis of internal representation stability in Figure 7. This experiment, conducted on SplitMNIST, visualizes the *internal representation drift* experienced by data from past tasks as new tasks are learned. The plot tracks the L_1 norm of the activation difference for specific reference images (e.g., the image "0" from Task 1, "2" from Task 2, etc.) compared to their original activations recorded after their respective tasks were first learned.

Tabela 13. Extended results on the ImageNet-R dataset (CIL setting). We report the mean and standard deviation of AA and AF averaged over 5 random seeds. Results include 10- and 20-tasks baselines. Best results in each column are highlighted in bold. Results from baselines are obtained from [47].

Method	10 steps		20 steps	
	AA (\uparrow)	AF (\downarrow)	AA (\uparrow)	AF (\downarrow)
Upper-Bound	77.13	—	77.13	-
ER (5000)	64.43 ± 1.16	10.30 ± 0.05	52.43 ± 0.87	7.70 ± 0.13
FT	10.12 ± 0.51	25.69 ± 0.23	4.75 ± 0.40	16.34 ± 0.19
FT++	48.93 ± 1.15	9.81 ± 0.31	35.98 ± 1.38	6.63 ± 0.11
LwF.MC	66.73 ± 1.25	3.52 ± 0.39	54.05 ± 2.66	2.86 ± 0.26
L2P	69.29 ± 0.73	2.03 ± 0.19	65.89 ± 1.30	1.24 ± 0.14
w/ InTAct	69.44 ± 0.45	2.89 ± 0.46	66.18 ± 0.36	1.30 ± 0.13
DualPrompt	71.32 ± 0.62	1.71 ± 0.24	67.87 ± 1.39	1.07 ± 0.14
w/ InTAct	70.98 ± 0.60	1.68 ± 0.07	67.89 ± 0.66	1.19 ± 0.18
CODA-P	75.45 ± 0.56	1.64 ± 0.10	72.37 ± 1.19	0.96 ± 0.15
w/ InTAct	76.40 ± 0.16	2.07 ± 0.16	73.30 ± 0.41	1.56 ± 0.26



Rysunek 7. Activation drift across tasks on SplitMNIST (DIL scenario). For each task, we record hidden activations of a reference image and then track the normalized absolute change of these activations as new tasks are learned. LwF shows steadily increasing drift, whereas InTAct keeps activation changes bounded, indicating substantially more stable internal representations.

Conclusions from the Visualization. The results show a stark contrast between our method (InTAct) and the LwF baseline. For LwF (dashed lines), the activation drift is both significant and cumulative. As new tasks are introduced, the internal representations for data from all previous tasks drift substantially. For instance, the reference sample from Task 1 (blue dashed line) experiences a large initial drift that continues to increase as more tasks are added. In contrast, InTAct (solid lines) demonstrates superior stability. While a small, initial drift is observed when the first new task is introduced (e.g., the blue solid line at Task 2), the drift remains bounded and stabilizes for all subsequent tasks. This visualization provides strong evidence that InTAct success-

fully preserves the functional behavior of the network on data from previous tasks, directly mitigating a key cause of catastrophic forgetting.