I2I: INITIALIZING ADAPTERS WITH IMPROVISED KNOWLEDGE

Tejas Srinivasan[1], **Furong Jia**[1], **Mohammad Rostami**[1,2], **Jesse Thomason**[1]
[1]University of Southern California
[2]USC Information Sciences Institute
tejas.srinivasan@usc.edu

ABSTRACT

Adapters present a promising solution to the catastrophic forgetting problem in continual learning. However, training independent Adapter modules for every new task misses an opportunity for cross-task knowledge transfer. We propose *Improvise to Initialize* (*I2I*), a continual learning algorithm that initializes Adapters for incoming tasks by distilling knowledge from previously-learned tasks' Adapters. We evaluate *I2I* on CLiMB, a multimodal continual learning benchmark, by conducting experiments on sequences of visual question answering tasks. Adapters trained with *I2I* consistently achieve better task accuracy than independently-trained Adapters, demonstrating that our algorithm facilitates knowledge transfer between task Adapters. *I2I* also results in better cross-task knowledge transfer than the state-of-the-art AdapterFusion without incurring the associated parametric cost. [1]

1 INTRODUCTION

Continual Learning (CL) is a learning setting where a single model must learn incoming tasks sequentially, without access to previous tasks' training data when learning new ones (Chen & Liu, 2018). CL presents models with the dual challenges of effectively transferring knowledge across tasks while mitigating catastrophic forgetting (French, 1999). Learning strategies that finetune the full pre-trained model, suffer from catastrophic forgetting in the CL setting—when learning new tasks, previous task parameters get overwritten, resulting in diminished model performance on older tasks. Further, finetuning pre-trained models on intermediate tasks can harm the model's ability to generalize to new tasks, as the model's parameters diverge further from the pre-trained model checkpoint (Pruksachatkun et al., 2020). While regularization (Kirkpatrick et al., 2017) and replay (Chaudhry et al., 2019) methods can mitigate these issues, none of these are perfect solutions. Existing CL algorithms suffer from the forgetting issue while also failing to effectively utilize knowledge from both the pre-trained model and intermediate task checkpoints.

Adapters (Houlsby et al., 2019) present a promising solution to the catastrophic forgetting problem for Transformer-based CL models. Adapters are bottleneck Multi-Layer Perceptron networks that are trained on tasks by inserting inside a frozen pre-trained Transformer. In the CL setting, we can train a separate Adapter module for each task, allowing models to retain previous tasks' knowledge by keeping the shared pre-trained Transformer frozen. However, independently training Adapter modules for each new task prevents the model from utilizing previously-learned Adapter knowledge. AdapterFusion (Pfeiffer et al., 2021) proposes a two-phase algorithm: *knowledge extraction* by first learning an Adapter module for the new task, and *knowledge composition* by fusing knowledge from multiple task Adapters through an Attention layer. However, adding an AdapterFusion layer to the model for each task adds a large parametric cost, with $\approx 20 - 40\%$ parameter increase over the base Transformer for each task-specific AdapterFusion layer added.

We propose *Improvise to Initialize* (*I2I*), a three-phase CL algorithm that utilizes knowledge from previously-learned task Adapters to learn an initialization for the incoming task's Adapter module. We initially *improvise* on the incoming task by learning an AdapterFusion over the previous tasks' Adapters. We then *initialize* an Adapter for the new task by distilling knowledge from the AdapterFusion trained in the first phase. Finally, we train the initialized Adapter on the new task. By discarding the AdapterFusion after knowledge distillation, we can avoid the parametric cost while still fusing knowledge learned from previously-seen tasks to enable cross-task knolwedge transfer.

We perform experiments on CLiMB (Srinivasan et al., 2022), a multimodal CL framework, by training models on sequences of visual question answering tasks. *I2I* facilitates knowledge transfer between task Adapters, outperforming AdapterFusion on learning new tasks *without incurring the large parametric cost*. To mitigate *I2I*'s training time cost, we experiment with variants that do not require the full training data for the *Improvise* and *Initialize* phases. These variants reduce the training overhead while outperforming independently-trained Adapters and AdapterFusion.

[1]Our code is available at https://github.com/GLAMOR-USC/CLiMB/tree/i2i.

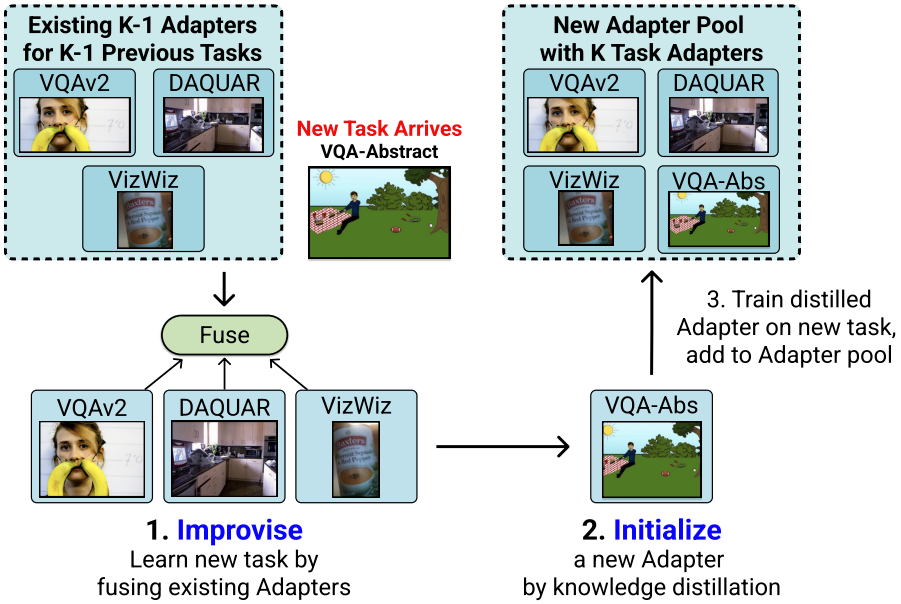

Figure 1: We propose *I2I*: *Improvise to Initialize*, an Adapter-based continual learning algorithm that initializes new task Adapters by first improvising using existing Adapter knowledge.

The primary contributions of this work are as follows:

- We propose *I2I*: *Improvise to Initialize*, an Adapter-based continual learning algorithm that initializes new task Adapters by first improvising using existing Adapter knowledge.
- We perform experiments on sequences of visual question answering tasks in the CLiMB framework, showing that *I2I* outperforms AdapterFusion without requiring additional model parameters for knowledge transfer.
- We analyze each phase of our *I2I* algorithm, and show that improvising from existing Adapters results in better performance on the incoming task.

2 Related Work

We apply *I2I* in a sequential, multimodal task learning setting and use Adapters as the basis for knowledge transfer.

Multimodal Continual Learning Task-incremental continual learning has been primarily studied in unimodal settings (Rebuffi et al., 2017; McCann et al., 2018). Within multimodal learning, various works have explored CL over visual question answering, where CL happens over question types (Greco et al., 2019), visual scenes (Lei et al., 2022) and distinct VQA tasks (Zhang et al., 2022b). CLiMB (Srinivasan et al., 2022) constructs a more general framework for continual learning over vision-and-language tasks. Suhr & Artzi (2022) move beyond vision-and-language into interactive embodied environments, where an instruction-following agent must continually learn from user feedback. These works primarily rely on traditional CL methods such as weight consolidation (Kirkpatrick et al., 2017) or experience replay (Chaudhry et al., 2019) to mitigate forgetting which either compromise learning capacity of the model or require a memory buffer. Our focus is on adoption of Adapters within Transformer models in a CL setting.

Transfer Learning with Adapters Adapters (Houlsby et al., 2019) are task-specific modules inserted within a frozen pre-trained Transformer and enable parameter-efficient fine-tuning. The size of adapters is typically significantly less than the pre-trained Transformer. Applied to continual learning, Adapters can solve the catastrophic forgetting issue by learning a separate set of Adapter parameters for each task. However, training independent Adapter modules for each task prevents knowledge transfer between the task Adapters. One approach to remedy this issue is weight sharing between Adapters. Sung et al. (2022) experiment with *Half-Shared Adapters*, where different task Adapters shared the same weights for the upsampling layer. Zhang et al. (2022a) first identify which Adapter layers from previous tasks can be re-used for the new task, and then learn parameters for the remaining Adapter layers for the new task. Jin et al. (2021) train a Hypernetwork (von Oswald et al., 2020) to generate Adapter weights for different tasks—tasks with

similar representations would generate similar Adapter parameters from the Hypernetwork. AdapterFusion (Pfeiffer et al., 2021) differs from these parameter sharing approaches, by combining representations from multiple task Adapters to improve performance on a target task. Our $I2I$ algorithm is closest in spirit to AdapterFusion, but uses knowledge fusion to learn an initialization for the new task Adapter rather than as a post-hoc transfer learning step.

3 METHODOLOGY

We propose $I2I$: *Improvise to Initialize*, an Adapter-based algorithm that leverages knowledge from already-learned task Adapters when creating Adapters for a new task.

3.1 PRELIMINARIES

We describe the continual learning problem (Section 3.1.1), and how Adapters (Section 3.1.2) solve the catastrophic forgetting problem. AdapterFusion (Section 3.1.3) seeks to solve the lack of cross-task knowledge transfer with traditional vanilla Adapters.

3.1.1 CONTINUAL LEARNING

We consider a *task-incremental* continual learning setting, where a model encounters a sequence of K distinct tasks, $\mathcal{T}_{1...K}$, in order. In this work, the initial model is a pre-trained Transformer $\mathcal{M}_0$. For every task $\mathcal{T}_i$, the model is initialized with the previous checkpoint $\mathcal{M}_{k-1}$, and trained to minimize training loss on the dataset $\mathcal{D}_i$:

$$\mathcal{M}_k \leftarrow \arg\min_{\mathcal{M}} \mathcal{L}(\mathcal{D}_k; \mathcal{M}). \tag{1}$$

After learning task $\mathcal{T}_i$, the model cannot access the training data $\mathcal{D}_k$ or the previous model checkpoints $\mathcal{M}_{0...k-1}$. We consider a *task-aware* continual learning setting, where at test time we know the task identity for every model input.

3.1.2 ADAPTERS

Adapter modules (Houlsby et al., 2019) are Multi-Layer Perceptron layers typically inserted within each layer of the pre-trained Transformer. In general, they amount to $\approx 1\%$ of the Transformer model $\mathcal{M}$ parameters. When training on task $\mathcal{T}$, the Transformer parameters $\mathcal{M}_0$ are kept frozen while the Adapter modules Φ are learned. Some additional task-specific parameters Ψ outside the Transformer may also be learned, such as a classification head for discriminative models. In our model, Ψ is a linear layer that projects visual features before passing into a language model (Figure 3).

In the continual learning setting, we learn Adapter parameters Φ_k and task-specific parameters Ψ_k for every task $\mathcal{T}_k$:

$$\Phi_k, \Psi_k \leftarrow \arg\min_{\Phi,\Psi} \mathcal{L}(\mathcal{D}_k; \mathcal{M}_0, \Phi, \Psi). \tag{2}$$

When learning task $\mathcal{T}_k$, the previous tasks' Adapter modules $\Phi_{1...k-1}$ remain untouched. Since we are operating in a task-aware setting, at inference time we can load the Adapter modules associated with the corresponding inference time task. In that way, even after learning K tasks, the model retains the same performance on the evaluation set of $\mathcal{T}_k$ as when the Adapter module Φ_k was originally trained. In other words, there is no catastrophic forgetting during the continual learning at the cost of adding $\approx K \times 1\%$ Adapter parameters to the base model.

However, by training Adapters independently on distinct datasets, there is no cross-task knowledge transfer. We hypothesize that Adapters can benefit knowledge acquired while learning previous tasks to achieve the best of both worlds: forward knowledge transfer without forgetting past tasks during continual learning.

3.1.3 ADAPTERFUSION

AdapterFusion (Pfeiffer et al., 2021) is a two-phase transfer learning method that composes knowledge from multiple task Adapters to improve model performance on individual tasks. In the first *knowledge extraction* phase, the model learns an Adapter Φ_k for task $\mathcal{T}_k$. In the second *knowledge composition* phase, the model is again trained on task $\mathcal{T}_k$ using a Fusion layer $\mathcal{F}_k$ that combines representations from multiple frozen task Adapters $\Phi_{1...k}$. The Fusion layer $\mathcal{F}_k$ is typically an Attention layer inserted after the Adapter modules Φ within each Transformer layer.

$$\text{Phase 1: } \Phi_k, \Psi_k \leftarrow \arg\min_{\Phi,\Psi} \mathcal{L}(\mathcal{D}_k; \mathcal{M}_0, \Phi, \Psi) \tag{3}$$

$$\text{Phase 2: } \mathcal{F}_k, \Psi_k \leftarrow \arg\min_{\mathcal{F},\Psi} \mathcal{L}(\mathcal{D}_k; \mathcal{M}_0, \mathcal{F}(\Phi_1...\Phi_k), \Psi) \tag{4}$$

$$\mathcal{F}(\Phi_1...\Phi_k) = \texttt{Attn}\big(Q = x, \quad K, V = \Phi_1(x)...\Phi_k(x)\big) \tag{5}$$

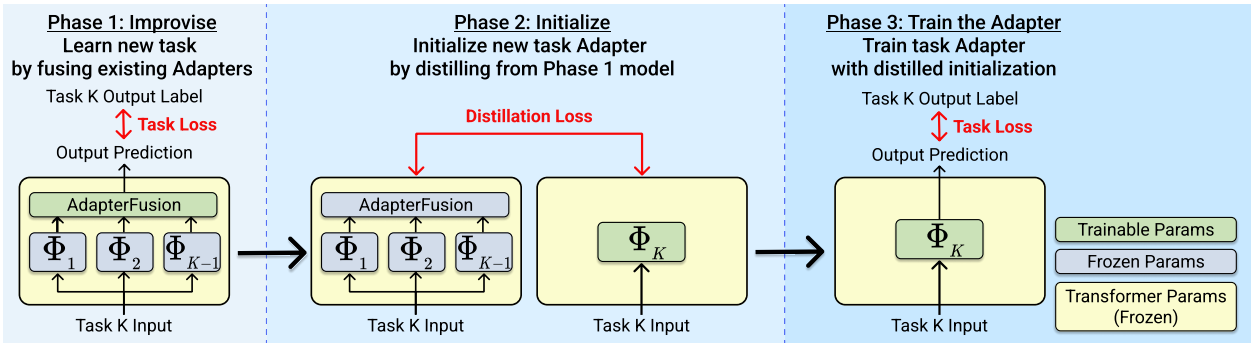

Figure 2: Our proposed *Improvise to Initialize* algorithm. We first *improvise* on a new task $\mathcal{T}_k$ by training a Fusion layer $\mathcal{F}_k(\Phi_1, \Phi_2, ..., \Phi_{k-1})$ that learns to fuse representations from each of the previously-learned Adapters $\Phi_{1...k-1}$. We then *initialize* a new Adapter Φ_k by distilling knowledge from the Fusion layer, and discard the parameters of the much larger Fusion layer. Finally, the distilled Adapter Φ_k is trained on the task $\mathcal{T}_k$.

AdapterFusion facilitates forward transfer learning between different task Adapters, but comes with an exponential cost in parameters with respect to the number of tasks to be learned. Since the Fusion layer consists of Query, Key, and Value matrices, the added parameters from a single task's Fusion layer range from 20-40% of the pre-trained Transformer parameters. As the number of tasks K increases, this added parameter cost will even exceed the pre-trained Transformer size. For instance, with a pre-trained ViLT model, the added AdapterFusion layers will have more parameters than the original ViLT Transformer after learning just 5 tasks in the CL setting considered here.

Additionally, performing the fusion operation as a post-hoc transfer learning step may be sub-optimal. Specifically, after the model has already learned an Adapter Φ_k that has converged to a local minima, it may be difficult to encourage the model to move out of that local minima using knowledge from Adapters $\Phi_{1...k-1}$.

3.2 Improvise to Initialize

We hypothesize that fusing knowledge from existing tasks $\mathcal{T}_{1...k-1}$ to learn an initialization for a new task Adapter Φ_k will yield better knowledge transfer than AdapterFusion's post-hoc knowledge composition. We propose $I2I$: *Improvise to Initialize* (Figure 2), a three-phase training strategy that initially learns a fusion of already-learned Adapters $\Phi_{1...k-1}$, initializes a new task Adapter Φ_k by distilling knowledge from the fusion, and then trains the Adapter Φ_k as usual. We apply the $I2I$ algorithm to learn Adapters Φ_k for $k > 1$. For task $\mathcal{T}_1$, we directly train Adapter Φ_1 on training data $\mathcal{D}_1$.

Phase One: Improvise We first *improvise* on the new task, using knowledge from the already-trained Adapters.

When learning the second task, *i.e.* $k = 2$, we only have one already-learned Adapter Φ_1. We minimize the training loss $\mathcal{L}$ on the dataset $\mathcal{D}_k$ by learning task-specific parameters Ψ_2 using the frozen Adapter Φ_1.

$$\Psi_2 \leftarrow \arg\min_{\Psi} \mathcal{L}(\mathcal{D}_k; \mathcal{M}_0, \Phi_1, \Psi) \tag{6}$$

When combining multiple already-learned Adapters, i.e. $k \geq 3$, we additionally train a Fusion layer $\mathcal{F}_k(\Phi_1...\Phi_{k-1})$.

$$\mathcal{F}_k, \Psi_k \leftarrow \arg\min_{\mathcal{F},\Psi} \mathcal{L}(\mathcal{D}_k; \mathcal{M}_0, \mathcal{F}(\Phi_1...\Phi_{k-1}), \Psi) \tag{7}$$

The only parameters trained in the *Improvise* phase are the Fusion parameters $\mathcal{F}_k$ and the task-specific parameters Ψ_k.

Phase Two: Initialize In Phase Two, we *initialize* a new Adapter Φ_k by distilling knowledge from the model trained in the *Improvise* phase.

For the second task $\mathcal{T}_2$, we can initialize the new Adapter Φ_2 by directly copying the parameters of Adapter Φ_1, and copying the task-specific parameters Ψ_2 learned from the *Improvise* phase.

When initializing the Adapter Φ_k for $k \geq 3$, we use knowledge distillation. The teacher model, $\mathcal{M}_T$, is the previously-learned model with the Fusion layer $\mathcal{F}_k(\Phi_1...\Phi_{k-1})$. The student model, $\mathcal{M}_S$, is a new Adapter Φ_k inserted inside the pre-trained Transformer $\mathcal{M}_0$. The task-specific parameters Ψ_k for the student model are copied from the teacher model.

Task	Image source	Train/Val QA Pairs	Score Metric
VQAv2	MSCOCO images	443k/214k	VQAScore[2]
Visual7W	MSCOCO images	69.8k/28k	Exact Answer Match
VQA-Abstract	Abstract scenes	60k/30k	VQAScore
VizWiz	Images captured by blind people	20k/4.3k	VQAScore
DAQUAR	Indoor household scenes from NYU Depth V2	10k/2.5k	Exact Answer Match

Table 1: We experiment with continual learning over five visual question answering tasks.

During distillation, the student model $\mathcal{M}_S$ is trained to produce the same representations as the frozen teacher model $\mathcal{M}_T$, by minimizing a distillation loss $\mathcal{L}_D$. The only parameters trained during this phase are the student model's task-specific parameters Ψ_k and Adapter Φ_k.

$$\Phi_k, \Psi_K \leftarrow \arg\min_{\Phi,\Psi} \mathcal{L}_D(\mathcal{M}_T(\mathbf{x}), \mathcal{M}_S(\mathbf{x})) \tag{8}$$

After completing the distillation phase, we can discard the Fusion layer $\mathcal{F}_k$, alleviating our model from the parametric growth that AdapterFusion suffers from.

Phase Three: Train the Adapter The Adapter Φ_k is again trained on task $\mathcal{T}_k$, using the Adapter Φ_k and task-specific parameters Ψ_k from the Phase Two student model as the initial checkpoint.

$$\Phi_k, \Psi_k \leftarrow \arg\min_{\Phi} \mathcal{L}(\mathcal{D}_k; \mathcal{M}_0, \Phi, \Psi) \tag{9}$$

Since we discard the Fusion layer $\mathcal{F}_k$, the only added parameters for each task are from the Adapter Φ_k, which are typically $\approx 1\%$ of the full Transformer size. Our $I2I$ algorithm solves the large parametric cost of AdapterFusion, while achieving cross-task knowledge transfer.

3.2.1 Mitigating I2I's Training Time Cost

The $I2I$ algorithm performs three passes over the training data, making our training procedure more time-consuming than standard Adapter training. We propose three variants of our algorithm to mitigate the training time cost:

1. $I2I_{FF}$: The model is trained using the Full training data in both the *Impovise* and *Initialize* phases.
2. $I2I_{FL}$: The model is trained using the Full training data for the *Improvise* phase, but the *Initialize* phase is trained using a Low-shot version of the training data (5% in our experiments).
3. $I2I_{LL}$: The *Improvise* as well as *Initialize* phases are trained using Low-shot versions of the training data.

In all three variants, Phase Three of the algorithm, *i.e.* final training of the Adapter Φ_k on the task data $\mathcal{D}_k$, is performed using the full training data.

4 Experiments

We perform Continual Learning over sequences of visual question answering tasks (Section 4.1) on a CLIP-BART model (Section 4.2). We compare our method against the AdapterFusion baseline (Section 4.3), and evaluate each algorithm's ability to transfer knowledge across Adapters (Section 4.4).

4.1 Continual Learning Tasks

We perform experiments on CLiMB, the Continual Learning in Multimodality Benchmark. The CLiMB benchmark contains a variety of vision-language tasks, including question answering, visual entailment, and vision-language reasoning. We hypothesize that cross-task knowledge transfer is more feasible between similar tasks. For this reason, we extend the CLiMB framework and perform experiments on five visual question answering (VQA) tasks (Table 1): **VQAv2** (Goyal et al., 2017), **Visual7W** (Zhu et al., 2016), **VQA-Abstract** (Antol et al., 2015), **VizWiz** (Gurari et al., 2018), and **DAQUAR** (Malinowski & Fritz, 2014). We evaluate continual learning algorithms on three randomly selected task orders:

[2]https://visualqa.org/evaluation.html

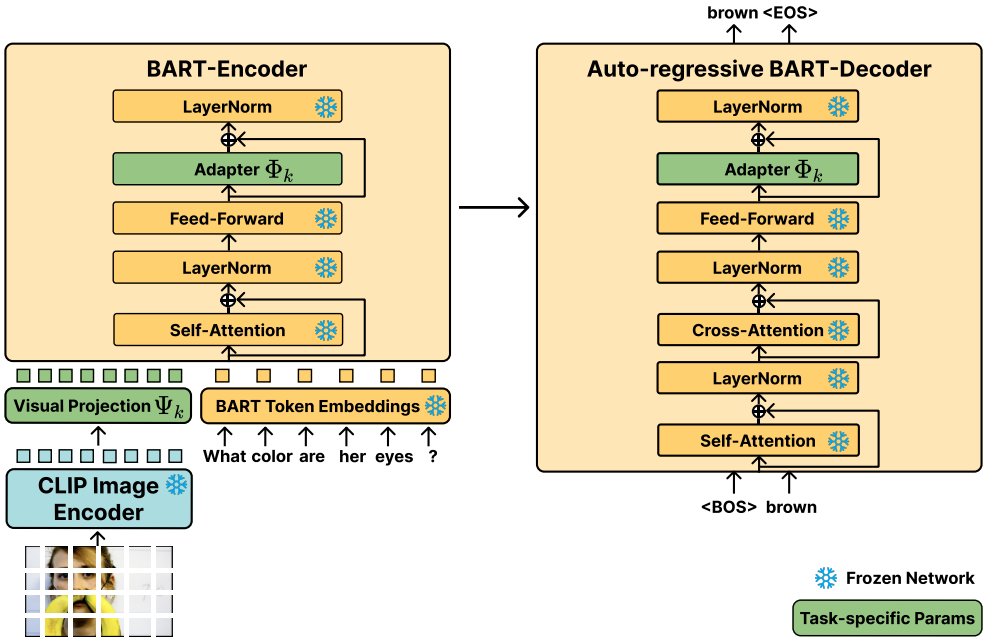

Figure 3: The CLIP-BART architecture. The CLIP image encoder is used to extract image representations, and the BART model uses projected visual features and question token embeddings to generate an answer. For training on task $\mathcal{T}_k$, task-specific parameters Ψ_k and Φ_k are learned while the CLIP and BART parameters are kept frozen.

Method	VQAv2	Visual7W	VQA-Abstract	VizWiz	DAQUAR
Full-Model Finetuning	64.44	25.21	67.55	43.57	25.65
Independently-Trained Adapters	61.42	24.56	63.38	42.04	23.58

Table 2: We report performance of CLIP-BART trained individually on each VQA task, when the full model is fine-tuned and when only Adapter modules are learned. Note that this single-task learning setting is not comparable with the CL setting in Table 3 and represents an upper bound on individual task accuracy without knowledge transfer.

1. VQAv2 → Visual7W → VQA-Abstract → DAQUAR → VizWiz
2. VizWiz → DAQUAR → Visual7W → VQA-Abstract → VQAv2
3. DAQUAR → VQAv2 → VizWiz → Visual7W → VQA-Abstract

4.2 Continual Learning Model

We perform Continual Learning over open-ended QA tasks using CLIP-BART, a generative Vision-Language model.

CLIP-BART Architecture Following previous work (Sung et al., 2022), our CLIP-BART model (Figure 3) combines visual representations from CLIP (Radford et al., 2021) into a text generation BART model (Lewis et al., 2020). Images are encoded using a frozen CLIP-ViT image encoder (Radford et al., 2021) into a set of patch features. A visual linear projection Ψ is applied on the patch features, and the projected image features are concatenated to the sequence of BART token embeddings before passing into a pre-trained BART-base model. BART (Lewis et al., 2020) is a pre-trained encoder-decoder language model. We modify the BART encoder Transformer to jointly encode the image patch features and input token embeddings. The BART Transformer decoder is trained to generate a sequence of tokens corresponding to the answer, by attending over the encoder features. The CLIP-BART model has a total of 227M parameters, with just 139M learnable parameters since the CLIP image encoder is frozen.

CLIP-BART Pre-training Since CLIP-BART is a combination of two separately pre-trained models, we introduce a visual projection layer from CLIP features to BART token inputs tasked with aligning the two pre-trained models for each task. We pre-train CLIP-BART on the MS-COCO training dataset using masked language modeling and

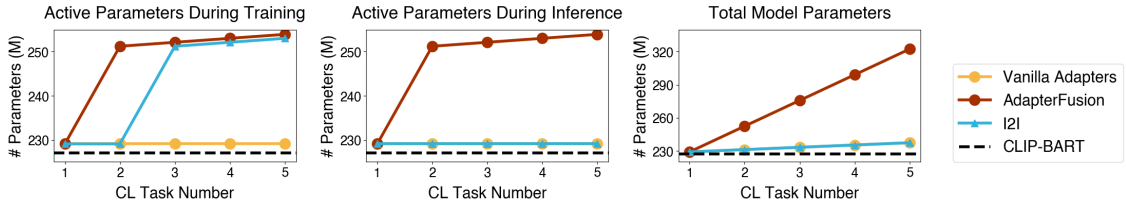

Figure 4: How the number of parameters active during a forward pass during training (left) and inference (center), as well as overall model size (right), increase as each new CL task arrives. Note the backbone CLIP-BART model has 227M parameters.

image-text matching objectives, keeping the CLIP encoder frozen and pre-training only the visual projection layer and the BART model. At the start of continual learning, the model's BART parameters are initialized with the pre-trained checkpoint and remain frozen during continual learning. Additionally, the pre-trained visual projection layer parameters are used to initialize the task-specific visual projection layer Ψ_k for each task $\mathcal{T}_k$.

Training CLIP-BART with Adapters Following Stickland & Murray (2019), we insert Adapter modules after the feed-forward network in every Transformer layer in the BART encoder and decoder—Adapters are not added inside the frozen CLIP image encoder. The Adapter parameters amount to 0.9M parameters per task, which is less than 0.5% of the full CLIP-BART model. We also train a visual projection layer Ψ_k for each task $\mathcal{T}_k$. In Table 2, we compare CLIP-BART fine-tuning with Adapter training on our VQA tasks.

Training CLIP-BART with *I2I* In *Phase One: Improvise* and *Phase Three: Train the Adapter*, we directly optimize the model using the ground-truth answer as supervision, using a cross-entropy loss over the output tokens. In *Phase Two: Initialize*, we train a student model $\mathcal{M}_S$ to produce the same representations as a teacher model $\mathcal{M}_T$ by minimizing a distillation loss $\mathcal{L}_D$. For CLIP-BART, we minimize the MSE loss between the teacher and student models' encoder hidden states h^E and decoder hidden states h^D.

$$h^E_T, h^D_T \leftarrow \mathcal{M}_T(\mathbf{x}) \tag{10}$$

$$h^E_S, h^D_S \leftarrow \mathcal{M}_S(\mathbf{x}) \tag{11}$$

$$\mathcal{L}_D(\mathcal{M}_T(\mathbf{x}), \mathcal{M}_S(\mathbf{x})) = MSE(h^E_T, h^E_S) + MSE(h^D_T, h^D_S) \tag{12}$$

4.3 Baseline Methods

In this work, we do not focus on the catastrophic forgetting problem, and instead aim to facilitate cross-task transfer between Adapters. Therefore, the only CL algorithms that we compare $I2I$ against are independently-trained, *vanilla* Adapters, and AdapterFusion. Figure 4 highlights the impact of adding Adapter parameters for each CL algorithm. We observe that $I2I$ uses a similar number of parameters to AdapterFusion while training, due to the training of the fusion layer $\mathcal{F}_k$ in *Phase One: Improvise*. However, during inference the number of model parameters used is much lower, since the fusion layer $\mathcal{F}_k$ is discarded and only the Adapter parameters Φ_k are added. Further, the overall model size using $I2I$ grows at the same rate as vanilla Adapters, and at a much slower rate than AdapterFusion. Additionally, we introduce a ClosestTaskInit baseline, where each new task's Adapter is initialized using parameters of the most similar already-learned task's Adapter. More details of how the most similar task was selected are available in Appendix A.

We do not compare against other traditional continual learning methods such as Experience Replay (Chaudhry et al., 2019) and EWC (Kirkpatrick et al., 2017), since these primarily target overcoming forgetting. Further, these methods have been shown to hurt the model's ability to generalize to new vision-language tasks (Srinivasan et al., 2022).

4.4 Evaluation Metrics

Our experiments compare learning algorithms on their ability to transfer knowledge between task Adapters. We evaluate knowledge transfer by computing relative improvements for each fusion method over vanilla Adapters. If $S^i_{\mathcal{F}}$ is the score for task $\mathcal{T}_i$ using fusion algorithm $\mathcal{F}$, and $S^i_{\mathcal{A}}$ is the score for the same task using an independently-trained vanilla Adapter, the knowledge transfer for that task, $\mathbb{T}_{\mathcal{K}}(i)$ is computed as

$$\mathbb{T}_{\mathcal{K}}(i) = \frac{S^i_{\mathcal{F}} - S^i_{\mathcal{A}}}{S^i_{\mathcal{A}}} \times 100\%. \tag{13}$$

Method	Individual Task Knowledge Transfer $\mathbb{T}_{\mathcal{K}}(i)$ and Task Score $[S^i](\%)$					Overall Knowledge Transfer, $\tilde{\mathbb{T}}_{\mathcal{K}}$
	VQAv2	Visual7W	VQA-Abstract	VizWiz	DAQUAR	
Vanilla Adapters	[61.42]	[24.56]	[63.38]	[42.04]	[23.58]	0.00%
AdapterFusion	0.16% [61.52]	-5.40% [23.23]	-4.93% [60.26]	3.04% [43.32]	-0.74% [23.41]	-2.08%
ClosestTaskInit	-0.22% [61.28]	1.05% [24.82]	2.90% [65.22]	1.71% [42.76]	1.90% [24.03]	1.90%
$I2I_{LL}$	-0.16% [61.32]	2.78% [25.24]	-0.72% [62.92]	2.97% [43.29]	-0.55% [23.45]	1.17%
$I2I_{FL}$	0.32% [61.62]	2.71% [25.23]	0.66% [63.80]	**4.42% [43.90]**	-0.68% [23.42]	1.94%
$I2I_{FF}$	0.08% [61.47]	**4.22% [25.60]**	**2.96% [65.26]**	3.86% [43.66]	**4.65% [24.68]**	**4.03%**

Table 3: For each CL algorithm, we report the knowledge transfer $\mathbb{T}_{\mathcal{K}}(i)$ and task score $[S^i](\%)$ for every task $\mathcal{T}_i$, averaged across three task orders. We also report overall knowledge transfer $\tilde{\mathbb{T}}_{\mathcal{K}}$, averaged across three task orders.

For a sequence of K tasks $\mathcal{T}_{1\ldots K}$, the overall knowledge transfer $\tilde{\mathbb{T}}_{\mathcal{K}}$ is calculated over all tasks $\mathcal{T}_{2\ldots K}$:

$$\tilde{\mathbb{T}}_{\mathcal{K}} = \frac{\sum_{i=2}^{K} \mathbb{T}_{\mathcal{K}}(i)}{K-1} = \frac{1}{K-1} \sum_{i=2}^{K} \frac{S^i_{\mathcal{F}} - S^i_{\mathcal{A}}}{S^i_{\mathcal{A}}}. \tag{14}$$

Note that task $\mathcal{T}_1$ does not involve fusion, so we do not include it in the overall knowledge transfer calculation.

4.5 Implementation Details

We train our continual learning models with a batch size of 64 on a single 48GB NVIDIA RTX A6000 GPU. When training on a single sequence of five CL tasks: vanilla Adapters take about 30 hours; AdapterFusion takes 57 hours; and I2I$_{LL}$,I2I$_{LF}$ and I2I$_{FF}$ take 49, 63 and 83 hours respectively. Note that at inference time, the $I2I$ methods runtimes are equivalent to vanilla Adapters (Figure 4) while achieving higher performance than the parameter-heavy AdapterFusion method (Table 3).

5 Results and Discussion

In Table 3, we report the knowledge transfer $\mathbb{T}_{\mathcal{K}}(k)$ and task score S^i during continual learning for each task $\mathcal{T}_i$, averaged across three task orders. We also report the overall knowledge transfer $\tilde{\mathbb{T}}_{\mathcal{K}}$, averaged across all task orders.

All variants of our $I2I$ algorithm achieve positive knowledge transfer compared to independently-trained Adapters. Increasing the access to training data during the *Improvise* and *Initialize* phases result in better Adapter initialization and subsequent knowledge transfer. The $I2I_{FF}$ variant, which uses the full training data of each task to fuse existing task Adapters and distill to a new one, achieves an overall knowledge transfer of 4% over vanilla Adapters, averaged over the three task orders.

We also see that our method results in much better knowledge transfer than AdapterFusion. In fact, on average, AdapterFusion results in worse knowledge transfer than independently-trained Adapters. Further, we observe that the ClosestTaskInit baseline also achieves positive knowledge transfer, although not as significantly as the $I2I_{FF}$ variant. These results lend credence to our hypothesis that fusing knowledge from other Adapters is more useful for Adapter initialization, rather than as a post-hoc transfer learning step.

We also see that certain tasks are able to effectively utilize knowledge from previously-learned Adapters than others. Visual7W and VizWiz in particular show positive task transfer across all three variants of our method. On the other hand, VQAv2 has close to zero knowledge transfer across all methods.

6 Analysis

We systematically analyze each phase of the $I2I$ algorithm. We first evaluate how much new tasks benefit from fusing previous task knowledge in *Phase One: Improvise* (Section 6.1), how much knowledge is transferred to the new Adapter during *Phase Two: Initialize* (Section 6.2), and how much each $I2I$ variant benefits from the final Adapter training in *Phase Three: Train the Adapter* (Section 6.2).

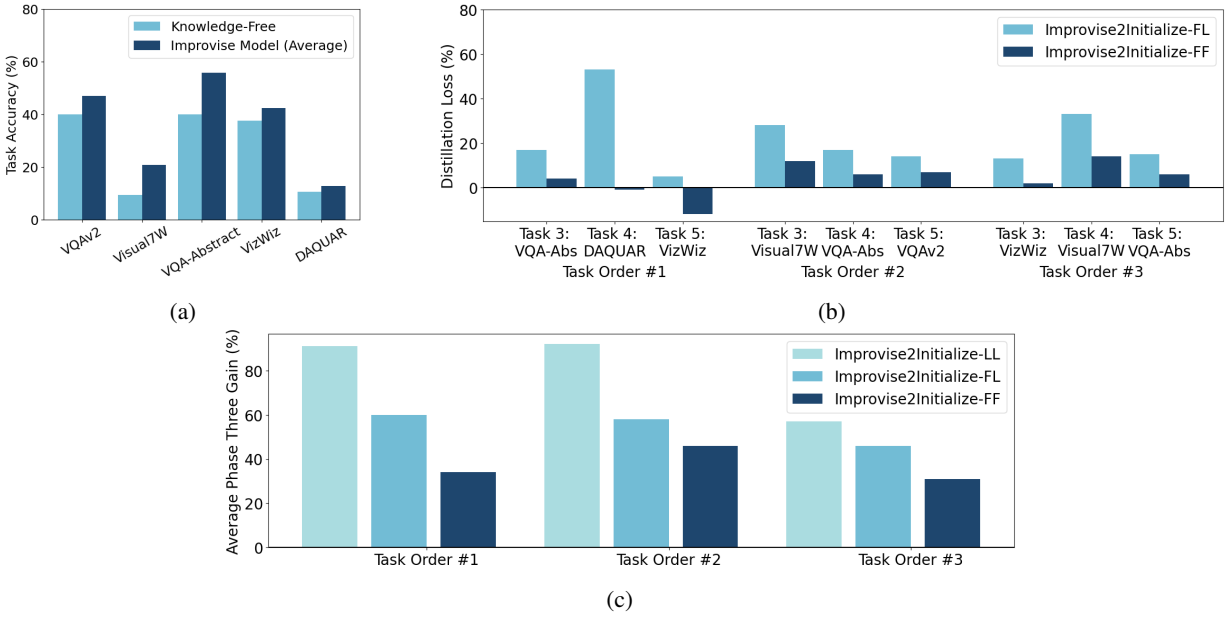

Figure 5: **(a)** Task accuracies for the *Knowledge-Free* model and the *Improvise* models (averaged over all task orders) **(b)** The Distillation Decay for Adapters trained with an *Initialize* phase, using Low-Shot and Full training data ($I2I_{FL}$ and $I2I_{FF}$, respectively). **(c)** The Average Phase Three Gain for each $I2I$ variant, for each of the three task orders.

6.1 How much Knowledge is Gained from Improvising?

We analyze the benefit of fusing existing Adapters in *Phase One: Improvise*. In the *Improvise* phase, we train the task-specific visual projection layer Ψ_k and an AdapterFusion of existing Adapters $\mathcal{F}_k(\Phi_1 ... \Phi_{k-1})$. For each of our visual question answering tasks, we evaluate the *Improvise* model from our $I2I_{FF}$ algorithm in all three task orders (omitting those orders in which the task in question appeared first), and average these scores.

We compare the *Improvise* model with a *Knowledge-Free* baseline. We train a CLIP-BART model on each task, freezing the pre-trained CLIP and BART parameters and only training the visual projection layer Ψ_k. Figure 5a shows that the *Improvise* model results in better task accuracy than the Knowledge-Free baseline, across all tasks. These results reveal that the *Improvise* model is able to utilize knowledge from existing task Adapters to learn new tasks better.

6.2 How much Knowledge is Lost During Distillation?

In *Phase Two: Initialize*, we distill knowledge from the *Improvise* model to a new task Adapter Φ_k by minimizing a loss that trains the new Adapter to produce hidden representations similar to that of the *Improvise* model. However, this distillation process may result in a performance drop from the teacher model to the student Adapter. We measure this performance drop by computing the Distillation Decay for each task $\mathcal{T}_k$, which is the relative decrease in task acurracy from the *Improvise* model ($S_{\mathcal{F}_k}$) to the distilled Adapter (S_{Φ_k}).

$$\text{Distillation Decay for task } \mathcal{T}_k = \frac{S_{\mathcal{F}_k} - S_{\Phi_k}}{S_{\mathcal{F}_k}} \times 100\% \tag{15}$$

In Figure 5b, we compare the Distillation Decay between Adapters that were initialized using the Low-Shot and Full training data ($I2I_{FL}$ and $I2I_{FF}$, respectively). Unsurprisingly, we observe that performing the *Initialize* phase using the full training data results in lower Distillation Decay than only a Low-shot version of the training data (5%).

6.3 How much does the Final Adapter Training Help?

In *Phase Three: Train the Adapter*, the Adapter Φ_k initialized in *Phase Two* is further trained on the training data $\mathcal{D}_k$ of task $\mathcal{T}_k$. We compute the extent to which this training helps by computing the *Average Phase Three Gain*, which is the relative increase in Adapter performance after Phase Three, averaged across tasks $\mathcal{T}_{2...K}$ in a given task order. In

Figure 5c, we compare the Average Phase Three Gain for each $I2I$ variant. We observe that $I2I_{LL}$ benefits the most from the *Phase Three* training, and variants with more exposure to the full training data benefit less.

7 Conclusions and Future Work

We propose *Improvise to Initialize* ($I2I$), a CL algorithm that utilizes knowledge from previously-learned task Adapters to learn an initialization for the incoming task's Adapter module. Our experiments demonstrate that $I2I$ is capable of transferring knowledge between task Adapters for sequences of VQA tasks, outperforming AdapterFusion without incurring the associated large parametric cost.

There are several opportunities for improving the current design of $I2I$. Future work can explore making $I2I$ more training time efficient—the best-performing variant, $I2I_{FF}$, requires three full passes of the training data. The $I2I_{LL}$ variant requires minimal additional training time over normal Adapter training, but struggles to both effectively use existing Adapters in *Phase One: Improvise* and transfer that knowledge to the new task Adapter in *Phase Two: Initialize*.

Similarly, the *FL* variant suffers from high Distillation Decay in *Phase Two: Initialize* (Section 6.2). Future work can explore better distillation methods to reduce this decay, so that the initialized Adapters can more effectively use knowledge from other task Adapters learned in *Phase One: Improvise*.

Finally, our method still trains a separate Adapter module for each task. This form of model expansion can become costly as the number of tasks scales. Future work can explore how distillation-based methods (Ermis et al., 2022) can mitigate the memory cost of model expansion.

Acknowledgments

This work was supported by the Laboratory for Analytic Sciences (LAS), National Security & Special Research Initiatives, and in part by DARPA under contract HR001121C0168.

REFERENCES

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *International Conference on Computer Vision (ICCV)*, 2015.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *International Conference on Learning Representations (ICLR)*, 2019.

Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2018.

Beyza Ermis, Giovanni Zappella, Martin Wistuba, Aditya Rawal, and Cedric Archambeau. Memory efficient continual learning with transformers. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 1999.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

Claudio Greco, Barbara Plank, Raquel Fernández, and Raffaella Bernardi. Psycholinguistics meets continual learning: Measuring catastrophic forgetting in visual question answering. In *Association for Computational Linguistics (ACL)*, 2019.

Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P Bigham. VizWiz Grand Challenge: Answering visual questions from blind people. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning (ICML)*, 2019.

Xisen Jin, Bill Yuchen Lin, Mohammad Rostami, and Xiang Ren. Learn continually, generalize rapidly: Lifelong knowledge accumulation for few-shot learning. In *Findings of Empirical Methods in Natural Language Processing (Findings of EMNLP)*, 2021.

Wonjae Kim, Bokyung Son, and Ildoo Kim. ViLT: Vision-and-language transformer without convolution or region supervision. *International Conference on Machine Learning (ICML)*, 2021.

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 (PNAS)*, 2017.

Stan Weixian Lei, Difei Gao, Jay Zhangjie Wu, Yuxuan Wang, Wei Liu, Mengmi Zhang, and Mike Zheng Shou. Symbolic Replay: Scene graph as prompt for continual learning on vqa task. *AAAI Conference on Artificial Intelligence*, 2022.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Association for Computational Linguistics (ACL)*, 2020.

Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. In *Neural Information Processing Systems (NeurIPS)*, 2014.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv*, 2018.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *European Chapter of the Association for Computational Linguistics (EACL)*, 2021.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel Bowman. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *Association for Computational Linguistics (ACL)*, 2020.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Tejas Srinivasan, Ting-Yun Chang, Leticia Leonor Pinto Alva, Georgios Chochlakis, Mohammad Rostami, and Jesse Thomason. CLiMB: A continual learning benchmark for vision-and-language tasks. In *Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2022.

Asa Cooper Stickland and Iain Murray. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning (ICML)*, 2019.

Alane Suhr and Yoav Artzi. Continual learning for instruction following from realtime feedback. *arXiv preprint arXiv:2212.09710*, 2022.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. VL-Adapter: Parameter-efficient transfer learning for vision-and-language tasks. In *Computer Vision and Pattern Recognition (CVPR)*, 2022.

Johannes von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2020.

Yanzhe Zhang, Xuezhi Wang, and Diyi Yang. Continual sequence generation with adaptive compositional modules. In *Association for Computational Linguistics (ACL)*, 2022a.

Yao Zhang, Haokun Chen, Ahmed Frikha, Yezi Yang, Denis Krompass, Gengyuan Zhang, Jindong Gu, and Volker Tresp. CL-CrossVQA: A continual learning benchmark for cross-domain visual question answering. *arXiv*, 2022b.

Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7W: Grounded Question Answering in Images. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Task $\mathcal{T}_k$ \ Task $\mathcal{T}_j$	VQAv2	Visual7W	VQA-Abs	VizWiz	DAQUAR
VQAv2	-	0.9167	0.9480	0.9877	0.9816
Visual7W	0.9167	-	0.9771	0.8670	0.9280
VQA-Abs	0.9480	0.9771	-	0.9070	0.9541
VizWiz	0.9877	0.8670	0.9070	-	0.9673
DAQUAR	0.9816	0.9820	0.9541	0.9673	-

Table 4: Task similarity score $\text{sim}(h_k, h_j)$ between pairs of tasks $\mathcal{T}_k$ and $\mathcal{T}_j$.

A CLOSESTTASKINIT BASELINE DETAILS

We begin by computing pairwise similarity between each of our continual learning tasks. Although we can define task similarity using several heuristics, we select the previous task whose inputs are most similar to the new task's inputs.

For each task $\mathcal{T}_k$, we encode all training examples using the ViLT vision-language encoder (Kim et al., 2021), and extract the average representation of the [CLS] token, h_k. For each pair of tasks $\mathcal{T}_k$ and $\mathcal{T}_j$, the task similarity is computed as the cosine similarity between their respective task representation vectors h_k and h_j.

$$\text{sim}(\mathcal{T}_k, \mathcal{T}_j) = \cos(h_k, h_j)$$

The task similarity scores for our 5 visual question answering tasks are presented in Table 4. For each newly arriving task $\mathcal{T}_k (k > 1)$, we compute its representation h_k, and then find the most similar task $\mathcal{T}_{k^*}$.

$$k^* = \underset{j \in \{1 \ldots k-1\}}{\arg\max} \ \text{sim}(\mathcal{T}_k, \mathcal{T}_j)$$

We initialize the new Adapter Φ_k by copying the parameters of Adapter Φ_{k^*}, and also copy the task-specific parameters Ψ_{k^*}. We then further train the Adapter Φ_k and task-specific parameters Φ_k on task $\mathcal{T}_k$.