A Broader Impact, Limitations and Future Work

Limitations. In this work, our aim was to create a meaningful benchmark, provide practical guidelines, and offer insights into various multimodal continual pretraining scenarios. We focused on *continual, controlled, minor* model updates. We developed `FoMo-in-Flux` to include many publicly accessible datasets covering a wide range of potential adaptation sub-domains. However, our findings on knowledge accumulation $\mathcal{A}_{KA}$ and zero-shot retention $\mathcal{A}_{ZS}$ are tied to our chosen adaptation and evaluation datasets. Consequently, though unlikely, various sub-domains relevant for future applications might not be sufficiently covered. Additionally, our methods were based off of default hyperparameter ranges from original publications (`LoRA`, `VeRA`, `DoRA`, `BitFit`, `LNFit`, `FS-Merge`, `EMA-Merge`) or continual learning repositories (`mammoth` [17]). While we tested the validity of each method and the chosen hyperparameters to elicit meaningful finetuning responses on respective single datasets (as highlighted *e.g.*, for normal full-finetuning in Tab. 5), it overall means that our conclusions rely on the optimality of these provided hyperparameter ranges.

Broader Impact. Better continual model pretraining and the ability to minimize the need for large-scale model retraining can have significant impact on cost, compute and consequently environmental footprint. By encouraging research into extending the re-usability of large-scale pretrained models before a major continual model update or even full retraining from scratch is needed, we believe our work will lead to more economical and ecological utilization of foundation models. We do not believe that there are any immediate negative societal consequences as a result of this work, but we outline the limitations of our datasets in appendix K.

Future Work. Our benchmark and findings provide a crucial starting point reference for further research into continual multimodal pretraining. We sketch a few important and immediate future research directions:

- **(Meta-) Learning Rate Schedules and Beyond:** Our experiments show the importance of learning rate schedules (and meta-variants) designed for longer horizon continual (minor) model updates. We used a default cosine learning rate schedule and one infinite learning rate schedule (rsqrt), along with five meta-schedule variants, but our results showcase that there is a lot of potential in further exploring infinite schedules, as well as extensions into task- and order-conditioned learning rate schedules to allow for continual model pretraining and model updates.
- **Further Scaling Up Compute and Models:** We studied continual learning under realistic constraints (MAFs), with compute budgets derived from `DataComp-small`. Investigating other computational budgets including over-training, and extending budgets to be potentially task-order dependent could have practical relevance. Extending our insights to even larger model scales (ViT-bigG/14 and beyond) can offer further practical guidance. We have investigated the effect of model and compute scaling (see fig. 5) independently and to a first degree, however we believe there is a lot more exciting future work to be done.
- **Text-to-Image Generative Models:** Besides vision-language representation learning, `FoMo-in-Flux` can be used to study continuous minor updates of text-to-image generative models (such as generative diffusion models) on a fine-grained class and concept level, leveraging its diverse set of captions and information about respective image concepts.
- **Optimal Training Mixtures:** Our results indicate that knowledge retention during minor updates depends heavily on replaying data from previous tasks, guided towards "iid"-fying the learning task. This process helps prevent knowledge forgetting related to pretraining. However, there is room to better understand optimal training mixtures within limited compute budgets. Finding the best ways to allocate FLOPs and memory for replay on large pretraining data is crucial.

B The `FoMo-in-Flux` Benchmark: Additional details.

B.1 Creating our *obscure* datasets

We first query ChatGPT to produce a set of 100 obscure animal names and 100 obscure object names. We then ask ChatGPT to produce diverse prompts for each class name to be used as text

Class: Thayers Gull
Generated Caption:
"A photo of a Thayers Gull, a bird, standing on a rock in a stream of water."

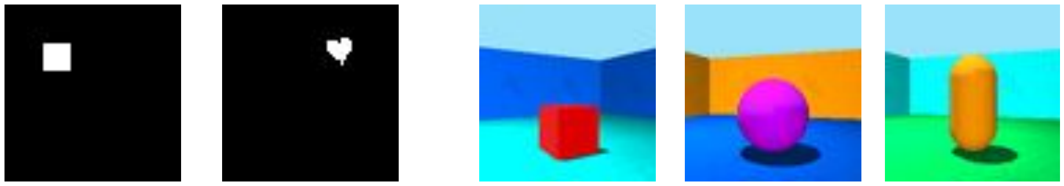
Class: Industrial Area
Generated Caption:
"The photo depicts an industrial area with a cooling tower emitting billowing smoke."

Class: Gray Kingbird
Generated Caption:
"The Gray Kingbird, a small bird, can be seen in a photo standing on top of a tall stem."

Class: Ruin
Generated Caption:
"A photo of a temple built on a hillside in the desert captures the essence of a ruin."

Class: Violet green Swallow
Generated Caption:
"In the clear blue sky, a photo captures the Violet green Swallow perched on a wire."

Figure 7: **Visualisation of generated captions.** We showcase some sample captions generated using our two-stage pipeline for fine-grained classes (birds from Birdsnap [9]), and general, coarse classes (taken from SUN397 [202]). The generated captions combine both image descriptions as well as important semantic class information.



Caption:
"A black-and-white photo of a square located in the top left (exact position: x = 0.097 (0: left, 1: right) and y = 0.097 (0: top, 1: bottom)."

Caption:
"A black-and-white photo of a heart located in the top right: Located at x = 0.839 (with 0 as left, 1 as right) and y = 0.032 (0 top, 1 bottom)."

Caption:
"Captured from a -30.00 degree angle: a red, small cube against a blue wall on a blue flooring."

Caption:
"A pink sphere (larger size), displayed on blue flooring with a orange background, viewed from a 17.14 degree angle."

Caption:
"From the 17.14 degree direction, the orange pill makes a striking impression against a simple backdrop."

Figure 8: **Visualisation of programmatically generated captions** for Shapes3D [19] (*right*) and DSprites [115] (*left*, black and white). Chosen at random, some captions are complete with exact details, while some only have more generic descriptors. Caption style leverages templates generated by GPT-4. The default resolution of these images is 64×64, hence the low-resolution appearance.

prompts to feed into a text-to-image model. We manually reviewed the quality of text prompts for faithfulness to real world contexts, and then used Kandinsky-2.1 [147], Stable Diffusion-2.1 [153], and Dreamlike-PhotoReal [1] text-to-image models to generate images for each classname using the curated text prompts. Finally, for each class we manually cleaned and filtered the images to ensure faithfulness. We conservatively removed an entire class if more than 30% of its images were ambiguous or unfaithful to the class using reference images from Google Images.

B.2 Additional information on `FoMo-in-Flux` datasets.

Tables 2 and 3 highlight the diversity of domains and concepts covered in `FoMo-in-Flux`—ranging from diagrams and paintings, natural high- and low-resolution images, to synthetic and generative images, covering fine-grained and specialized domains, such as remote sensingand medical images. On the language side, concept and classes covered also vary noticeably, with e.g. ArtBench10 built around art-style and artist classification (as reflected in the captions), Quilt-1M introducing medical captions for histopathological image data, or our synthetic *Obscure* datasets introducing rare, fantastical concepts with corresponding captions. Dataset licenses are provided in both tables, all of which permit academic re-use. We provide references to original publications, most of which contain information how to download each dataset. To facilitate reproduction, our codebase comes with automatic download mechanisms for datasets where possible, and manual instructions otherwise. Examples for our generated captions are provided in fig. 7 for natural images., and in fig. 8 for procedural generation. Figure 9 contains examples from our generated *obscure* datasets.

B.3 Pipeline, Compute Budgeting and Data Restrictions - Full Overview.

We illustrate the general `FoMo-in-Flux` training and evaluation pipeline in fig. 1. We start with a model θ_0 trained on a large pretraining dataset $\mathcal{P}$, and an empty buffer $\mathcal{B}$.

Table 2: **Adaptation-only datasets** over various visual and textual domains like diagrams, paintings, natural, synthetic or generative images, remote sensing, art styles, traffic signs or textural data; with datasets from Radford et al. [142] with lower zero-shot performance, common transfer or aggregation benchmark datasets such as DomainNet [129] or VTAB [210] and specialized datasets like MVTec-AD [10].

Dataset	#Train	#Test	#Classes	Domain	License	Captions
Classification-based						
AI2Diagrams [84]	2720	681	15	diagrams	CC BY-SA	generated
ArtBench10 [99]	47531	11883	1870	paintings	Fair Use	generated
Birdsnap [9]	31905	7977	500	finegrained, natural	Unspecified, but academic usage	generated
Cifar100 [93]	50000	10000	100	natural	Unspecified, but academic usage	generated
CLEVR [83]	55931	13983	217	synthetic	CC BY 4.0	generated
CLRS [151]	13525	1475	25	remote sensing	Academic purposes [151]	generated
Country211 [142]	31650	21100	211	natural	various CC	generated
CUB200-2011 [186]	5994	5794	200	finegrained, natural	custom non-commercial	generated
DF20-mini [131]	32724	3637	179	finegrained, natural	custom non-commercial	generated
Dollarstreet [152]	13555	4103	1701	finegrained, natural	CC BY-SA 4.0	generated
Domainnet-Clipart [129]	33525	14604	345	illustrations	custom non-commercial	generated
Domainnet-Infograph [129]	36023	15582	345	diagrams	custom non-commercial	generated
Domainnet-Painting [129]	50416	21850	344	paintings	custom non-commmerical	generated
Domainnet-Sketch [129]	48212	20916	345	sketch	custom non-commercial	generated
Dsprites [115]	75000	25000	27	synthetic	Apache 2.0	procedural
DTD [31]	1880	1880	47	textural	custom non-commercial	generated
FGVCAircraft [110]	3334	3333	100	finegrained, natural	custom non-commercial	generated
Flowers102 [125]	6149	1020	102	finegrained, natural	Unspecified, but academic usage	generated
FRU92 [69]	55814	9200	92	finegrained, natural	Apache 2.0	generated
iNaturalist2021 [79]	125000	25000	2500	finegrained, natural	custom non-commercial	generated
Isicmelanoma [41]	2245	562	7	medical	CC-BY-NC	generated
Mitstates [80]	43002	10751	1959	finegrained, natural	Unspecified, but academic usage	generated
Mtsd [44]	59978	8737	227	finegrained, traffic signs	CC BY-NC-SA 4.0	generated
MVTec-AD (Base) [10]	2903	726	15	high-resolution, industrial	CC BY-NC-SA 4.0	generated
MVTec-AD (Faults) [10]	1380	345	88	high-resolution, industrial	CC BY-NC-SA 4.0	generated
ObjectNet [7]	40134	10000	313	natural	CC BY 4.0	generated
Obscure Animals	17000	4238	74	generative	MIT	custom
Obscure Things	19128	4758	84	generative	MIT	custom
OpenImages [90]	115333	8593	589	natural	Apache 2.0	available
PatternNet [226]	26600	3800	38	remote sensing	custom non-commercial	generated
Places365 [221]	120231	36499	365	natural	custom non-commercial	generated
Plantvillage [75]	43444	10681	38	finegrained, natural	CC0	generated
Quilt-1M [77]	95862	23966	157	medical	Academic purposes	available
Resisc45 [68]	18900	6300	45	remote sensing	Unspecified, but academic usage	generated
Shapes3D [19]	75000	25000	864	synthetic	Apache 2.0	procedural
SnakeCLEF2023 [130]	151031	14117	1599	finegrained, natural	custom non-commercial	generated
SUN397 [202]	15880	19850	397	natural	custom non-commercial	generated
SynthCLIP106 [60]	84800	13886	106	generative	CC BY-NC 4.0	generated
Veg200 [69]	61117	20000	200	finegrained, natural	Apache 2.0	generated
Zappos50k [205]	37829	9458	1847	finegrained, object	custom non-commerical	generated
Retrieval-based						
FSCOCO [30] (avg T2I/I2T R@5)	7105	1777	115	sketch	CC BY-NC 4.0	Available
Total	**1759782**	**453020**	**18449**			

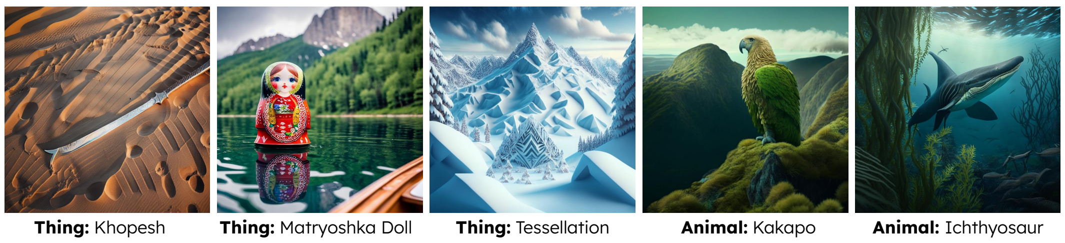


Figure 9: **Examples of our generated obscure things and animals along with captions**, covering 100 rare and uncommonly occurring things and animals. For each class, images are generated using either Kandinsky-2.1 [147], Stable Diffusion 2.1 [153] or Dreamlike-PhotoReal [1].

Continual Pretraining Updates. Within the allocated update budget, at each update step $j \in \{1, 2, \dots, T\}$, the following happens in order:

1. The stream reveals a task update pool of n_j image-text pairs $\mathcal{D}_j = \{(i_k^j, t_k^j)\}_{k=1}^{n_j}$ spanning $\mathcal{C}_j$ concepts.
2. We create the training data mixture $\mathcal{S}_j$ by sampling from the pretraining data $\mathcal{P}$, buffer $\mathcal{B}$, and current task data $\mathcal{D}_j$ with respective ratios $\lambda_{\mathcal{P}}$, $\lambda_{\mathcal{B}}$, and $\lambda_{\mathcal{D}}$, such that $\lambda_{\mathcal{P}} + \lambda_{\mathcal{B}} + \lambda_{\mathcal{D}} = 1$.

Table 3: `FoMo-in-Flux` **Evaluation-only Datasets.** We utilize a subset of standard evaluation datasets used in Radford et al. [142], as well as an array of ImageNet-like variations (including the original ImageNet) to probe different aspect of vision-language understanding and alignment. Moreover, datasets like Food101 [18] or OxfordPets [126] were selected due to their high initial zero-shot performance scores.

Dataset	# Train	# Test	# Classes	Domain	License	Captions
Classification-based						
Caltech101 [94]	6026	2651	101	natural	CC BY 4.0	generated
Caltech256 [55]	21307	9300	257	natural	CC BY 4.0	generated
Cars196 [169]	8144	8041	196	finegrained, natural	custom non-commercial	generated
Cifar10 [91]	50000	10000	10	natural, low-res	Unspecified, but academic usage	generated
Domainnet-Quickdraw [129]	60375	25875	345	sketch	custom non-commercial	generated
EuroSAT [65]	18900	8100	10	Remote Sensing	MIT	generated
FashionMNIST [201]	60000	10000	10	b&w, low-res	MIT	generated
Food101 [18]	75750	25250	101	finegrained, natural	Unspecified, but academic usage	generated
GTSRB [71]	18635	8005	43	traffic signs	CC0	generated
ImageNet [39]	0	50000	1000	natural	custom non-commercial	generated
ImageNet-A [67]	0	7500	200	adversarial, natural	MIT	generated
ImageNet-D [214]	0	4835	103	generative	MIT	generated
ImageNet-R [66]	0	30000	200	renditions (e.g. sketch, paintings)	MIT	generated
ImageNet-S [188]	0	50889	1000	sketch	MIT	generated
ImageNet-V2 [150]	0	10000	1000	natural	MIT	generated
MNIST [40]	60000	10000	10	b&w, low-res	CC BY-SA 3.0	generated
Monkeys10 [2]	1097	272	10	natural	CC0	generated
OxfordPets [126]	3680	3669	37	natural	CC BY-SA 4.0	generated
STL10 [32]	5000	8000	10	natural, low-res	custom non-commercial	generated
SVHN [122]	73257	26032	10	natural, low-res	custom non-commercial	generated
Retrieval-based						
MSCOCO [100] (avg T2I/I2T R@5)	0	5000	0	natural	CC BY 4.0	available
Flickr30k [132] (avg T2I/I2T R@5)	0	1000	0	natural	CC0	available
Total	**462171**	**314419**	**4653**			

If samples in $\mathcal{B}$ are insufficient (particularly at the start of task adaptation), we oversample from $\mathcal{D}_j$, with $\lambda_{\mathcal{D}}$ fixed.

3. We apply a continual update method $\mathcal{M}$ with a fixed compute budget F: $\theta_j = \texttt{train}(\mathcal{M}, \mathcal{D}_j, \theta_{j-1})$. This compute budget F also determines the overall number of update steps conducted.

4. We add samples from the update pool $\mathcal{D}_j$ to the unrestricted buffer $\mathcal{B}$. However, while all samples can be stored in buffer $\mathcal{B}$, they cannot all be sampled for training set $\mathcal{S}$, as the compute budget F imposes an implicit memory restriction [136].

How to Measure Continual Pretraining Computational Cost? To keep our setting practical and ensure a fair comparison, we impose a fixed computation cost budget for each time step to account for the efficiency of each method. However, there is no universally adopted measure of computational cost. Recent works use the number of iterations (forward/backward passes) [136, 49], number of parameters updated [88, 13, 118], FLOPs [50], and time/throughput [118]. However, a single metric does not paint a complete picture of efficiency that is releveant in practice [38, 118].

To account for this, we introduce *Memory-Adjusted-FLOPs (MAFs)*, a novel metric that highlights two aspects most relevant from a practitioner's perspective: the total number of FLOPs per iteration and the maximum utilization of device memory. To compute MAFs, we multiply the FLOPs count of each method by a *memory multiplier*, the ratio of that method's maximum memory utilization to the maximum memory utilization of a full fine-tuning of the base model. The total amount of MAFs for each method and backbone determines the allowed number of update steps each method can take during each adaptation task.

Data Restrictions. We allow unrestricted access to pretraining data (e.g., `LAION-400M` [160]), and an unlimited replay buffer $\mathcal{B}$, as data storage is a negligible contributor to real-world cost [135, 136], and buffer memory is only utilized during the continual pretraining process. To study different retraining data pools, we use four popular image-text pretraining datasets of varying sizes, quality and curation strategies—`LAION-400M` [160], `CC-12M` [23], `CC-3M` [161], and `DataComp-Small` [45].

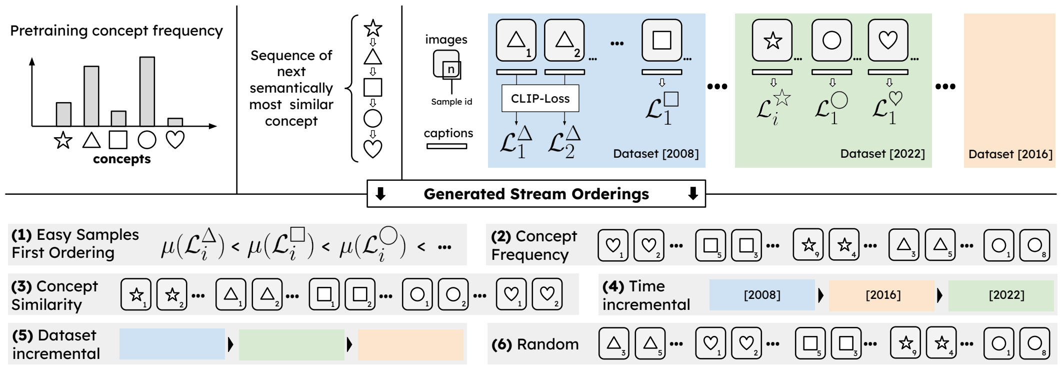


Figure 10: **Pictographic visualization of different data stream orderings** included within the FoMo-in-Flux benchmark setup.

B.4 Designing Data-Centric Task-Sequences

In addition to studying different pretraining sets $\mathcal{P}$ and data mixture ratios $(\lambda_{\mathcal{P}}, \lambda_{\mathcal{B}}, \lambda_{\mathcal{D}})$, we also investigate different realistic orderings by breaking down the FoMo-in-Flux datasets into individual concepts, which are then ordered according to a chosen criterion (including the option to study reverse orderings). This is visualized in Fig. 10. In order to do so, having a controlled set of image-caption pairs is critical, as it allows for well-defined and meaningful arrangement of concepts into sequences according to an ordering $\pi(\mathcal{C})$. Each ordering π divides the set of samples $\mathcal{D}$ into T disjoint subsets $\{\mathcal{D}_1, \ldots, \mathcal{D}_T\}$ of concepts $\mathcal{C}$ sampled without replacement, i.e. $\mathcal{C}_i \bigcap \mathcal{C}_j = \phi, \forall i, j$. We define and motivate six different orderings below:

1. Easy-To-Hard Ordering (performance) is motivated by curriculum learning [58, 154, 165, 170, 208], assuming users deploying their model to easier concepts and usecases first, with incremental movement towards to harder concepts.

Implementation. We approach the notion of “easy” vs. “hard” samples by ordering them according to base model performance. For each concept, we select 50 random image-text pairs and then randomly sample further 50 image-text pairs from the CC-3M dataset to represent random samples from CLIP’s pretraining data pool [29]. For each of the 100 image-text pairs, we compute the sample-wise contrastive loss using a CLIP ViT-L-14 model, and average it over concepts. The lower the mean loss per concept, the easier it is. We then sort all the concepts by their mean loss in ascending order, and consider that to be the data stream ordering.

2. Concept Frequency Ordering (concept-frequency) draws motivation from Udandarao et al. [180], with user requests for model improvement starting from least frequent concepts first (as these constitute edge cases that are most likely to cause undesired performance drops) and incrementally extending to more frequent concepts, which are already represented well in the pretraining pool.

Implementation. We use the *What’s In My Big Data* [43] tool’s elastic search index to search for the frequency of occurrence of each of the class names in the C4 [144] dataset. We compute the frequencies of each of the classes, and order them such that the least frequent concepts (long-tail) occur first and the most frequent ones (head-concepts) are at the end.

3. Concept Similarity Ordering (similarity), inspired by Yıldız et al. [204], is based on the hypothesis that training on conceptually similar tasks allows users to minimize catastrophic forgetting over tasks.

Implementation. To find a *trajectory* with the highest semantic similarity between subsequent concepts, we start with a similarity matrix containing the pairwise similarities between all the class names (via CLIP ViT-L-14 text embeddings of templated text captions of the respective classes). Defining each class as a node in a graph, with weights between the classes being their similarity, the problem reduces to finding the minimum spanning path. We use a simple greedy algorithm: pick a starting class, find its closest neighbour from the remaining set of classes, and keep repeating until we exhaust all classes. We repeat this procedure for every class as a starting point and pick the path with the smallest total weight across all starting classes.

4. Time-incremental Ordering (`time`), inspired by [15, 73, 21, 135, 49], arranges in chronological order.

Implementation. As we only have reliable time information about datasets (via release dates of corresponding publications or the official dataset upload date), concepts are ordered on a dataset-level [15]. These year-level groups are arranged from oldest to most recent, assuming that older datasets are more likely to be conceptually integrated within the pretraining data. Within each year, concepts are randomly ordered. Alongside the above orderings, we compare with two baseline methods popular in continual learning, to better understand the trade-offs made by these data-centric orderings:

5. Dataset-Incremental Ordering (`dataset`) is motivated by [148, 111, 112, 190, 206], but extended to a larger sequence of datasets. To set up `dataset`, we simply randomly sample datasets from Tab. 2 to create a dataset-incremental concept sequence. This sequence is then broken down into the desired number of tasks T.

6. Random Ordering (`random`), a baseline class-incremental ordering widely used across continual learning setups [149, 200, 70, 136], mimics a scenario where user requests for model improvement are unstructured. For this ordering, we simply shuffle class names at random.

B.5 Verifying Downstream Datasets: Finetuning must improve Performance

In order to estimate a reference upper bound on adaptation performance, verify the quality of generated captions, and perform a sanity-check on our training pipeline, we fine-tune CLIP-ViT-B/32 and CLIP-ViT-B/16 individually on each dataset in our training split, as well as all the evaluation-only datasets which come with training samples. We fine-tune the models on each dataset for 10 epochs, with exact results and training details shown in Supp. table 5. For *all datasets*, we find that finetuning a pretrained CLIP model on our generated captions consistently, and in parts very significantly, improves initial zero-shot performance. This showcases the validity of our generated captions, and supports the inclusion of each listed dataset in the `FoMo-in-Flux` benchmark.

C Continual Pretraining: Additional Details to our Method Perspective

C.1 Detailed Method Overview.

In detail, we study several promising directions for continual pretraining of foundation models: *(1) Naive continual finetuning* [49, 136, 76], which has emerged as a dominant approach for major updates on realistic large-scale benchmarks, making it a contender for handling minor updates as well. *(2) Parameter-efficient tuning methods* like `LoRA` [72], which have become a method of choice for minor updates on a smaller scale or for adapting to new tasks with reduced memory requirements [62, 109, 196, 166, 167, 48, 98] through the use of low-rank weight approximations. In a related fashion, recent work by Zhao et al. [219] has shown promise for model finetuning through low-rank approximations on the optimization gradients (`GaLore`). *(3) Parameter-selective tuning methods* such as `BitFit` [8] or `LNFit` [37], which only tune and update particular parameter subsets in the pretrained model such as bias or normalization terms. *(4) Traditional regularization strategies* from continual learning literature [86, 209], which have yielded surprisingly strong performance in recent studies both in parameter [95, 217] and feature space [121], despite being developed and tested in small-scale scenarios where the model is trained from scratch. *(5) Model merging*, which has gained popularity [197, 78, 146] in non-continual learning scenarios as a means to aggregate models tuned across different tasks, and has been studied in some recent [172, 114] and concurrent works [89, 113] as a method to facilitate continual pretraining over longer adaptation periods. All model merging variations are visualized in fig. 16.

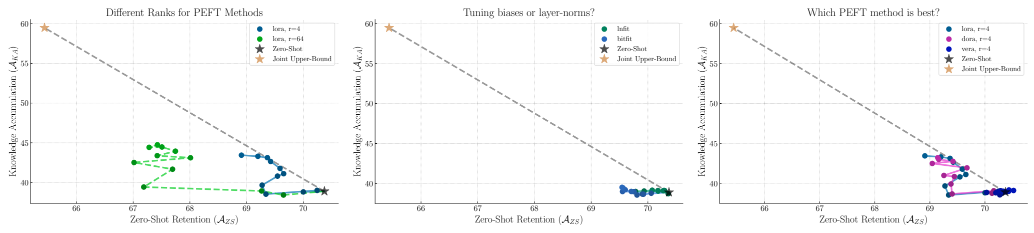


Figure 11: **More Detailed Method Ablations.** (***Left***) Impact of different ranks on continual pretrainability; favouring lower rank values ($r = 4$) over large rank values ($r = 64$) when contrasted against the hypothetical linear tradeoff line between original zero-shot behaviour and performance when finetuned over all data at once. (***Center***) Comparison between parameter-selective `LNFit` [37] and `BitFit` [8]. Both exhibit similar behaviour: strongly limited ability to continuously incorporate new context, with correspondingly minimal deviation in original zero-shot behaviour. (***Right***) Overview of adaptation versus evaluation trajectories for different PEFT methods: `LoRA` [72], `DoRA` [104] and `VeRA` [88]. `LoRA` and `DoRA` behave comparably, with low adaptable parameter counts in `VeRA` heavily limiting the ability to accumulate new knowledge.

C.2 Additional study on parameter-efficient finetuning methods.

For parameter-efficient tuning, the scaling between the accumulation-retention trade-off and the tunable parameter count is also unsurprisingly reflected when adjusting the rank of `LoRA` (fig. 11 left)—though the loss in original generalization performance outweighs the achievable knowledge accumulation when contrasted against the hypothetical trade-off line between initial zero-shot behaviour and joint finetuning.

D Continual Pretraining: Additional Details to General Training Recipes

D.1 On the Influence of Learning Rate Choices for Continual Pretraining.

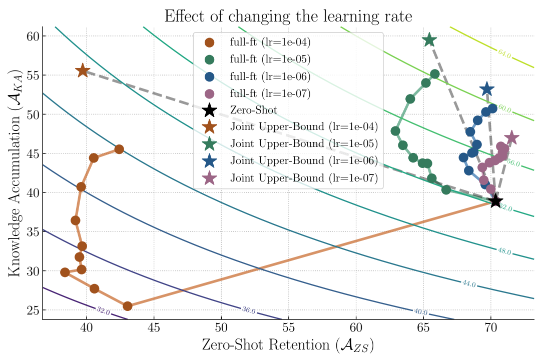


Figure 12: **The effect of the base learning rate on continual pretraining**. The learning trajectory is shown for each value of the learning rate, with the joint training performance as an upper bound. The contour lines show the geometric mean of knowledge accumulation and zero-shot retention ($\sqrt{\mathcal{A}_{KA} \times \mathcal{A}_{ZS}}$). A learning rate of 1e−5 derived from the inital pretraining learning rate achieves the highest final knowledge accumulation and provides the optimal balance between $\mathcal{A}_{KA}$ and $\mathcal{A}_{ZS}$.

To define the learning rate of choice for our continual pretraining problem, we derive it directly from the original pretraining values in Cherti et al. [29] (*1e-3*). We note that the exact peak values are corrected for our practical differences in compute availability (operating on a batch-size of $b_{\text{ours}} = 512$ instead of $b_{\text{openclip}} = 88064$); testing both the commonly utilized linear resizing [53]: $\lambda_{\text{scaled}} = {}^{b_{\text{ours}}}/_{b_{\text{openclip}}} \cdot \lambda_{\text{openclip}}$ and the respective square-root resizing [92] (giving $5.81e-6$ and $7.625e-5$, respectively). In preliminary experiments, we found that rounding up the linearly resized reference (to $\lambda_{\text{scaled}} = 1e-5$) worked slightly better than both options, and provides a much cleaner entry point. As such, we chose to utilize $1e-5$ as our learning rate reference value. As we find in fig. 12, this (mostly) direct re-use of the maximum learning rate has most importantly the highest degree of knowledge accumulation, but also achieves the highest base joint tradeoff with respect to zero-shot retention. Larger learning rates incur significantly higher rates of particularly early-task forgetting, while smaller learning rates limit the amount of knowledge gained. As such, we set $\lambda_{\text{scaled}} = 1e-5$ as our base learning rate.

D.2 Model-specific tuning choices in compute-restricted scenarios

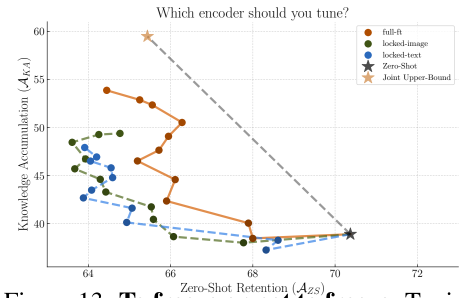


Figure 13: **To freeze or not to freeze.** Tuning both encoders beats single encoder tuning in line with finetuning insights from Goyal et al. [54].

Finally, we highlight the relevance of freezing either image or text encoder in practically compute-restricted continual pretraining in Fig. 13. As freezing either the image or language encoder can allow for significant increases (over a magnitude) in the tuning step budget (as total FLOPs and memory use go down), we find that within the compute-restricted continual multimodal pretraining scenario, tuning both encoders still remains beneficial (aligning with insights provided in Goyal et al. [54] for simple finetuning). While there is negligible difference when freezing each encoder respectively (despite the substantial difference in FLOPs reduction based on tuning the image-encoder alone vs. tuning the text-encoder alone), updating the vision-language model as a joint system incurs a more favorable trade-off between knowledge accumulation and zero-shot retention for each update.

D.3 Softmax Temperatures for Contrastive Losses—*Not Too Hot*!

Recall that CLIP's contrastive loss uses a temperature parameter τ, and it is typically learnable during pretraining. At the beginning of training, it is initialized to 0.07 [142]. Further, to prevent training instabilities, the temperature is clipped to avoid becoming smaller than 0.01. Post training, the learned temperature for all CLIP models considered in this study are found to be exactly 0.01. Moreover, most works that fine-tune a pretrained CLIP model for different downstream tasks, use exactly this learned temperature [54, 177, 178, 197, 42, 78, 61].

Across our main experiments, we follow this standard practice of initializing τ to 0.01 and setting it to be a learnable parameter during continual pretraining. We now explore the impact of different initializations for τ, and sweep over 5 different temperature values, $\{0.01, 0.1, 0.5, 0.75, 1.0\}$. From fig. 14, we observe that τ plays a crucial role for continual pretraining. As we increase the temperature from 0.01 to 0.1, zero-shot retention $\mathcal{A}_{ZS}$ gets impacted by 20% while also noting modest drops on knowledge accumulation $\mathcal{A}_{KA}$, as stability gap issues are excacerbated. Further increasing τ, degrades both A_{ZS} and A_{KA} even more greatly, with the model degenerating to very poor performance. Such drastic changes in model behaviour were also observed in prior work investigating CLIP fine-tuning for downstream tasks [177, 97, 33]—fine-tuning at higher temperatures leads to a decrease in the modality gap between the image and text embedding spaces on the CLIP embedding hypersphere, and hence very quickly degrades the quality of the embedding space for performing downstream tasks [158, 163, 97]. We reproduce and extend the findings of these previous works for the continual pretraining regime, and emphasise the importance of retaining low temperature values for providing optimal $\mathcal{A}_{ZS}$ and $\mathcal{A}_{KA}$.

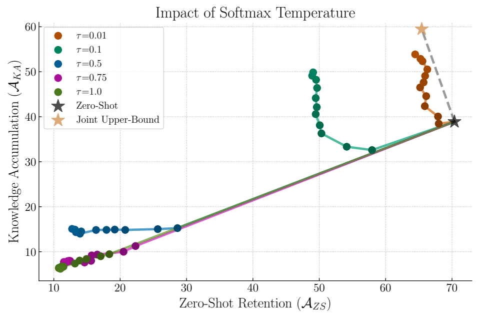


Figure 14: **The softmax temperature for the contrastive loss** is crucial for continual pretraining optimization. The learned temperature after CLIP pretraining is 0.01 (brown trajectory)—higher temperatures than the optimal 0.01 hinder continual pretraining optimization and degrade model weights.

E Continual Pretraining: Additional Details to our Data-Centric Perspective

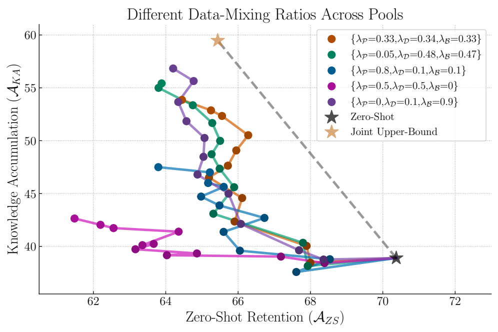
This section extends section 6 with detailed information on data-stream reversals specific data-pool choices and mixing ratios between streaming, buffer and pretraining data ($\mathcal{D}/\mathcal{B}/\mathcal{P}$ and $\lambda_{\mathcal{D}}, \lambda_{\mathcal{B}}, \lambda_{\mathcal{P}}$, respectively, in appendix E.2), and subsampling over the pretraining data for replay.

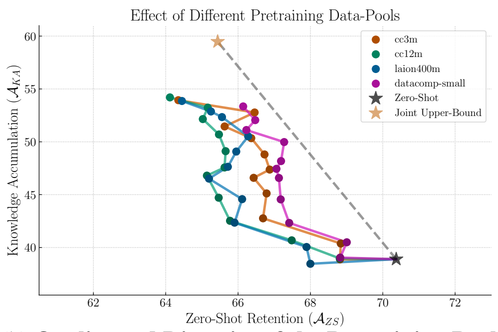
E.1 What Happens if We Reverse Data-Streams?

Each sequence introduced in appendix B.4 introduces its own particular deployment scenario. Naturally, these scenarios may also either occur or be designed to occur in reverse; updating the model for example with hardest examples first, or choosing highly unrelated concepts before honing in on one specific ordering of similar concepts (by reversing `similarity`). These scenarios do not have to be related to their precursors, and can present their own unique update cycle. Evaluating fig. 6 (*right*), `random` remains consistent. The prevalent difference we find in reversing `similarity`; starting with a stream of unrelated concepts (more so than just random subsampling) and then moving towards a stream of more related concepts. Effectively, early task composition becomes forcibly harder. In doing so, the loss in retention along the trajectory comes with increased knowledge accumulation[1].

This allows the trajectory to remain consistent and close to the hypothetical linear trade-off line between the initial zero-shot behavior and the finetuning upper bound - more so even than `random` streams. Both cases however point towards high variation in the presented concepts during each update step being very beneficial for continual pretraining over longer update cycles, especially when trying to retain consistent model behaviour for each update. Still, even when also accounting for the reversed `performance` ordering, end-points converge to comparable end points! We find the only outlier to this to be the reverse `frequency` stream. As head concepts are encountered early, knowledge accumulation is lower, while the controlled placement of long-tailed, rare concepts

[1]By composing harder tasks, batch composition becomes also more difficult, which has been aligned with improved vision-language representation learning in *e.g.*, Zhai et al. [213]. Though by reversing `similarity` in our case, the aggregation of similar concepts towards the end of the stream results in diminished knowledge accumulation towards the end of the sequence.





(a) **Different Data Mixture Ratios** $\lambda_{\mathcal{D}/\mathcal{P}/\mathcal{B}}$ between pretraining $\mathcal{P}$, update $\mathcal{D}$ and buffer pool $\mathcal{B}$ yield significantly different adaptation-retention behaviour.

(b) **Quality and Diversity of the Pretraining Pool** $\mathcal{P}$ can matter significantly for retention of initial zero-shot performance.

Figure 15: **Study on Mixture Ratios and Pretraining Pools.**

towards the end of the update cycle, result in disproportionate forgetting of frequent concepts crucial for achieving and retaining overall accumulation and retention performance.

E.2 Data mixtures inform knowledge accumulation and zero-shot retention

Data control is also reflected in the use of different mixing ratios $\lambda_{\mathcal{P}/\mathcal{D}/\mathcal{B}}$, which we study in Fig. 15a. The particular ratios investigated are motivated as follows (note that the baseline reference ratios we use for all our experiments are $\{\lambda_{\mathcal{P}}=0.33,\lambda_{\mathcal{D}}=0.34,\lambda_{\mathcal{B}}=0.33\}$ (in orange)):

No Buffer $\{\lambda_{\mathcal{P}}=0.5,\lambda_{\mathcal{D}}=0.5,\lambda_{\mathcal{B}}=0\}$ **(in pink)** significantly degrades both accumulation and retention, hampering the $\mathcal{A}_{\text{KA}}$–$\mathcal{A}_{\text{ZS}}$ tradeoffs ($-14\%\mathcal{A}_{\text{KA}}$ and $-2.5\%\mathcal{A}_{\text{ZS}}$ compared to the reference).

Pretrain-heavy $\{\lambda_{\mathcal{P}}=0.8,\lambda_{\mathcal{D}}=0.1,\lambda_{\mathcal{B}}=0.1\}$ **(in blue)** also does not improve over the reference, since at each update step, we input fewer update samples from $\mathcal{D}$, limiting the accumulation capacity.

Ibrahim et al. [76] $\{\lambda_{\mathcal{P}}=0.05,\lambda_{\mathcal{D}}=0.48,\lambda_{\mathcal{B}}=0.47\}$ **(in green)** defines the mixture ratio used in past CPT work operating on LLMs. We reproduce the findings of [76], finding a 5% pretraining replay suffices to provide a better accumulation tradeoff compared to the reference ($+2.2\%\mathcal{A}_{\text{KA}}$ and $-0.3\%\mathcal{A}_{\text{ZS}}$), suggesting that replaying pretraining data is less essential for optimal performance.

IIDify $\{\lambda_{\mathcal{P}}=0,\lambda_{\mathcal{D}}=0.1,\lambda_{\mathcal{B}}=0.9\}$ **(in violet).** Inspired by the previous result of [76], the question arises on the importance of the overall pretraining pool $\mathcal{P}$. Extending findings in Prabhu et al. [136], we jointly also increase the buffer mixing ratio to encourage more IID training distributions at each update step from the full $\mathcal{D}$ and $\mathcal{B}$ pools. Doing so provides the favored tradeoff compared to all the previous mixtures, corroborating findings in [136].

E.3 Choice of pretraining data pool significantly impacts zero-shot retention

While the overall relevance of replay on pretraining data may be smaller than suitable buffer choices, we complete the previous study by investigating the impact of the pretraining data pool $\mathcal{P}$ on the end model. We experiment with three other pretraining data pools of diverse volumes, caption-sources, curation strategies, and quality measurements—`CC-3M` [161], `CC-12M` [23], `DataComp-Small` [45]—beyond our reference pool `LAION-400M`. For a fair comparison, we randomly subsample each pretraining data pool to a total size of 2M samples, and use this subset as our final pretraining pool P. Here too, we use the reference mixture ratio setting of $\{\lambda_{\mathcal{P}}=0.33,\lambda_{\mathcal{D}}=0.34,\lambda_{\mathcal{B}}=0.33\}$. From fig. 15b, it is immediately evident that the choice of the pretraining data pool has a relevant impact on the $\mathcal{A}_{\text{KA}}$–$\mathcal{A}_{\text{ZS}}$ tradeoffs. While adaptation capabilities are barely impacted, using `DataComp-Small` (in pink) yields significantly better zero-shot retention properties, (upto $2.4\%\mathcal{A}_{\text{ZS}}$) gains). We speculate that this could be attributed to the purely English-centric nature of the `CC/LAION` pools compared to the unfiltered `DataComp-Small` which has a significantly higher multilingual and cultural diversity, which has been shown to be beneficial for downstream performance previously [123, 124, 133].

F Method and Schedule Details

In the main paper, we study and reference different methods for their ability to encourage better continual multimodal pretraining on `FoMo-in-Flux`. In this section, we provide details on the methods utilized, alongside information not included in the main text with respect to the utilized learning rate schedules.

F.1 Adaptation Methods

LoRA [72] is the most commonly deployed form of parameter-efficient finetuning based on *Low-rank Adaptation*, which avoids explicitly changing pretrained weights, but instead recommends weight updates to be of the form

$$W' = W_0 + BA$$

with pretrained weights W_0, where B, A are two low-rank matrices, *i.e.*, where $W \in \mathbb{R}^{d\times f}$, $A \in \mathbb{R}^{r\times f}$ and $B \in \mathbb{R}^{d\times r}$. By choosing $r << \min{(d, f)}$, memory requirements during finetuning can be significantly reduced. Moreover, any learned adapter weights can be absorbed into the pretraining weights. Note however that while memory is reduced, total FLOPs for backward **and** forward pass are commonly increased over simple finetuning, as full backpropagation still needs to be conducted, as noted in Mercea et al. [118] and as consequently seen in the final MAFs breakdown (see table 4). By default, LoRA (as well as its subsequent variations VeRA and DoRA, see below) introduces an additional weighting α over the weight update BA, which we set to a constant $\alpha = 1$ [72, 88]; as it only acts as an implicit change in learning rate. As noted in Hu et al. [72], the rank r is the essential hyperparameter to define for optimal changes in behaviour.

VeRA [88] introduces a simple variation over LoRA by randomly initializing and freezing A, B into fixed low-rank projections, and instead learning simple learnable vectors Λ_B and Λ_A such that

$$W' = W_0 + \Lambda_B B \Lambda_A A$$

where $\Lambda_B \in \mathbb{R}^f$ and $\Lambda_A \in \mathbb{R}^r$ (utilizing the same dimensional notation as above). This reduces the total number of tunable parameters significantly (though also mitigating possible adaptation capabilities), but similar to LoRA, does not positively impact FLOPs counts for backward and forward passes together.

DoRA [104] minimally alters LoRA by disentangling norm and directions of the introduced adapter matrices to encourage increased stability, and moving training dynamics of LoRA-style approaches closer to those of simple finetuning. Effectively, this defines the DoRA adaptation step as

$$W' = m \cdot \frac{W_0 + BA}{\|W_0 + BA\|}$$

with magnitude vector $m \in \mathbb{R}^{1\times f}$, where m is initialized as $\|W_0\|_c$, before being jointly updated during finetuning alongside the directional (through normalization) updates induced by B and A.

BitFit [8] introduces parameter-selective model finetuning by only updating bias-terms in the model (and retaining remaining (kernel) weights as frozen). In doing so, changes to the model behaviour are supposed to be kept minimal, will still introducing several degrees of freedom for finetuning. Note however that similar to LoRA, while GPU peak memory is reduced, FLOPs are still high, as backpropagation through the full network still has to occur.

LNFit[37] succeeds in the spirit of BitFit, by recommending to only tune scale and bias parameters in model architectures that leverage LayerNorm [6] layers, showcasing particular success on small continual learning benchmarks.

F.2 Standard Continual Learning Methods

EWC [86] (*Elastic Weight Consolidation*) is a regularization scheme on weight updates initially introduced to tackle rehearsal-free continual learning from scratch. The core motivation behind EWC is the assumption that for each continual task, deviation from "task-optimal" weights learned in preceding tasks should be kept meaningfully minimal. In particular, Kirkpatrick et al. [86] argue that deviation should be individual to each model parameter. Assuming full model weights θ after task t, EWC tries to approximate the curvature in parameter-loss space around θ_t via the *Fisher Information Matrix* $\mathcal{F}^t$. To estimate $\mathcal{F}^t$, several forward and backward passes have to be conducted, with the final regularization during training in task $t+1$ defined as

$$\mathcal{L}_{t+1}^{\text{total}}(\theta) = \mathcal{L}_{t+1}(\theta) - \frac{\lambda}{2}\sum_{k\in|\theta|}\mathcal{F}_k^t(\theta_k - \theta_{t,k})^2$$

with penalty weight λ, loss function for task $t+1$, $\mathcal{L}_{t+1}$, θ_t the weights from the previous task, and k the parameter index. Note that for more than two tasks, $\mathcal{F}$ is commonly estimated through a rolling average, as done in implementation, borrowing from the `mammoth` codebase [17].

SI [209] (*Synaptic Intelligence*) follows a motivation conceptually related to that of EWC, in that parameters defined as more influential (by some measure) are regularized more strongly to minimize change. However, unlike EWC which computes one single point estimate using final parameter values after each task, SI computes importance measures used for regularization along the entire training trajectory. By tracking past and current parameter values, an online importance estimate is computed and incorporated as regularization as follows:

$$\mathcal{L}_{t+1}(\theta) = \mathcal{L}_{t+1}(\theta) + c\cdot\sum_{k\in|\theta|}\left(\sum_{\tau<t}\frac{\omega_k^\tau}{(\Delta_k^\tau)^2+\zeta}\right)\left(\theta_k^t - \theta_k\right)^2.$$

with final task weights θ^t from the previous task. Here, ω_k^{tau} is regarded as the per-parameter contribution to changes in the total loss, approximated as the running sum of the product between gradient $g_k(s) = \frac{\delta\mathcal{L}}{\delta\theta_k}$ and parameter update $\theta'_k(s) = \frac{\delta\theta_k}{\delta s}$ (with within-task update step s). Finally, $\Delta_k^\tau = \theta_k(s^\tau) - \theta_k(s^{\tau-1})$ estimates how much a particular parameter has moved. Alongside a simple regularization term ζ to avoid division by zero, this defines the online importance term in SI.

F.3 Model Merging Methods

`FT-Merge` **[197, 78]** introduces a simply model merging recipe, in which different finetuned variants of a same base pretrained model are linear interpolated (using interpolation coefficient α) into a final, more general new base model. While this was initially not introduced for continual learning / pretraining tasks, this form of interpolation can be naturally extended to our problem scenario. After each task, given an interpolation coefficient $alpha$, we interpolate pre- and post-task weights (θ_{t-1} and θ_t, respectively). These updated weights are then passed to the subsequent task $t+1$. Note that we incorporate the interpolation process into the overall MAF compute budget as well.

`EMA-Merge` **[172]** extends Ilharco et al. [78], but shows how a simple exponential moving average can achieve promising regularization beyond implicit learning rate changes for small, toy-ish continual learning image classification benchmarks. Similar to `FT-Merge`, `EMA-Merge` introduces an interpolation coefficient α, and each interpolation step is account for in the overall compute budget.

`ZS-Merge` operates in a fashion close to both merging methods - with the only differentiating factor being that after each task, interpolation occurs not with respect to preceding model weights, but instead to the initial zero-shot baseline.

G Differentiating Factors: `FoMo-in-Flux` with TiC-CLIP [49] and NEVIS [15]

In this section, we elaborate on the details presented in Table 1 of the main paper. We highlight the distinctive features of our benchmark, `FoMo-in-Flux`, in comparison to two closely related benchmarks: NEVIS and TiC-CLIP.

NEVIS. NEVIS [15], like our work, studies long-horizon continual learning with changing data distributions. However, NEVIS focuses on improving performance in a task-incremental setup, where task separation is based on dataset creation timestamps, and concentrates on performance for the current, ongoing task. In contrast, `FoMo-in-Flux` studies the ability for continual knowledge

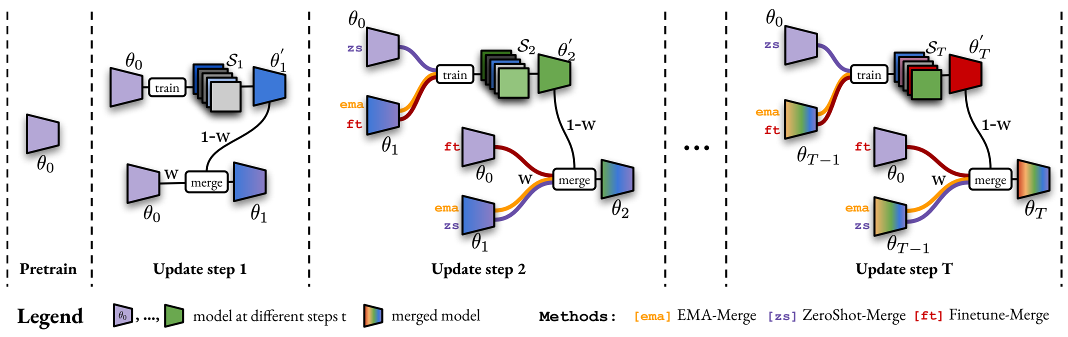

Pretrain
Update step 1
Update step 2
Update step T
train
merge
w
1-w
zs
ema
ft
Legend
model at different steps t
merged model
Methods: [ema] EMA-Merge [zs] ZeroShot-Merge [ft] Finetune-Merge

Figure 16: **Different model merging strategies** explored in this work. We use θ' to denote weights θ finetuned after a respective task. Merging θ_{t-1} and θ'_t then results in the merged outputs weights for task t, θ_t. `EMA-Merge`, or exponential moving average merging, merges previously merged weights θ_{t-1} with current task weights θ'_t produced by tuning the same previously merged θ_{t-1} on task t. `ZeroShot-Merge` always tunes the original pretraining weights θ_0 on each task, then weight-interpolates between the finetuned θ'_t and the previously merged θ_{t-1}. `Finetune-Merge` always interpolates between the original pretraining weights θ_0 and the finetuned weights θ'_t. To arrive at θ'_t, the previously merged model θ_{t-1} is trained on task t.

aggregation, while balancing the retention of good downstream zero-shot performance; measuring open-ended performance in both cases and not limited to a fixed set of classes. We also tackle multimodal vision-language tasks like image-text retrieval, which are more complex to formulate than vision-only tasks. Moreover, `FoMo-in-Flux` allows as to study the impact of different concept and class streams to emulate task orderings that can potentially be encountered when realistically deployed.

TiC-CLIP. The TiC-Datacomp benchmark [49] evaluates the best methods for continual learning over *major* updates, using pretraining budgets similar to those used for pretraining CLIP. In contrast, our work focuses on *minor* updates, utilizing sample and compute scales that are $20\times-100\times$ lower than the corresponding pretraining budgets. Furthermore, TiC-CLIP operates with only six timesteps and uses large, monolithic time-incremental batches of image-text pairs. Our experiments, however, extend up to 200 timesteps and involve four carefully controlled fine-grained data-centric streams across a variety of subdomains, including medical and remote sensing images. Our study provides insights into how models can be pretrained continually over time, in scenarios working with far smaller sample and compute budgets and a larger number of timesteps, ensuring efficiency and scalability across different subdomains. Moreover, we are able to cover and study different data-centric deployment scenarios, alongside a wide array of methods and their trajectority in the knowledge aggregation and retention space. Together, `FoMo-in-Flux` allows us to provide the transitional benchmark towards the much more compute-intensive *major* updates as studied in TiC-Datacomp.

H Additional Experimental Details and Results

H.1 Experimental Setup: Full Overview.

For complete replication, we detail the default models, compute budgets, metrics, training schedules, and data mixtures used here in significantly extended detail here.

Pretrained Models. We conducted our main experiments using a ViT-B-16 CLIP model pretrained on the LAION-2B dataset [159]. We also conducted some additional ablation experiments with a ViT-B-32 CLIP model (to understand the effects of different patch resolution) and ViT-S/16, ViT-L/14, ViT-H/14 and ViT-g/14 models. All our CLIP models are pretrained on LAION-2B, except for the ViT-S/16 model which is pretrained on the DataComp-1B dataset [45].

Default Continual Pretraining Settings. Unless otherwise specified, we always train each continual pretraining method for 20 update steps, T=20 (we test longer sequences with T=$\{50, 200\}$ in Supp. fig. 18). Each update step comprises of continually training a CLIP model for a fixed number of samples derived by the computational budget outlined above. We fix the compute budgets per update step by taking the DataComp-Small total FLOP budget, i.e., 1.8×10^9 GFLOPs and dividing it by the total number of update steps. The exact number of update steps for each method is provided in Supp. Tab. 4. By default, we use a random 2M subset of LAION-400M as our pretraining data pool $\mathcal{P}$ and operate with uniform mixing ratios $\{\lambda_\mathcal{P}{=}0.33, \lambda_\mathcal{D}{=}0.34, \lambda_\mathcal{B}{=}0.33\}$. For our reference upper bound performance, we train a CLIP model initialized from the same open_clip checkpoints jointly on all 41 adaptation datasets (with the samples randomly shuffled). We do this training for a compute budget of $T \times F$ MAFs, equivalent to the overall compute budget available for the entire continual pretraining process.

Training Details. We train all continual pretraining methods with the CLIP contrastive loss [142, 54] and learnable temperature τ, initialized to 0.01 (we provide ablations for the impact of τ initialization in appendix D.3). We select the best-reported hyperparameters for each method from previous literature, only tuning the peak learning rate for each method. We use cosine-decay LR-scheduling with linear warmup of 10% (we study more LR-schedules in section 5.1), with an AdamW optimizer [107], a batch-size of 512 [107], and clip gradients with norm higher than 1. We run all experiments using PyTorch [128]. To truly study updates in both vision and language space, we update both encoders jointly (following Zhai et al. [212], we ablate this choice in appendix D.2). Finally, the exact reflections of MAFs in method updates steps are provided in the supplementary, alongside individual reference scores finetuning CLIP on each dataset individually.

Metrics. From a model updating perspective, there are two main quantities of interest: the degree of *adaptation* to new data and the *retention* of pretraining knowledge. For all experiments, we therefore report two main metrics: Knowledge Accumulation ($\mathcal{A}_{KA}$), the average accuracy (or recall@5 for retrieval) over all concepts in the 41 adaptation datasets, and Zero-Shot Retention ($\mathcal{A}_{ZS}$), the zero-shot transfer accuracy (or recall@5 for retrieval) on the held-out set of 22 datasets.

Plotting Style. In most plots showing our main experimental result, we depict the zero-shot baseline as a black star and the joint training upper-bound as a golden star, with a dotted line connecting the two to approximate the joint training trajectory on the $\mathcal{A}_{KA}$-$\mathcal{A}_{ZS}$ plane. Every other trajectory depicts the training progression of individual experimental runs. Note that these trajectories always begin at the zero-shot baseline (black star).

H.2 Further Replication Details.

We provide our full code, datasets, download pipeline, and experimental results here: github.com/ExplainableML/fomo_in_flux. The provided code covers all relevant details that make up FoMo-in-Flux: All dataset loaders, method implementations, streaming files and all generated captions for every single dataset image (c.f. data_lib/00_info). The code also comes with an automated downloader for preprocessed versions of each utilized dataset.

Compute cluster and run details. For all our experiments, we used a compute cluster with 8×40GB A100 nodes. For most of our ViT-B-16 runs, we used 2 GPUs from these nodes which was sufficient for all our method implementations. To ensure memory efficiency, we optimised our implementations to use CPU offloading for model weights where possible (for *e.g.*, for the EWC, SI and Merge methods). For comparability and reproducibility, all runs and methods share the same seed and equivalent overall experiment setting, with changes in *e.g.*, data stream ordering, modified compute budgets, method or data-mixtures only done when explicited noted.

Justification for CLIP models used. To ensure that our experiments were most relevant to the community, we further verified that the choice of our base CLIP models were validated by practitioner usage. On Huggingface, the open_clip models that were downloaded the most were CLIP ViT-B-32-laion2b (6.11M times), CLIP-ViT-H-14-laion2b (4M times), and CLIP-ViT-B-16 (2M times). Hence, we investigate these models - particularly as ViT-B/16 has been used in other studies on continual major model updates such as Garg et al. [49].

Exact number of update steps, MAFs and samples seen. We provide the full breakdown of how we compute MAFs per time step for each of the methods, and the total compute budget in terms of samples seen per method (in Appendix table 4). We use the datacomp-small [45] compute budgets as our reference. Hence, this means that our total compute budget for the full continual

Table 4: **Compute Budgets used in all ViT-B-16 experiments.** We provide the total number of GFlops taken per task for each of the methods in the `Per-Task GFlops` column. We also showcase the maximum GPU memory requirements for each method in the `Max. Memory Reqd.` column—we convert this into a memory multiplier for each method by dividing with respect to the reference `full-ft` max memory required. Finally, for each method the `Per-Task MAFs` are computed as the product of the `Per-Task GFlops` and the `Memory Multiplier`. Then, we show the total number of gradient update steps that are allowed for these compute budgets per update step t, for the four total number of time step settings, T={20,50,100,200}. Finally, we also show the total number of gradient steps used (`Total Num. steps`) and the total number of samples seen (`Total Num. samples seen`) for the full continual pretraining process—our joint upper-bound oracle also uses this total compute budget.

Method	Per-Task GFlops	Max Memory Reqd.	Memory Multiplier (wrt `full-ft`)	Per-Task MAFs	Num. steps (T=20)	Num. steps (T=50)	Num. steps (T=100)	Num. steps (T=200)	Total Num. steps	Total Num. samples seen
`full-ft`	63394.7585	46.5917	1	63394.7585	1420	568	284	142	28,400	14,540,800
`locked-text`	57254.6183	37.5761	0.8064	46170.1241	1949	780	390	195	39,000	19,968,000
`locked-image`	27176.6698	11.8847	0.2551	6932.7684	12982	5193	2596	1298	259,600	132,915,200
`LNFit`	43165.5968	30.5566	0.6558	28307.9983	3179	1272	636	318	63,600	32,563,200
`BitFit`	43165.5968	30.5546	0.6558	28307.9983	3179	1272	636	318	63,600	32,563,200
`LoRA, r=4`	54479.2515	40.5449	0.8702	47407.8446	1898	759	380	190	38,000	19,456,000
`LoRA, r=64`	54505.0151	40.6757	0.873	47582.8781	1891	757	378	189	37,800	19,353,600
`DoRA, r=4`	54479.8241	40.6582	0.8726	47539.0945	1893	757	379	189	37,800	19,353,600
`DoRA, r=64`	54514.1754	40.7871	0.8754	47721.7091	1886	754	377	189	37,800	19,353,600
`VeRA, r=4`	54479.3393	40.5449	0.8702	47407.921	1898	759	380	190	38,000	19,456,000
`VeRA, r=64`	54507.8336	40.5742	0.8708	47465.4214	1896	758	379	190	38,000	19,456,000
`EWC`	6276081.094	47.207	1.0132	6358925.364	14	6	3	1	200	102,400
`SI`	63394.7585	46.6523	1.0013	63477.1716	1418	567	284	142	28,400	14,540,800
`ZS-Merge`	63394.7585	46.5917	1	63394.7585	1420	568	284	142	28,400	14,540,800
`FT-Merge`	63394.7585	46.5917	1	63394.7585	1420	568	284	142	28,400	14,540,800
`EMA-Merge`	63394.7585	46.5917	1	63394.7585	1420	568	284	142	28,400	14,540,800

pretraining is set to $5.7{\times}10^8$ GFlops for the ViT-B-32 architecture and $1.8{\times}10^9$ GFlops for the ViT-B-16 architecture.[2]

Variance across seeds. To ensure that our results are statistically valid and generalizable, we re-run our canonical continual pretraining experiment with a ViT-B/16 backbone on the 20-task random data stream, with three different seeds. fig. 17 showcases that the three trajectories across the different seeds result in very similar patterns and low variance across runs. This validates that all our main results are generalizable across seeds.

Additional Experiment Results. Finally, we augment our suite of experiments conducted in the main paper.

Fig. 18 provides additional higher-level experiment insights and verification, covering changes in backbone architecture, compute budget and total update steps / task counts. More precisely, Fig. 18 (*left*) shows the impact an increase or decrease in overall compute budget has. As can be seen, all trajectories behave similarly on a qualitative level - experiencing forgetting and stability gap [36] issues at the beginning, before recovering towards the linear zeroshot-finetuning trend line. Comparing end points, we do find that larger compute budgets encourage slightly increased knowledge accumulation gains, but at the cost of disproportionately larger losses in knowledge retention. This means that in practice, large compute budgets may be less favoured even from a performance standpoint to incorporate minor model updates and bridge time between large, major model updates. On top of that, Fig. 18 (*right*) highlights that under a fixed compute budget, in order to bridge time to large model updates, keeping the number of minor model updates small, while maximizing the size of each respective minor update, is preferable from both a knowledge accumulation and retention perspective. Further, we note the strong robustness of model merging even under very long task streams, further strengthening their applicability for long-step continual pretraining.

Fig. 18 (*center*) augments our results on the impact of different data-centric deployment scenarios for continual minor model updates, under a different patch resolution for the vision-transformer. In this experiment, we continually pretrain ViT-B-32 image-encoder models instead of the standard ViT-B-16 image-encoder. We note that the overall trends from this experiment closely match those of

[2]Note that the compute budgets outlined in the original paper [45] were in GMacs—we convert these numbers to GFlops by multiplying by 2 (see here for reference.)

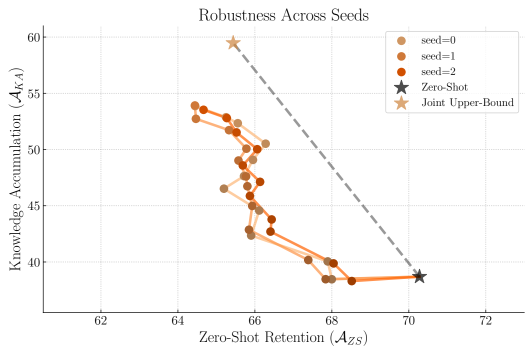


Figure 17: Our continual pretraining insights are robust across different random seeds—the variance in trajectories across three different seeds is minimal.

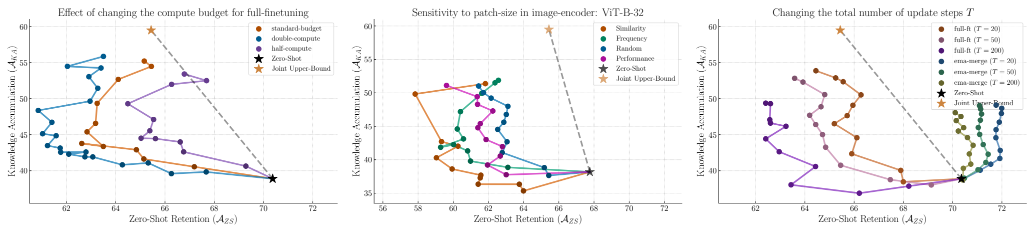


Figure 18: We provide additional experiment insights and verifications, covering changes in backbone architecture, compute budget and update steps. ***(Left)*** Changing the available compute budget noticeably affects knowledge retention, however with limited gains in knowledge accumulation. ***(Center)*** Replacing our default patch-size of 16×16 to 32×32 (*i.e.*, ViT-B-16 to ViT-B-32) for ablating the effect of lower the patch-resolution of our vision-transformer backbones, retains comparable behaviour across different deployment scenarios, with surprisingly similar trajectory endpoints, and comparable accumulation performance. ***(Right)*** Changing the number of tasks the data stream (referred to as update steps T) is divided into, we find drops in both knowledge retention and accumulation. Correspondingly, these results generally recommend to keep the number of minor updates as small as possible, and the respective sizes as large as can be. Note that each trajectory has been uniformly subsampled to visualize the same number of trajectory points for better visual readability. Additionally, note that the robustness of the `EMA-Merge` method extends to longer task streams, reinforcing its potential as a strong approach for continual pretraining.

the original ViT-B-16 experiments (fig. 6), suggesting the overall robustness of our main data-centric results to the patch-resolution of the input images.

I `FoMo-in-Flux`: Datasets

I.1 Finetuning verification

In order to estimate a reference upper bound on adaptation performance, we verify the quality of generated captions, and perform a sanity-check on our training pipeline, we fine-tune ViT-B/32 and ViT-B/16 individually on the datasets in our training split, as well as the evaluation-only datasets which come with training samples. We fine-tune the model on each dataset for 10 epochs with the same learning rate scheduling and the results are shown in table table 5. As can be seen, we find a consistent, and in parts significant improvements conducting CLIP-style training across all individual benchmarks—highlighting the validity of our generated captions, and support for each benchmark to be included in `FoMo-in-Flux`.

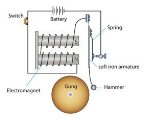
Table 5: Per-dataset fine-tuning results for the ViT-B/32 and ViT-B/16 backbone. FT Performance is the maximum accuracy over 10 epochs. Delta to ZS is the difference between FT Performance and the initial zero-shot accuracy.

Dataset	ViT-B-16		ViT-B-32	
	FT Performance	Delta to ZS	FT Performance	Delta to ZS
Ai2Diagrams [84]	88.00	10.67	83.67	12.33
ArtBench10 [99]	22.86	11.64	21.20	9.08
Birdsnap [9]	63.70	13.30	57.60	10.00
Caltech101 [94]	93.33	1.33	93.67	1.67
Caltech256 [55]	93.97	1.39	92.61	2.61
Cars196 [169]	93.88	5.07	90.56	2.25
Cifar100 [93]	90.33	15.83	91.33	15.93
Cifar10 [91]	99.67	4.67	99.00	4.70
CLEVR [83]	71.05	67.19	55.87	52.94
CLRS [151]	92.67	29.33	91.33	30.00
Country211 [142]	20.38	3.74	20.38	6.11
CUB200 [186]	80.50	10.38	74.00	10.27
DF20mini [131]	50.84	49.46	43.30	41.64
DollarStreet [152]	18.31	11.88	17.96	12.26
DomainNet-Clipart [129]	83.62	3.14	81.74	3.93
DomainNet-Infograph [129]	61.16	3.71	54.93	2.55
DomainNet-Painting [129]	74.64	3.61	71.72	1.47
DomainNet-Quickdraw [129]	66.81	48.45	66.52	48.24
DomainNet-Sketch [129]	78.26	3.94	76.96	4.89
Dsprites [115]	100.00	88.16	100.00	88.36
DTD [31]	68.00	16.00	66.33	11.33
EuroSAT [65]	99.67	43.62	99.33	47.85
FashionMNIST [201]	96.33	16.93	94.67	18.07
FGVCAircraft [110]	48.67	22.24	39.33	14.41
Flowers102 [125]	95.67	21.33	94.67	21.33
Food101 [18]	90.67	5.08	88.00	5.66
FRU92 [69]	91.67	42.97	88.33	39.64
GTSRB [71]	99.33	49.46	100.00	56.12
iNaturalist2021 [79]	50.40	44.76	43.10	37.80
Isicmelanoma [41]	59.33	51.00	56.00	40.33
MITStates [80]	28.30	4.75	26.35	3.02
MNIST [40]	100.00	34.70	99.67	30.57
Monkeys10 [2]	97.79	15.07	96.69	13.97
MTSD [44]	90.97	72.41	90.75	70.93
MVTec-AD (Base) [10]	100.00	27.67	100.00	21.00
MVTec-AD (Faults) [10]	52.33	38.67	38.00	20.67
ObjectNet [7]	54.63	16.75	48.88	16.98
Obscure Animals	89.67	27.49	89.33	33.78
Obscure Things	73.33	17.54	68.67	14.98
OpenImages [90]	58.64	0.00	59.40	0.38
OxfordPets [126]	95.00	4.29	90.67	0.23
PatternNet [226]	99.67	30.72	99.67	34.14
Places365 [221]	48.49	6.62	49.86	7.22
Plantvillage [75]	100.00	80.02	99.67	76.55
Quilt-1M [77]	66.45	65.45	67.10	66.80
Resisc45 [68]	94.33	25.60	93.33	30.16
Shapes3d [68]	100.00	87.16	100.00	85.68
SnakeCLEF2023 [130]	22.17	21.98	16.51	16.45
SUN397 [202]	75.69	6.22	73.93	5.62
STL10 [32]	100.00	3.25	98.67	1.42
SVHN [122]	99.33	46.32	99.00	57.01
SynthClip106 [60]	46.67	5.46	44.00	4.30
VEG200 [69]	84.75	53.90	79.50	46.70
Zappos50k [205]	35.14	22.36	31.29	18.25

J `FoMo-in-Flux`: Caption Pipeline

As part of our `FoMo-in-Flux` pipeline, we converted 63 different classification and retrieval datasets into a format that made them amenable for contrastive language-image pretraining. This entailed providing text captions for each of the images in the classification datasets. For this, our main aims were to ensure: (1) *scalability* of the captioning pipeline, (2) that the captions captured *real-world and fine-grained* details about the image, (3) that the captions were *not verbose* so that they would fit into the context length of CLIP's text encoder (77 tokens), and (4) that the captions *contained the true classname* of each of the images from the classification datasets.

To this end, we proceeded to caption the images in a three-stage manner—(1) We first used a BLIP-2 model [96] using a T5-XL decoder to ensure high captioning performance along with scalability to provide initial seed synthetic captions for each of the images, (2) we next generated templated captions for each of the images using the classnames, for *e.g.*, for an image of a `tench` in the ImageNet dataset, we use a templated caption, "A photo of a tench" and similarly for an image of a `manted howler` in the Monkeys10 dataset, we use a templated caption, "A photo of a mantled howler, a type of monkey.", and finally (3) we merge both the templated and seed synthetic captions using the Capsfusion [207] model—a LLaMA model that is finetuned to take in two captions for an image, and return a merged caption capturing the key aspects of both the captions. Using our three-stage pipeline, we are able to generate diverse yet faithful captions for each of the images in our set of 63 datasets. We showcase a visualisation of our generated captions for some of our constituent datasets in fig. 19.



Dataset: `AI2Diagrams` Class caption: `“A photo of a circuits diagram.”` Synthetic caption: `“the main parts of the model of the circuit are :”` Merged caption: `“The main parts of the model of the circuit are depicted in a photo of a circuit diagram.”`

Dataset: `ArtBench10` Class caption: `“A painting of the artist adam baltatu.”` Synthetic caption: `“two men in a boat sitting on a cliff.”` Merged caption: `“A painting by the artist Adam Baltatu depicts two men sitting in a boat on a cliff.”`

Dataset: `Birdsnap` Class caption: `“A photo of the bird Acadian Flycatcher.”` Synthetic caption: `“small green bird on the branch.”` Merged caption: `“On the branch, there is a small green bird known as the Acadian Flycatcher.”`

Dataset: `Caltech101` Class caption: `“A photo of a camera.”` Synthetic caption: `“the canon eos dslr has a flash and lens attached to the front.”` Merged caption: `“The Canon EOS DSLR, depicted in a photo, features a flash and a lens attached to the front.”`

Dataset: `Cars196` Class caption: `“A photo of a GMC Terrain SUV 2012, a type of car.”` Synthetic caption: `“gmc terrain slt.”` Merged caption: `“The GMC Terrain SUV 2012 is a type of car, specifically the SLT model.”`

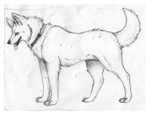
Dataset: `CLRS` Class caption: `“A satellite image of a airport.”` Synthetic caption: `“a google satellite image of the airport and parking lot”` Merged caption: `“A Google satellite image of the airport and parking lot provides a detailed real-world view.”`

Dataset: `Dollarstreet` Class caption: `“A photo of a washing machine from Pakistan.”` Synthetic caption: `“image of washing machine”` Merged caption: `“The image depicts a washing machine from Pakistan.”`

Dataset: `DF20-mini` Class caption: `“A photo of the fungi Agaricus arvensis.”` Synthetic caption: `“the image shows three white and black mushrooms”` Merged caption: `“The image depicts three Agaricus arvensis mushrooms, which are characterized by their white and black coloration.”`

© Can Stock Photo

Dataset: `DomainNet-C` Class caption: `“A clipart of a butterfly.”` Synthetic caption: `“a colorful butterfly is shown on a white background”` Merged caption: `“A colorful butterfly is depicted in a clipart, set against a white background.”`



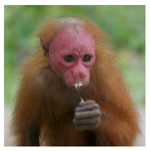
Dataset: `DomainNet-S` Class caption: `“A sketch of a dog.”` Synthetic caption: `“a sketch of an american bulldog standing”` Merged caption: `“A detailed sketch of an American Bulldog standing.”`

Dataset: `DTD` Class caption: `“A photo of a waffled texture.”` Synthetic caption: `“i made chocolate chip banana waffles and ate them with milk”` Merged caption: `“I made chocolate chip banana waffles and enjoyed them with milk, capturing a photo of their delightful waffled texture.”`

Dataset: `FGVCAircraft` Class caption: `“A photo of a EMB-120, a type of aircraft.”` Synthetic caption: `“The airplane is blue and white and sitting on the runway”` Merged caption: `“A photo of an EMB-120, a type of aircraft, shows a blue and white airplane parked on the runway.”`

Dataset: `Flowers102` Class caption: `“A photo of a passion flower, a type of flower.”` Synthetic caption: `“a purple flower with an orange bud”` Merged caption: `“A photo of a passion flower, which is a type of flower characterized by its purple color and orange bud.”`

Dataset: `FRU92` Class caption: `“A photo of a candied date, a type of fruit.”` Synthetic caption: `“nuts in small bag with chinese and chinese chinese written”` Merged caption: `“A photo of a candied date, a type of fruit, along with nuts in a small bag with Chinese characters written on it.”`



Dataset: `Monkeys10` Class caption: `“A photo of a bald uakari, a type of monkey.”` Synthetic caption: `“a monkey holds a piece of food, and is eating it”` Merged caption: `“A photo captures a bald uakari, a type of monkey, holding a piece of food and enjoying a meal.”`

Figure 19: **Random Samples from `FoMo-In-Flux`.** We showcase some sample captions generated using our three-stage pipeline for a few of the datasets in `FoMo-In-Flux`. The `Class caption` is the templated caption using the class-name, `Synthetic caption` is the caption generated using BLIP-2, and the `Merged caption` is the final merged caption using Capsfusion (merging both `Class caption` and `Synthetic caption`).

K Data Statement

Dataset Title: `FoMo-in-Flux`

Dataset Curator(s): N/A

Dataset Version: 1.0

Dataset Citation: N/A

Data Statement Authors: N/A

Data Statement Version: 1.0

Data Statement Citation and DOI: N/A

K.1 Executive Summary

`FoMo-in-Flux` is an aggregate benchmark comprising over 2.53M images from 63 classification and retrieval datasets, including 61 existing datasets and 2 newly introduced ones, described in appendix B. On top of image and labels provided by the original datasets, we provide a caption for each image, generated using the pipeline described in appendix J.

K.2 Curation Rationale

`Fomo-in-Flux` is a benchmark for continual multimodal pretraining that emphasizes adaptation across distinct subdomains over long time horizons, while allowing for finegrained controllability of particular concepts and classes presented at respective update steps for a data-centric perspective on continual multimodal pretraining. The constituent datasets were selected based on availability, licensing, quality of labels, diversity of data domains, quality of the resulting captions, and the degree of adoption in the computer vision and machine learning research communities.

K.3 Documentation for Source Datasets

The licensing information for source datasets, as well as relevant citations, are provided in table 2 and table 3. We release the captions, as well as the Obscure Animals and Obscure Things datasets under the MIT license (https://opensource.org/license/mit).

K.4 Language Varieties

All the class labels and captions are in English.

K.5 Speaker Demographic

N/A

K.6 Annotator Demographic

The captions were created using an automated pipeline and based on original class labels, as outlined in appendix J. For selected simpler datasets, we use the templated captions directly, as shown in table 2 and table 3. For the information about annotators of source datasets, please see the references in table 2 and table 3.

K.7 Speech Situation and Text Characteristics

N/A

K.8 Preprocessing and Data Formatting

The class labels are used as-is with no modification. All images are resized to 224x224 pixels.

K.9 Capture Quality

N/A

K.10 Limitations

Although great care was taken to ensure the correctness of the dataset and random samples of the captions were manually inspected for a quality check, we did not verify the captions for all 2.53M samples. Given the dependence on BLIP-2 [96] and Capsfusion [207], the captions might reflect the biases and idiosyncracies of these models.

Moreover, as an aggregate benchmark, `Fomo-in-Flux` reflects the data collection and annotation biases of the source datasets. However, by pooling diverse sources of data, we avoid a systematic dataset-wide curation bias.

K.11 Broad Impact

Our dataset helps assess the continual multimodal pretraining performance across various methods, data stream orderings, learning rate schedulers, and compute budgets. The insights gained will help optimize continual pretraining, facilitating fewer large-scale model updates. This optimization, in turn, will help decrease energy consumption and lower carbon emissions associated with continual adaptation of foundation models, and overall encourage a more economical and ecological treatment of these large architectures.

K.12 Metadata

License: `https://opensource.org/license/mit`

Annotation Guidelines: N/A

Annotation Process: Automatic

Dataset Quality Metrics: N/A

Errata: N/A

K.13 Disclosures and Ethical Review

N/A

K.14 Other

N/A

K.15 Glossary

N/A

About this data statement

A data statement is a characterization of a dataset that provides context to allow developers and users to better understand how experimental results might generalize, how software might be appropriately deployed, and what biases might be reflected in systems built on the software.

This data statement was written based on the template for the Data Statements Version 2 Schema. The template was prepared by Angelina McMillan-Major, Emily M. Bender, and Batya Friedman and can be found at `http://techpolicylab.uw.edu/data-statements`.

L Paper Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We ensure that the main claims accurately reflect the contributions.
   (b) Did you describe the limitations of your work? [Yes] Provided in the Supplementary Material.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Provided in Supplementary Material.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We ensure that our paper conforms to ethics review guidelines.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Supplementary material for details.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See H.1 for details.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] We fix the random seed in our experiments for fairness and report error bars in the supplementary material to demonstrate that randomness in performance is minimal.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Supplementary Material for details.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We appropriately cited existing resources including links. See Supplementary Material.
   (b) Did you mention the license of the assets? [Yes] We provide licenses for all original datasets whenever possible. See Supplementary Material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We have included scripts to regenerate it alongside our generated captions in Croissant format in Supplementary Material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use existing publically accessible datasets for images, and merely recaption them using them off-the-shelf captioning models.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Yes, we ran checks to ensure no PII or offensive content was added during our curation process.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We have no human experiments.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We have no human experiments.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We have no human experiments.

We provide additional details including datasheet for our dataset, along with our benchmark provided in Croissant format, reproducible scripts for all of our experiments and our visualization interface in the supplementary material.