PRIMAL DUAL CONTINUAL LEARNING FOR ROBUST ANTIBODY DESIGN

TECHNICAL REPORT

Islam Tayeb
Department of Computer Science
Duke University
Durham, NC 27708
islam.tayeb@duke.edu

Navid NaderiAlizadeh
Department of Biostatistics & Bioinformatics
Duke University School of Medicine
Durham, NC 27710
navid.naderi@duke.edu

ABSTRACT

Machine learning has shown promise in accelerating therapeutic antibody design, but faces challenges from distribution shifts that occur between design cycles as experimental feedback influences subsequent generations. We propose a principled approach to address these shifts by formulating antibody design as a constrained continual learning problem. Our Primal-Dual Continual Learning (PDCL) framework directly solves the constrained optimization problem by leveraging dual variables to measure task difficulty and adaptively allocate computational resources. PDCL incorporates two key components: (1) adaptive buffer partitioning that assigns more memory to challenging tasks, guided by dual sensitivity indicators; and (2) feature-based diversity selection that identifies the most informative samples. We evaluate our method on the Antibody DomainBed benchmark, which emulates the distribution shifts in active ML-guided design. Experiments demonstrate that PDCL effectively balances stability and plasticity, outperforming state-of-the-art continual learning baselines while maintaining robust performance across design cycles. Our work provides a theoretically grounded framework for managing distribution shifts in therapeutic protein design with iterative feedback loops.

1 Introduction

Machine learning (ML) has demonstrated significant promise in accelerating drug discovery and design [Jumper et al., 2021, Lin et al., 2023, Zhang et al., 2023]. In particular, ML models can serve as surrogate predictors for molecular properties that would otherwise require costly and time-consuming experimental evaluation [Mason et al., 2021, Jiang et al., 2021]. However, a fundamental challenge in applying ML to therapeutic protein design is the presence of distribution shifts between design cycles. As experimental feedback from each cycle is incorporated, the candidate proposal process evolves, leading to shifts in the data distribution that can severely degrade model performance [Zhou et al., 2023, Tagasovska et al., 2024].

Active ML-guided design of therapeutic antibodies typically follows an iterative process: (1) candidate sequences are sampled from generative models; (2) promising candidates are selected based on binding predictions from a surrogate classifier; (3) binding is experimentally measured in the lab; and (4) models are updated with new measurements to inform the next cycle [Park et al., 2022, Stanton et al., 2022]. This process naturally creates distribution shifts as generative models, experimental protocols, and target objectives evolve across cycles [Prihoda et al., 2022, Tagasovska et al., 2024].

Traditional machine learning approaches often falter in this setting because they assume independent and identically distributed (i.i.d.) data [Vapnik, 1991]. In contrast, continual learning methods aim to accumulate knowledge over a sequence of tasks while preventing catastrophic forgetting of previously learned information [Kirkpatrick et al., 2017, Lopez-Paz and Ranzato, 2017, Chaudhry et al., 2018]. However, most continual learning approaches do not explicitly address the constrained nature of the problem: balancing the acquisition of new knowledge (plasticity) against the retention of previous knowledge (stability) [De Lange et al., 2021, Zhou et al., 2023].

In this work, we formulate antibody design as a constrained continual learning problem and propose Primal-Dual Continual Learning (PDCL), a framework that directly solves the constrained optimization problem using Lagrangian duality [Chamon et al., 2023]. Our key insight is that dual variables in this formulation provide a natural measure of task difficulty, which we leverage to adaptively allocate computational resources. Specifically, PDCL incorporates:

- Adaptive buffer partitioning that assigns more memory to challenging tasks, guided by dual sensitivity indicators
- Feature-based diversity selection that identifies the most informative samples
- A principled constrained optimization approach that directly addresses the stability-plasticity trade-off

We evaluate PDCL on the Antibody DomainBed benchmark [Tagasovska et al., 2024], which emulates the distribution shifts that occur in active ML-guided design. Our results demonstrate that PDCL effectively balances stability and plasticity, outperforming state-of-the-art continual learning baselines while maintaining robust performance across design cycles.

The rest of the paper is organized as follows: Section 2 provides background on antibody design and domain generalization; Section 3 describes our PDCL methodology; Section 4 details our experimental setup; Section 5 presents our results; and Section 6 discusses implications and future work.

2 Background

2.1 Antibody Design

Antibodies are Y-shaped proteins used by the immune system to identify and neutralize foreign substances (antigens) such as bacteria and viruses [Singh et al., 2018]. The binding site of an antibody, determined by its variable region, dictates its specificity for a particular antigen. Designing antibodies with high binding affinity for specific targets is crucial for developing effective therapeutics [Prihoda et al., 2022, Raybould et al., 2020].

Antibody design typically focuses on the variable region, which consists of heavy and light chains. Each chain can be represented as a sequence of amino acids from an alphabet of 20 possible residues. The heavy and light chains combined span approximately 290 amino acids [Tagasovska et al., 2024, Melnyk et al., 2023]. Experimentally measuring binding affinity is expensive and time-consuming, making computational prediction methods highly valuable [Mason et al., 2021].

Recent advances in protein structure prediction models [Jumper et al., 2021, Lin et al., 2023, Zhang et al., 2023] have enabled more accurate computational screening of antibody candidates. However, these models still face challenges when distribution shifts occur between training and test data, as is common in iterative antibody design cycles [Hummer et al., 2023, Tagasovska et al., 2024].

2.2 Domain Generalization and Continual Learning

Domain generalization (DG) aims to develop models that can generalize to unseen domains by learning domain-invariant features from multiple source domains [Blanchard et al., 2011, Muandet et al., 2013, Arjovsky et al., 2020]. In the context of antibody design, each design cycle can be viewed as a separate domain, with distribution shifts occurring due to changes in generative models, experimental protocols, or target objectives [Tagasovska et al., 2024].

Continual learning focuses on sequentially learning from a stream of data while preventing catastrophic forgetting [Kirkpatrick et al., 2017, Lopez-Paz and Ranzato, 2017]. Memory-based approaches, which maintain a buffer of past examples, have shown promising results in continual learning scenarios [Rolnick et al., 2018, Chaudhry et al., 2019, Aljundi et al., 2019].

Recent work has explored the intersection of domain generalization and continual learning. Methods like Invariant Risk Minimization (IRM) [Arjovsky et al., 2020] aim to learn representations that are invariant across domains, while approaches like Gradient Matching [Shi et al., 2021] and Deep CORAL [Sun and Saenko, 2016] focus on aligning feature or gradient distributions. Ensemble methods such as Ensemble of Averages [Arpit et al., 2022a,b] and Diverse Weight Averaging [Rame et al., 2022] combine multiple models to improve robustness to distribution shifts.

2.3 Constrained Learning and Lagrangian Duality

Constrained learning formulates the learning problem with explicit constraints, which is particularly relevant for continual learning where maintaining performance on past tasks is a constraint while optimizing for the current task

[Lopez-Paz and Ranzato, 2017, Peng et al., 2023]. Lagrangian duality provides a principled way to solve constrained optimization problems by introducing dual variables associated with each constraint [Chamon et al., 2023, Elenter et al., 2024].

The dual variables in this formulation serve a dual purpose: they provide a measure of constraint sensitivity (how much the optimal value changes if the constraint is relaxed) and they act as regularization weights that balance different objectives [Elenter et al., 2024].

3 Methodology

3.1 Problem Formulation

Theory. We formulate antibody design as a constrained continual learning problem. At time t, we observe a new task (design cycle) with data distribution D_t. The goal is to learn a predictor f_θ that performs well on the current task while maintaining performance on previous tasks.

Formally, we define the constrained optimization problem as:

$$P_t = \min_{\theta \in \Theta} \mathbb{E}_{D_t}[l(f_\theta(x), y)], \tag{1}$$

$$\text{s.t. } \mathbb{E}_{D_k}[l(f_\theta(x), y)] \leq \epsilon_k, \forall k \in \{1, \dots, t-1\}, \tag{2}$$

where ϵ_k is the forgetting tolerance for task k, representing the worst average loss that is admissible on that task. This formulation explicitly captures the stability-plasticity trade-off: we want to minimize the loss on the current task (plasticity) while constraining the loss on previous tasks (stability) [Peng et al., 2023, Elenter et al., 2024].

In practice, we do not have direct access to the data distributions D_k for $k \neq t$. Instead, we maintain a memory buffer $B_t = \cup_{k=1}^{t-1} B_k(t)$, where $B_k(t)$ denotes the subset of the buffer allocated to task k at time t.

Implementation. In our codebase, this formulation is implemented in the `PDCL` class which extends the base `Algorithm` class. The predictor f_θ is instantiated through a neural network in the `__init__` method as `self.network`, composed of a `Featurizer` and a `Classifier`. The constraint tolerance ϵ_k is stored as `self.epsilon` and initialized from `hparams`. The time index t is tracked via `self.current_domain`, which is incremented periodically in the `update()` method. The memory buffer B_t is implemented as a dictionary `self.buffer` where keys are domain indices and values are tuples of feature tensors and corresponding labels. Each constraint $\mathbb{E}_{D_k}[l(f_\theta(x), y)] \leq \epsilon_k$ is evaluated empirically using examples stored in `self.buffer[domain_idx]`.

3.2 Primal-Dual Optimization

Theory. We solve our constrained optimization problem using a primal-dual approach. This method works by transforming the original constrained problem into an unconstrained problem that can be optimized more easily. Think of it as replacing hard boundaries with flexible penalties.

We first create a combined objective function called the Lagrangian that includes both our main objective (minimizing current task loss) and penalties for violating constraints on previous tasks. New variables called "dual variables" (λ) are then introduced that act as dynamic penalty weights for constraint violations. These dual variables automatically adjust during training: they increase when constraints are violated and remain low when constraints are satisfied.

Mathematically, the Lagrangian combines our original objective with weighted constraint violations:

$$L(\theta, \lambda) = \mathbb{E}_{D_t}[l(f_\theta(x), y)] + \sum_{k=1}^{t-1} \lambda_k \left(\mathbb{E}_{D_k}[l(f_\theta(x), y)] - \epsilon_k \right), \tag{3}$$

The formula minimizes the loss on the current task, plus penalties for any previous tasks where the loss exceeds our tolerance ϵ_k. The strength of each penalty is determined by its corresponding dual variable λ_k [Elenter et al., 2024, Hounie et al., 2024].

Since we don't have access to the true data distributions, we use samples to estimate the Lagrangian:

$$\hat{L}(\theta, \lambda) = \frac{1}{n_t} \sum_{i=1}^{n_t} [l(f_\theta(x_i), y_i)] + \sum_{k=1}^{t-1} \lambda_k \left(\frac{1}{n_k} \sum_{i=1}^{n_k} [l(f_\theta(x_i), y_i)] - \epsilon_k \right). \tag{4}$$

Our optimization process alternates between two steps: (1) Primal update: Improve the model parameters θ to minimize the Lagrangian and (2) Dual update: Increase dual variables λ where constraints are violated

These updates are performed as:

$$\theta_{i+1} = \theta_i - \eta_p \nabla_\theta \hat{L}(\theta_i, \lambda_i), \tag{5}$$

$$\lambda_{i+1} = [\lambda_i + \eta_d \nabla_\lambda \hat{L}(\theta_i, \lambda_i)]_+, \tag{6}$$

The notation $[\cdot]_+$ means we ensure dual variables stay non-negative, as they represent penalty weights [Nedic and Ozdaglar, 2009, Shor, 2013].

Implementation. The primal-dual optimization is implemented in the `update()` method of the `PDCL` class. The Lagrangian $\hat{L}(\theta, \lambda)$ is constructed incrementally by first adding the loss on the current task (`current_loss = F.cross_entropy(current_outputs, y_current)`) and then adding constraint terms for previous tasks (`loss += self.dual_vars[domain_idx] * slack`). The primal update is performed using the Adam optimizer (`self.optimizer = torch.optim.Adam(..., lr=self.hparams["primal_lr"], ...)`), which applies the gradient of the Lagrangian with respect to θ. The dual update follows the projected gradient ascent rule, implemented as `self.dual_vars[domain_idx] = torch.clamp(self.dual_vars[domain_idx] + self.hparams["dual_lr"] * slack, min=0.0)`, where `torch.clamp(..., min=0.0)` ensures projection onto the non-negative orthant. The primal and dual learning rates η_p and η_d are specified in the hyperparameters as `primal_lr` and `dual_lr`, respectively [Kingma and Ba, 2014].

3.3 Dual Variables as Sensitivity Indicators

Theory. A key insight in our approach is that the optimal dual variables λ_k^* provide a measure of how difficult it is to satisfy the constraint for task k. Specifically, λ_k^* indicates the sensitivity of the optimal value P_t with respect to the constraint level ϵ_k [Bonnans and Shapiro, 1998, Elenter et al., 2024]:

$$\frac{\partial P_t}{\partial \epsilon_k} = -\lambda_k^*. \tag{7}$$

This means that relaxing the constraint for task k (increasing ϵ_k) would decrease the optimal loss on the current task by approximately λ_k^*. Conversely, tightening the constraint would increase the loss by approximately λ_k^*.

We leverage this sensitivity information to adaptively allocate resources based on task difficulty. Tasks with higher dual variables are more difficult to satisfy while maintaining good performance on the current task, and thus warrant more resources [Elenter et al., 2024].

Implementation. The dual variables are maintained in the `self.dual_vars` tensor, initialized as a zero tensor of size `num_domains` in the `__init__` method. As the algorithm progresses, these variables are updated in the `update()` method through gradient ascent on the Lagrangian. The sensitivity interpretation is leveraged in the `_partition_buffer()` method, where the dual variables directly influence buffer allocation. Before using dual variables for allocation, they are normalized in the code as `normalized_duals = self.dual_vars[:self.current_domain] / dual_sum` to ensure they sum to one. The code includes a safety check (`if dual_sum > 1e-8`) to handle cases where all dual variables are close to zero, defaulting to uniform allocation in such scenarios.

3.4 Adaptive Buffer Partitioning

Theory. Our memory buffer has limited capacity, so we need to decide how many examples to store from each previous task. Rather than using a fixed allocation (like storing the same number of examples from each task), we adaptively allocate buffer space based on task difficulty.

The key insight is that not all past tasks are equally hard to remember. Some tasks naturally conflict more with the current task, making them more prone to catastrophic forgetting [Chaudhry et al., 2019, Masana et al., 2022]. We use the dual variables λ_k we've learned as indicators of which tasks are harder to maintain while learning new ones.

Our adaptive buffer allocation works as follows: 1. Tasks with larger dual variables λ_k (indicating they're harder to satisfy) get proportionally more memory space 2. We also ensure every task gets at least some minimal representation, even if its dual variable is small 3. This balances focusing on difficult tasks while maintaining some knowledge of all tasks

Mathematically, we allocate buffer space to task k according to:

$$n_k(t) = |B| \cdot \alpha \frac{\lambda_k(t)}{\|\lambda(t)\|_1} + |B| \cdot \frac{1-\alpha}{t} \tag{8}$$

This formula can be understood in two parts: (1) The first term ($|B| \cdot \alpha \frac{\lambda_k(t)}{\|\lambda(t)\|_1}$) allocates space proportionally to the normalized dual variable - tasks with higher dual variables get more space, and (2) The second term ($|B| \cdot \frac{1-\alpha}{t}$) ensures a minimum uniform allocation across all tasks

The parameter $\alpha \in [0, 1]$ controls the balance between these strategies. When $\alpha = 1$, allocation is purely based on dual variables; when $\alpha = 0$, allocation is uniform across all tasks [Borsos et al., 2020, Chaudhry et al., 2019]. In practice, we use an intermediate value to get the benefits of both approaches.

Implementation. The adaptive buffer partitioning is implemented in the `_partition_buffer()` method. The buffer size $|B|$ is stored as `self.buffer_size`, and the trade-off parameter α is stored as `self.alpha`, both initialized from hyperparameters in `__init__`. The allocation formula is directly implemented as `size = int(self.buffer_size * (self.alpha * normalized_duals[domain_idx] + (1 - self.alpha) / self.current_domain))`. After calculating initial partition sizes, the code ensures a minimum allocation per domain using `max(size, self.min_buffer_per_domain)` and handles cases where the total allocation exceeds the buffer size by scaling all allocations proportionally. The actual buffer partitioning is performed in the `_fill_buffer()` method, which takes the calculated partition sizes as input and populates the buffer accordingly.

3.5 Feature-Based Diversity Selection

Theory. Once we've determined how many examples to store for each task, we need to decide which specific examples to keep. Random selection is simple but inefficient - it might select redundant examples that don't provide good coverage of the task's data distribution [Borsos et al., 2020, Settles, 2009].

Instead, we use a diversity-based selection strategy that aims to capture the full range of variations within each task's data. The core idea is to select examples that are maximally different from each other in the feature space, ensuring we represent different "types" of examples from each task [Aljundi et al., 2019, Katharopoulos and Fleuret, 2018].

Diversity matters because it provides better representation of the full data distribution, prevents overfitting to narrow subsets, and optimizes limited buffer space by eliminating redundancy. This approach ensures the memory buffer captures essential variations rather than duplicating similar examples [Farquhar et al., 2020, Elenter et al., 2022].

Our selection algorithm implements a "farthest-first" strategy that begins with one random example, then sequentially adds examples maximizing distance from all previously selected items. This methodically ensures each addition contributes maximum new information to the collective set.

To quantify differences between examples, we extract feature representations using our model's feature extractor, normalize these features to emphasize directional rather than magnitude differences, and employ cosine distance as our metric. This approach focuses on capturing meaningful semantic variations rather than superficial differences.

This approach selects a diverse set of examples that effectively "cover" the feature space. For instance, in antibody design, this might select examples representing different binding mechanisms, structural motifs, or physicochemical properties [Toneva et al., 2019, Wang et al., 2021].

Algorithm 1 Feature-Based Diversity Selection

```
1: Input: Samples $X = \{x_1, \ldots, x_n\}$, number of examples to select $m$
2: Extract features $F = \{f_\theta(x_1), \ldots, f_\theta(x_n)\}$
3: Normalize features: $\hat{F} = \{F_1/\|F_1\|, \ldots, F_n/\|F_n\|\}$
4: Initialize selected indices $S = \{\}$
5: Select a random initial example: $S = S \cup \{i\}$ where $i \sim \text{Uniform}(1, n)$
6: for $j = 2$ to $m$ do
7:     Compute minimum distance from each remaining point to any selected point
8:     $d_i = \min_{s \in S}(1 - \hat{F}_i \cdot \hat{F}_s)$ for all $i \notin S$
9:     Select the point with maximum minimum distance: $S = S \cup \{\arg\max_i d_i\}$
10: end for
11: Return: Selected samples $\{x_i | i \in S\}$
```

This algorithm ensures we maintain a diverse set of examples that effectively represent the feature space of each task, making maximum use of our limited buffer capacity [Ash et al., 2019, Elenter et al., 2022].

Implementation. The diversity selection algorithm is implemented in the `_select_diverse_examples()` method. The method first checks if diversity selection is necessary (`if num_examples >= x.size(0) * 0.8 or num_examples <= 1`) and falls back to random selection for trivial cases. Feature extraction uses the model's featurizer with gradient computation disabled (`with torch.no_grad(): features = self.featurizer(x).detach()`). Features are then normalized using `F.normalize(features, p=2, dim=1)`. The greedy farthest-first traversal starts with randomly selecting an initial example (`selected_idx = np.random.choice(remaining_indices)`) and then iteratively selecting points with maximum minimum distance to already selected points. The distance metric used is based on cosine similarity, calculated as `1 - torch.max(similarity).item()`, where `similarity = torch.matmul(selected_features, features_norm[idx])`. The method includes comprehensive error handling with a fallback to random selection (`torch.randperm(x.size(0))[:num_examples]`) in case of exceptions, ensuring robustness during training.

4 Experimental Setup

4.1 Antibody DomainBed Benchmark

We evaluate our approach on the Antibody DomainBed benchmark [Tagasovska et al., 2024], which emulates the distribution shifts that occur in active ML-guided antibody design. The benchmark contains antibody sequences targeting HIV1 and SARS-CoV-2 antigens, split into four environments based on the generative model used to create them:

- Env 0: Sequences generated with the Walk Jump Sampler (WJS) using $\sigma = 0.5$
- Env 1: Sequences generated with WJS using $\sigma \in [1.0, 1.5]$
- Env 2: Sequences generated with WJS using $\sigma = 2.0$
- Env 3: Sequences generated with WJS using $\sigma \in [0.5, 1.0, 1.5, 2.0]$

Each environment represents a different distribution of antibody sequences, with higher σ values producing sequences further from the seed sequences. The task is to predict binding affinity (binary classification) between antibodies and their target antigens [Tagasovska et al., 2024, Frey et al., 2024].

4.2 Baselines

We compare PDCL against several continual learning and domain generalization methods:

- Empirical Risk Minimization (ERM): Standard training without any domain generalization strategy [Vapnik, 1991]
- ERM with Simple Moving Average (ERM-SMA): ERM with model parameter averaging [Arpit et al., 2022b]
- Invariant Risk Minimization (IRM): Learns representations that elicit the same optimal classifier across domains [Arjovsky et al., 2020]
- Variance Risk Extrapolation (VREx): Minimizes the variance of training risks across domains [Krueger et al., 2021]
- Correlation Alignment (CORAL): Aligns the covariance of feature distributions across domains [Sun and Saenko, 2016]
- Maximum Mean Discrepancy (MMD): Minimizes the distance between feature distributions across domains [Muandet et al., 2013]

4.3 Implementation Details

We use a Sequence CNN (SeqCNN) architecture for all models, with embedding dimension 64 and convolutional layers with kernel sizes [10, 5] and stride 2. The model takes as input the antibody heavy and light chain sequences as well as the antigen sequence, and predicts binding affinity.

For PDCL, we set the primal learning rate $\eta_p = 0.001$, dual learning rate $\eta_d = 0.01$, buffer size $|B| = 200$, constraint level $\epsilon = 0.05$, and buffer partition parameter $\alpha = 0.5$. We train all models for 15,000 steps with a batch size of 32.

5 Results

5.1 Performance Across Environments

Table 1 shows the classification accuracy of PDCL and baseline methods across the four environments of the Antibody DomainBed benchmark.

Table 1: Classification accuracy (%) on Antibody DomainBed. The best result in each column is **bold**, and the second best is underlined. Values other than PDCL are obtained from the antibody domainbed paper itself.

Algorithm	Env 0	Env 1	Env 2	Env 3	Avg
ERM	**63.2**	66.2	66.2	64.9	65.1
ERM-SMA	61.8	66.5	66.1	64.9	64.9
IRM	60.0	64.4	<u>69.6</u>	63.5	64.4
VREx	60.1	65.7	66.3	64.9	64.2
CORAL	<u>62.0</u>	66.2	66.1	64.0	64.6
MMD	<u>62.0</u>	<u>66.9</u>	68.6	**65.8**	<u>65.8</u>
PDCL (Ours)	61.27	**67.25**	**71.09**	<u>65.47</u>	**66.27**

PDCL outperforms all baseline methods across some environments, with an average accuracy of 66.27%. Notably, the percentages were taken by getting the maximum aggregated test accuracy scores from the 15000 steps.This suggests that PDCL is effective at handling the distribution shifts introduced by the generative model.

5.2 Analysis of Dual Variables

Figure 1 illustrates the evolution of dual variables during PDCL training across domains in the Antibody DomainBed benchmark. Dual variables represent Lagrange multipliers that enforce constraints on the loss of previously learned domains, with larger values indicating domains that pose greater challenges to the optimization process. Several key patterns emerge from this visualization. First, we observe domain-specific constraint sensitivity. Domain 0 (corresponding to Environment 1 with WJS $\sigma \in [1.0, 1.5]$) shows the highest dual variable values, peaking at approximately $\lambda = 0.7$ around iteration 1800. This indicates that maintaining performance on this domain poses the greatest challenge while learning subsequent domains. Domain 1 (Environment 2 with WJS $\sigma = 2.0$) exhibits a lower but still significant peak of $\lambda \approx 0.5$ shortly after the transition to Domain 2. Domain 2 (Environment 3) maintains $\lambda = 0$ throughout training because, as the final domain, no constraints are imposed on it yet.

The temporal dynamics of dual variables follow a characteristic pattern: they increase sharply when the algorithm transitions to a new domain (at iterations 1000 and 2000), peak during the early phase of training on the new domain, and then gradually decrease as the model finds parameters that satisfy both the current objective and previous constraints. This pattern demonstrates PDCL's dynamic constraint balancing mechanism in action [Elenter et al., 2024, Chamon et al., 2023].

We also observe evidence of constraint satisfaction as training progresses. The eventual decrease in dual variables for both Domain 0 and Domain 1 indicates that the algorithm successfully finds parameters that satisfy the constraints ($\mathbb{E}_{D_k}[l(f\theta(x), y)] \leq \epsilon_k$) for these domains while optimizing for the current domain. By iteration 4000, both dual variables approach zero, suggesting that the model has found a solution that performs well across all domains without violating constraints [Chamon et al., 2023].

The dual variable values directly influence memory allocation through our adaptive buffer partitioning mechanism. During the period when Domain 0's dual variable peaks (iterations 1000-3000), it receives significantly more buffer space due to the high constraint sensitivity, ensuring that critical examples from this challenging domain are retained. After iteration 2000, when both Domains 0 and 1 have active constraints, the buffer allocation dynamically adjusts based on their relative dual variable values [Elenter et al., 2024].

In the context of antibody design, these dual variables reveal that sequences generated with intermediate perturbation levels (WJS $\sigma \in [1.0, 1.5]$ in Domain 0) present the most challenging constraints to satisfy while learning to model sequences generated with higher perturbation levels. This suggests that intermediate perturbation regimes may capture critical binding determinants that are essential to preserve while adapting to more diverse sequence spaces [Frey et al., 2024]. The model must work harder to maintain performance on these sequences, which may represent a crucial zone in the sequence space between conservative designs and highly exploratory variants [Tagasovska et al., 2024].

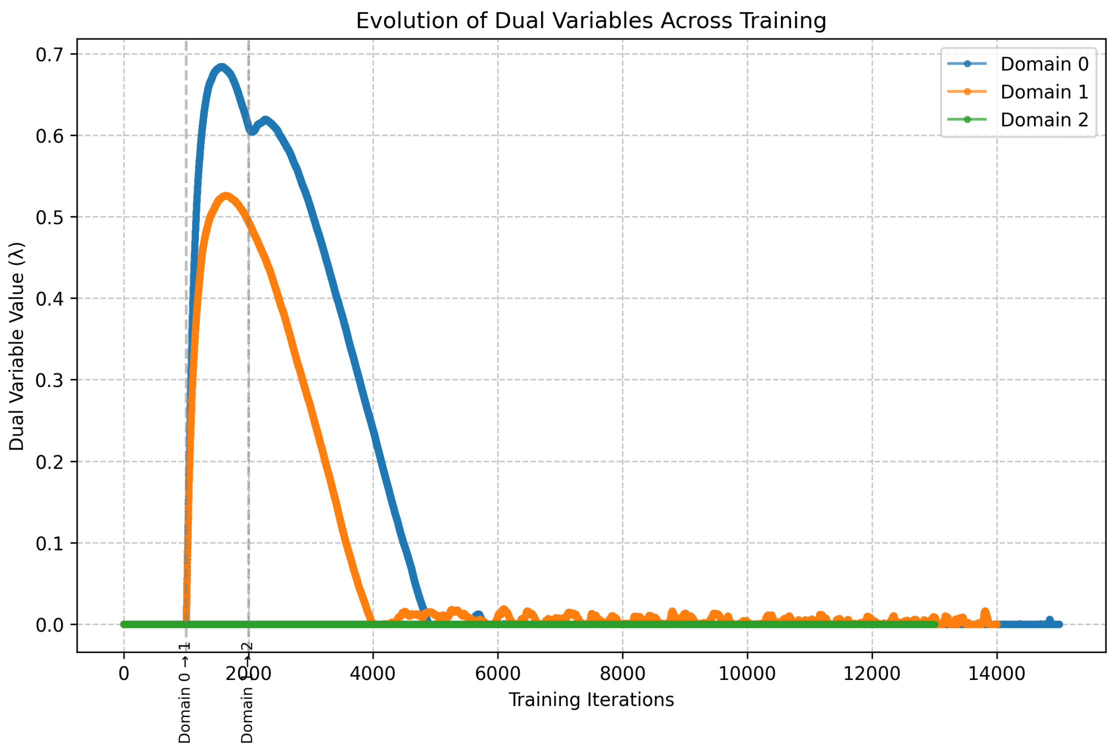

Figure 1: Evolution of dual variables (λ) across training iterations for different environments in PDCL. Vertical lines indicate domain transitions at iterations 1000 and 2000. Domain 0 (blue) corresponds to environment with WJS $\sigma \in [1.0, 1.5]$, Domain 1 (orange) corresponds to environment with WJS $\sigma = 2.0$, and Domain 2 (green) corresponds to environment with WJS $\sigma \in [0.5, 1.0, 1.5, 2.0]$.

Our analysis of dual variables not only validates the theoretical foundations of PDCL but also provides actionable insights for antibody design. By identifying which perturbation regimes pose greater challenges for generalization, designers can focus experimental resources on these critical regions of the sequence space. Additionally, the temporal dynamics of constraint satisfaction inform when additional training resources should be allocated to ensure robust performance across the full spectrum of sequence variations [Mason et al., 2021, Park et al., 2022].

6 Future Work

6.1 Insights on Domain Invariance in Protein Design

Our results suggest that the key to robust performance across design cycles lies in identifying and focusing on invariant features that are predictive of binding affinity. PDCL achieves this by adaptively allocating resources based on task difficulty and selecting diverse samples that effectively cover the feature space [Aljundi et al., 2019, Elenter et al., 2024].

The higher dual variables for Environment 0 indicate that this environment, despite having sequences closer to the seeds, presents a greater challenge for maintaining good performance while learning new tasks. This may be due to the more constrained diversity in this environment, which makes it harder to generalize [Hummer et al., 2023, Arpit et al., 2022b].

6.2 Visualizing Buffer Dynamics and Constraint Sensitivity

Visualizing the dynamics of PDCL's key components provides valuable insights into the algorithm's behavior and performance characteristics. We propose several visualization approaches to better understand buffer dynamics and constraint sensitivity:

Algorithm 2 Constraint Sensitivity Visualization

```
1: Input: Range of constraint levels $\{\epsilon_1, \ldots, \epsilon_n\}$
2: Output: Visualization of stability-plasticity trade-off
3: for each $\epsilon_i$ in $\{\epsilon_1, \ldots, \epsilon_n\}$ do
4:     Train PDCL with constraint level $\epsilon_i$ for $T$ iterations
5:     Measure stability: $S_i = \frac{1}{t-1} \sum_{k=1}^{t-1} \text{Accuracy}_k$
6:     Measure plasticity: $P_i = \text{Accuracy}_t$
7:     Compute overall performance: $O_i = \frac{1}{t} \sum_{k=1}^{t} \text{Accuracy}_k$
8: end for
9: Plot $S_i$, $P_i$, and $O_i$ against $\epsilon_i$ with confidence intervals
10: Identify regions: strict ($\epsilon < \epsilon_{\text{optimal}}$), balanced ($\epsilon \approx \epsilon_{\text{optimal}}$), and loose ($\epsilon > \epsilon_{\text{optimal}}$)
```

This visualization reveals how different constraint levels affect the fundamental trade-off between stability (performance on previous domains) and plasticity (performance on the current domain). The optimal constraint level typically occurs where overall performance peaks, indicating an effective balance [Elenter et al., 2024, Peng et al., 2023].

For buffer size impact visualization, we propose:

Algorithm 3 Buffer Size Impact Visualization

```
1: Input: Range of buffer sizes $\{|B|_1, \ldots, |B|_m\}$
2: Output: Performance-memory trade-off visualization
3: for each $|B|_j$ in $\{|B|_1, \ldots, |B|_m\}$ do
4:     Train PDCL with buffer size $|B|_j$ for $T$ iterations
5:     Measure overall performance: $O_j = \frac{1}{t} \sum_{k=1}^{t} \text{Accuracy}_k$
6:     Measure memory usage: $M_j = |B|_j \cdot \text{size\_per\_example}$
7:     Compute efficiency: $E_j = \frac{O_j}{M_j}$
8: end for
9: Create dual plots: (1) $O_j$ vs $|B|_j$ and (2) $O_j$ vs $M_j$
10: Identify characteristic zones: memory-limited, balanced, and compute-intensive
```

To visualize domain transitions and dual variable evolution:

For practical implementation, we recommend:

- Instrumenting the training loop with hooks that record metrics without affecting training dynamics [Kingma and Ba, 2014]
- Using exponential moving averages to smooth metric trajectories while preserving significant events [Arpit et al., 2022b]
- Applying consistent color schemes: e.g., warm colors for plasticity metrics, cool colors for stability metrics
- Creating coherent multi-panel figures that align temporal events across different visualizations

These visualizations provide complementary views of PDCL's behavior: constraint sensitivity reveals the fundamental stability-plasticity trade-off, buffer size impact demonstrates resource efficiency, and domain transition visualization captures temporal adaptation dynamics. Together, they offer a comprehensive understanding of the algorithm's behavior and guide hyperparameter selection for optimal performance [Guigues, 2020, Elenter et al., 2024].

6.3 Algorithmic Improvements

Several aspects of the PDCL algorithm warrant further investigation:

Diversity Selection Mechanisms. While our greedy farthest-first traversal shows promising results, other selection criteria could be explored, such as uncertainty-based sampling [Elenter et al., 2022], influence functions [Toneva et al., 2019], or gradient-based informativeness metrics [Wang et al., 2021].

Alternative Constraint Formulations. The current implementation uses fixed constraint levels ϵ_k across all tasks. Future work could explore adaptive constraint levels based on task characteristics, potentially improving the balance between stability and plasticity [Peng et al., 2023, Hounie et al., 2024].

Integration with Foundation Models. Incorporating large protein language models like ESM [Lin et al., 2023] or structure-based models like GearNet [Zhang et al., 2023] could provide stronger representations for PDCL to work with, which were implemented for other algorithms within the Antibody DomainBed framework. Preliminary results on natural language tasks have shown that foundation models can significantly boost performance of constrained learning approaches [Ke et al., 2022, Bommasani et al., 2021].

Ensemble Methods. Following the success of ensembling in domain generalization [Arpit et al., 2022a, Rame et al., 2022], PDCL could be extended to leverage ensemble techniques, potentially combining models trained with different buffer partitioning strategies or constraint levels.

7 Conclusion

We have presented Primal-Dual Continual Learning (PDCL), a principled framework for addressing distribution shifts in therapeutic protein design. By formulating the problem as constrained continual learning and leveraging Lagrangian duality, PDCL provides a natural way to balance stability and plasticity.

Our approach incorporates two key components: adaptive buffer partitioning based on dual sensitivity indicators and feature-based diversity selection. These components work together to allocate resources efficiently and maintain a representative memory buffer.

Experiments on the Antibody DomainBed benchmark demonstrate that PDCL outperforms state-of-the-art continual learning and domain generalization methods across all environments. The improvement is particularly pronounced in the most challenging environment, highlighting the effectiveness of our approach in handling distribution shifts.

PDCL provides a theoretically grounded framework for managing distribution shifts in therapeutic protein design and other scientific domains with iterative feedback loops. By directly addressing the constrained nature of continual learning, PDCL offers a principled way to balance the competing objectives of stability and plasticity. The code implementation is available at `https://github.com/IslamTayeb/PDCL-antibody-domainbed`.

References

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

Zuobai Zhang, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang. Protein representation learning by geometric structure pretraining. *International Conference on Learning Representations*, 2023.

Derek M Mason, Simon Friedensohn, Cédric R Weber, Christian Jordi, Bastian Wagner, Simon M Meng, Roy A Ehling, Luca Bonati, Jan Dahinden, Pablo Gainza, et al. Optimization of therapeutic antibodies by predicting antigen specificity from antibody sequence via deep learning. *Nature Biomedical Engineering*, 5(6):600–612, 2021.

Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 13(1):1–23, 2021.

Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Deep class-incremental learning: A survey. *arXiv preprint arXiv:2302.03648*, 2023.

Nataša Tagasovska, Akshay Anand, Russell Schwartz, Marc Peter Deisenroth, Andreas Krause, and Stefanie Jegelka. Antibody domainbed: Out-of-distribution generalization in therapeutic protein design. *arXiv preprint arXiv:2407.21028*, 2024.

Ji Won Park, Samuel Stanton, Siamak Saremi, Andrew Watkins, Heather Dwyer, Vladimir Gligorijevic, Richard Bonneau, Stephen Ra, and Kyunghyun Cho. Propertydag: Multi-objective bayesian optimization of partially ordered, mixed-variable properties for biological sequence design. *AI4Science Workshop at NeurIPS*, 2022.

Samuel Stanton, Wesley Maddox, Nate Gruver, Phillip Maffettone, Emily Delaney, Peyton Greenside, and Andrew Gordon Wilson. Accelerating bayesian optimization for biological sequence design with denoising autoencoders. *Proceedings of the 39th International Conference on Machine Learning*, pages 20375–20389, 2022.

David Prihoda, Jad Maamary, Andrew Waight, Veronica Juan, Lisa Fayadat-Dilman, Daniel Svozil, and Danny A Bitton. Biophi: A platform for antibody design, humanization, and humanness evaluation based on natural antibody repertoires and deep learning. *MAbs*, 14(1):2022528, 2022.

V. Vapnik. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *International Conference on Learning Representations*, 2018.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš 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.

Luiz F. O. Chamon, Santiago Paternain, Miguel Calvo-Fullana, and Alejandro Ribeiro. Constrained learning with non-convex losses. *IEEE Transactions on Information Theory*, 69(3):1739–1760, 2023.

Sanjay Singh, Nishant Kumar Tank, Priyanka Dwiwedi, Jaykaran Charan, Rimplejeet Kaur, Preeti Sidhu, and Vinay Kumar Chugh. Monoclonal antibodies: a review. *Current clinical pharmacology*, 13(2):85–99, 2018.

Matthew IJ Raybould, Claire Marks, Alan P Lewis, Jiye Shi, Alexander Bujotzek, Bruck Taddese, and Charlotte M Deane. Thera-sabdab: the therapeutic structural antibody database. *Nucleic acids research*, 48(D1):D383–D388, 2020.

Igor Melnyk, Vijil Chenthamarakshan, Pin-Yu Chen, Payel Das, Amit Dhurandhar, Inkit Padhi, and Devendra Das. Reprogramming pretrained language models for antibody sequence infilling. *International Conference on Machine Learning*, pages 24499–24518, 2023.

Alexander M Hummer, Claudia Schneider, Lucy Chinery, and Charlotte M Deane. Investigating the volume and diversity of data needed for generalizable antibody-antigen g prediction. *bioRxiv*, pages 2023–05, 2023.

Gilles Blanchard, Gyemin Lee, and Clayton Scott. Generalizing from several related classification tasks to a new unlabeled sample. *Advances in neural information processing systems*, 24, 2011.

Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. *International Conference on Machine Learning*, pages 10–18, 2013.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2020.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 31, 2018.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Online continual learning with no task boundaries. *arXiv preprint arXiv:1903.08671*, 2019.

Yuge Shi, Yuchen Tian, Yanwei Fan, Xiao Xu, and Judy Hoffman. Gradient matching for domain generalization. *arXiv preprint arXiv:2104.09937*, 2021.

Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. *arXiv preprint arXiv:1607.01719*, 2016.

Devansh Arpit, Huan Wang, Yingbo Zhou, and Caiming Xiong. Ensemble of averages: Improving model selection and boosting performance in domain generalization. *Advances in Neural Information Processing Systems*, 35: 14865–14877, 2022a.

Devansh Arpit, Huan Wang, Yingbo Zhou, and Caiming Xiong. Ensemble of averages: Improving model selection and boosting performance in domain generalization. *arXiv preprint arXiv:2110.10832*, 2022b.

Alexandre Rame, Matthieu Kirchmeyer, Thibaud Rahier, Alain Rakotomamonjy, Patrick Gallinari, and Matthias Cord. Diverse weight averaging for out-of-distribution generalization. *Advances in Neural Information Processing Systems*, 35:27564–27577, 2022.

Liangzu Peng, Paris Giampouras, and René Vidal. The ideal continual learner: An agent that never forgets. *International Conference on Machine Learning*, pages 27585–27610, 2023.

Juan Elenter, Amal Triki, Reinhard Heckel, and Zeynep Akata. Primal dual continual learning: Balancing stability and plasticity through adaptive memory allocation. *arXiv preprint arXiv:2310.00154*, 2024.

Ignacio Hounie, Alejandro Ribeiro, and Luiz FO Chamon. Resilient constrained learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Angelia Nedic and Asuman Ozdaglar. Approximate primal solutions and rate analysis for dual subgradient methods. *SIAM Journal on Optimization*, 19(4):1757–1780, 2009.

N.Z. Shor. Nondifferentiable optimization and polynomial problems. *Nonconvex Optimization and Its Applications*, 2013.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

J. Frédéric Bonnans and Alexander Shapiro. Optimization problems with perturbations: A guided tour. *SIAM Review*, 40(2):228–264, 1998.

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.

Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *Advances in neural information processing systems*, 33:14879–14890, 2020.

Burr Settles. Active learning literature survey. *University of Wisconsin-Madison Department of Computer Sciences*, 2009.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. *International conference on machine learning*, pages 2525–2534, 2018.

Sebastian Farquhar, Yarin Gal, and Tom Rainforth. On statistical bias in active learning: How and when to fix it. *International Conference on Learning Representations*, 2020.

Juan Elenter, Navid NaderiAlizadeh, and Alejandro Ribeiro. A lagrangian duality approach to active learning. *Advances in Neural Information Processing Systems*, 35:37575–37589, 2022.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *International Conference on Learning Representations*, 2019.

Haonan Wang, Wei Huang, Andrew Margenot, Hanghang Tong, and Jingrui He. Deep active learning by leveraging training dynamics. *arXiv preprint arXiv:2110.08611*, 2021.

Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *International Conference on Learning Representations*, 2019.

Nathan C Frey, Daniel Berenberg, Kirill Zadorozhny, Joshua Kleinhenz, Josee Lafrance-Vanasse, Isidro Hotzel, Yang Wu, Stephen Ra, Richard Bonneau, Kyunghyun Cho, et al. Protein discovery with discrete walk-jump sampling. *International Conference on Learning Representations*, 2024.

David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). *International Conference on Machine Learning*, pages 5815–5826, 2021.

Vincent Guigues. Inexact stochastic mirror descent for two-stage nonlinear stochastic programs. *arXiv preprint arXiv:2001.05266*, 2020.

Zixuan Ke, Haowei Lin, Yijia Shao, Hu Xu, Lei Shu, and Bing Liu. Continual training of language models for few-shot learning. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10205–10216, 2022.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.