A. Plotting Style

Across TIME, we utilize a common plotting style to visualize our results—with three base subplots (see for *e.g.*, Fig. 7):

- Knowledge Accumulation ($\mathcal{A}_{KA}$) versus number of tasks over time. In this plot, a gray star indicates the base-weight zero-shot performance on adaptation datasets (see Sec. 4.1 for more details). An orange star indicates an upper bound achieved through jointly training on all the data at once, with no separation over time.
- Zero-Shot Retention ($\mathcal{A}_{ZS}$) versus number of tasks over time. Similar to $\mathcal{A}_{KA}$ versus tasks, this plot visualizes merging results for TIME-variants, but measuring performance on withheld evaluation datasets (cf. Sec. 4.1). Again, gray and orange star indicate base and joint training lower and upper bounds, respectively.
- Finally, we also aggregate both previous plots into one showcasing the progression of merged performance geometric mean $\sqrt{\mathcal{A}_{ZS} \times \mathcal{A}_{KA}}$ over time; utilizing the same star indication as in the previous subplots.

The only deviation from this plotting style is Fig. 7. The left panel visualizes the trajectory across tasks in the $\mathcal{A}_{KA}$ - $\mathcal{A}_{ZS}$ space. Here, full-colored stars reference base model performance and hollow stars the corresponding joint training upper bounds. The right panel shows the geometric mean of $\mathcal{A}_{KA}$ and $\mathcal{A}_{ZS}$ at the end of the last task for different compute budgets.

Finally, several plots such as Figs. 3, 4 and 6 show the extensive scale of our experiments through background visualizations of sub-optimal hyperparameter choices in lighter colors (as opposed to the optimal choices using darker coloring). This plotting style is loosely inspired by Beyer et al. [4].

B. Experiments with Tasks as Datasets

In the main text, we presented all results using a data stream that randomly mixes concepts from different datasets into a coherent set of tasks—following the *random* data-stream in Roth et al. [68]. Here, we relax this constraint and re-run our experiments using individual datasets as tasks, consistent with the standard model merging literature [24, 25, 89]. Specifically, we use the *dataset-incremental* stream from Roth et al. [68]. Even in this setup, we reproduce our main findings. In Fig. 8, we confirm the results from Fig. 3, showing that all offline merging techniques perform poorly when exposed to the axis of time, failing to even match the performance of a simple continual fine-tuning *replay* baseline. Additionally, in Fig. 9, we corroborate the results from Fig. 5, demonstrating that the *best-in*-TIME method remains the most effective temporal model merging approach. We also confirm that the choice of model merging technique is far less critical for temporal model merging than the initialization and deployment strategies.

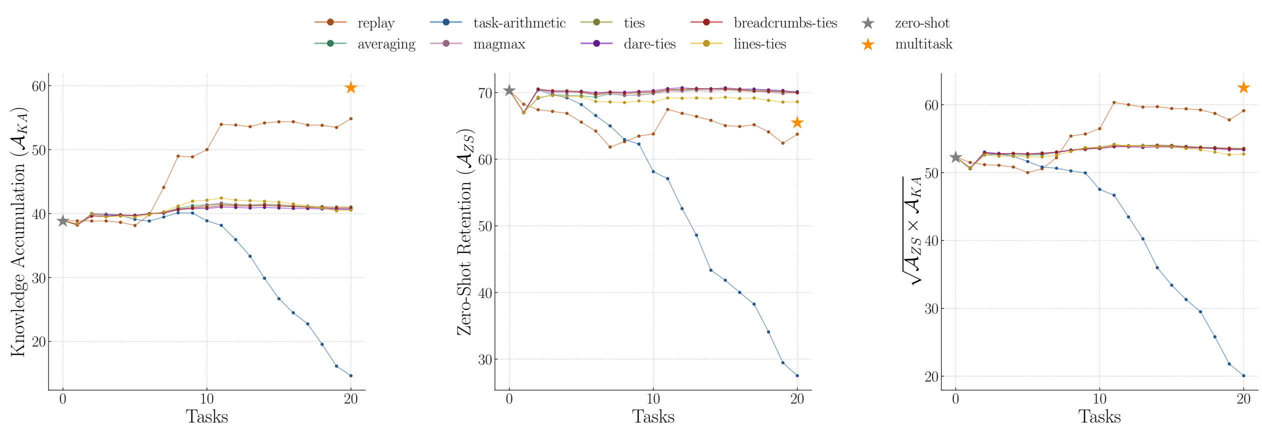

Figure 8. **Offline merging techniques still struggle in the tasks-as-datasets setting.** Switching from the *random* data-stream (Fig. 3 in the main paper) to the *dataset-incremental* stream, which aligns more closely with the standard multi-task merging literature setups, reveals that offline merging techniques still severely underperform compared to the simple *replay* baseline.

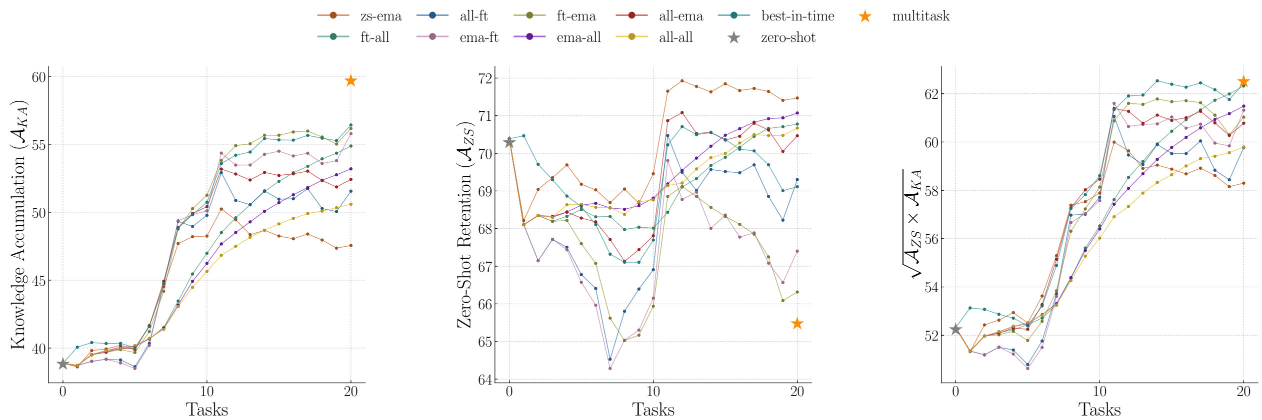

Figure 9. **Dataset-Incremental TIME Exploration.** We replicate the results from Fig. 5 using the dataset-incremental stream instead of the random stream. The main takeaways remain unchanged: initialization and deployment strategies primarily determine temporal merging performance, and the EMA-averaging initialization and deployment strategy utilized in *Best-in-*TIME is the best approach.

C. Experiments with Longer Task Sequences

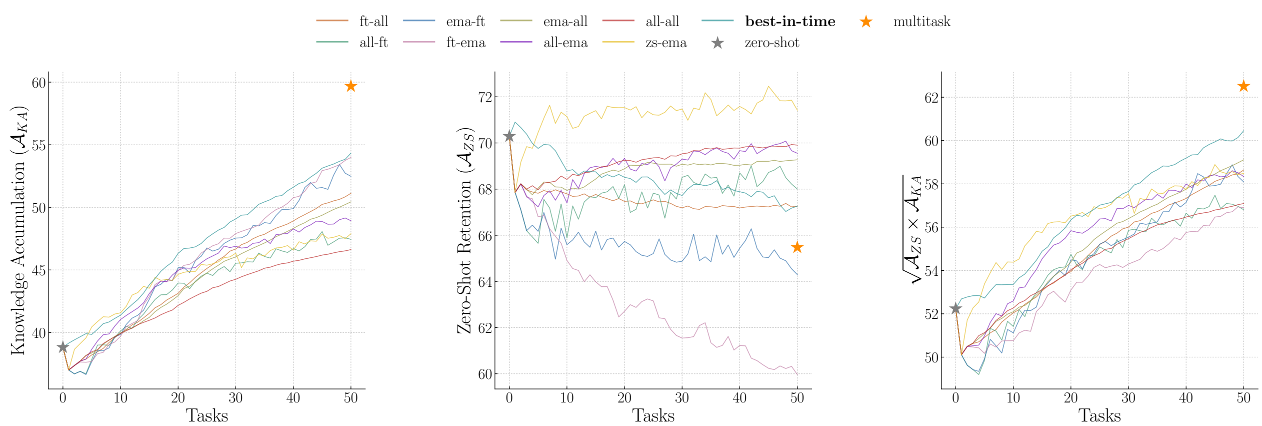

Figure 10. **A long journey through TIME.** We compare all valid combinations of initialization and deployment protocols on a longer sequence of 50 tasks. *Best-in-*TIME remains the best in balancing knowledge accumulation and zero-shot retention.

To test the robustness of our findings in Sec. 4.4, we repeat the experiment shown in Fig. 5 on a longer sequence with the number of tasks $T = 50$ (Fig. 10). For 50 tasks, *Best-in-*TIME still strikes the optimal balance between knowledge accumulation and zero-shot retention. One notable difference with respect to Fig. 5 is the large initial advantage of the zero-shot initialization strategy combined with the EMA deployment strategy. When the learning horizon is further extended to 100 tasks, this initial advantage is maintained, establishing the zero-shot initialization approach as the best-performing method, as shown in Fig. 11. Although the double EMA variant surpasses zero-shot initialization in knowledge accumulation, its poor retention relegates it to third place on the combined metric. In this exploration we re-use the optimal interpolation weight from the 20 task scenario, which may no longer be ideal for longer horizons, as it directly influences the balance between knowledge accumulation and zero-shot retention.

D. Non-Uniform Weighting Schemes for Improving Offline Merging

D.1. Details

In Sec. 4.3, we showed that recency-biased non-uniform weighting helps to improve offline merging performance, when used in conjunction with replaying old task data. Typically, when several models are merged using simple weight averaging, they are uniformly averaged. However, this clearly ignores the dimension of time, assuming all previous task-checkpoints as

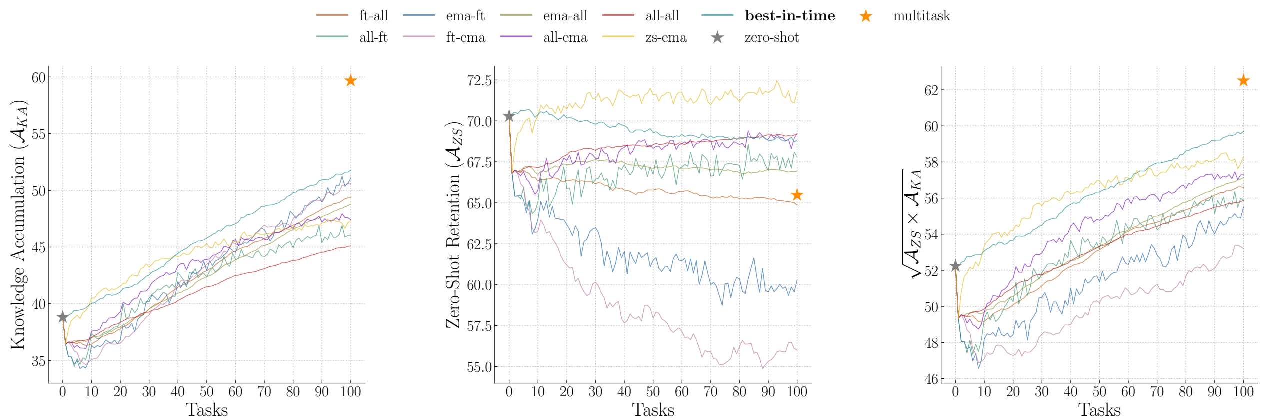

Figure 11. **An even longer journey through TIME.** We compare all valid combinations of initialization and deployment protocols on a longer sequence of 100 tasks. *Best-in-*TIME still remains the best approach balancing knowledge accumulation and retention, measured as the geometric mean of the two metrics in the right-most figure.

independent and agnostic of time. Hence, we explored 8 non-uniform recency-biased schemes including *linear*, *quadractic*, *sqrt*, *cubic*, *fifth-power*, *tenth-power*, *exponential*, *log*.

```

'''
Each weighting scheme below produces a list of N values, at each task N. The ith element of the output list denotes
    the weight coefficient of the ith task checkpoint.
'''

        def linearly_increasing_list(n):
            values = np.linspace(1, n, n)
            return normalize(values)

        def sqrt_scaling_list(n):
            values = np.array([np.sqrt(i) for i in range(1, n + 1)], dtype=float)
            return normalize(values)

        def quadratic_scaling_list(n):
            values = np.array([i**2 for i in range(1, n + 1)], dtype=float)
            return normalize(values)

        def cubic_scaling_list(n):
            values = np.array([i**3 for i in range(1, n + 1)], dtype=float)
            return normalize(values)

        def fifth_power_scaling_list(n):
            values = np.array([i**5 for i in range(1, n + 1)], dtype=float)
            return normalize(values)

        def tenth_power_scaling_list(n):
            values = np.array([i**10 for i in range(1, n + 1)], dtype=float)
            return normalize(values)

        def exponentially_increasing_list(n, base=2):
            values = np.array([base**i for i in range(n)], dtype=float)
            return normalize(values)

        def logarithmic_scaling_list(n):
            values = np.array([np.log(i + 1) for i in range(1, n + 1)], dtype=float)
            return normalize(values)

        def normalize(v):
            v /= v.sum()
            return v.tolist()
```

Listing 1. Recency-biased Non-uniform Weighting Algorithms

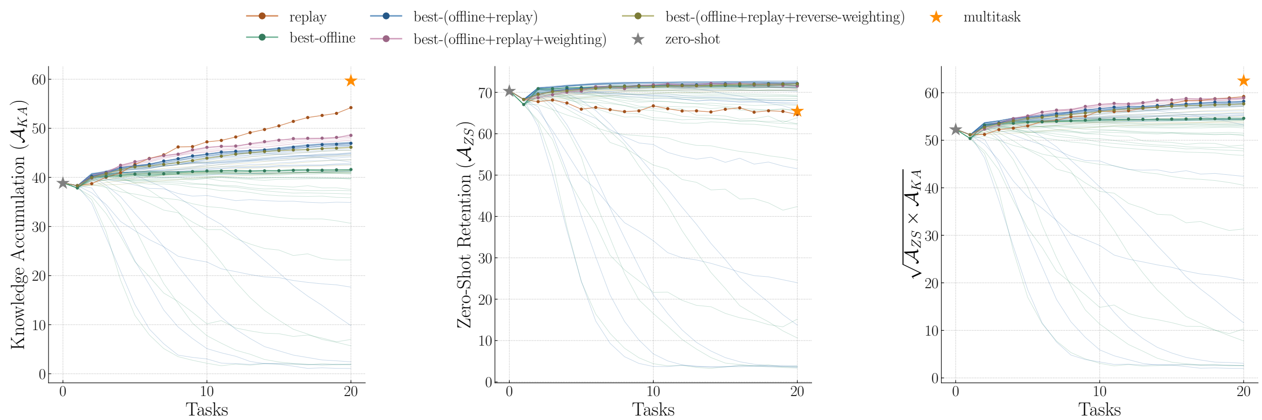

Figure 12. **Effect of reverse-weighting for offline merging techniques.** We find that reversing the weighting scheme that yielded consistent boosts from Fig. 4 is sub-optimal—indeed, it performs worse than the offline merging with replay methods.

D.2. Reversed Non-Uniform Weighting Schemes

In Fig. 4, we found that a simple yet effective method for boosting the performance of offline merging methods is recency-biased non-uniform weighting, i.e. giving larger weights to more recent checkpoints while merging. Here, we ask the question—what if we reversed the weighting schemes such that we give larger weights to older task checkpoints? From Fig. 12, we indeed observe that such a reverse strategy performs worse than the best recency-biased weighting schemes, since the knowledge accumulation ability is hampered by giving more emphasis to older tasks. However, note that such a sub-optimal reverse weighting strategy is still better than the pure offline merging strategy with *no replay*. This helps further ablate the exact importance of *replay* and *non-uniform weighting* for improving pure offline-merging techniques in the presence of the time axis.

E. Variance Analysis across Runs

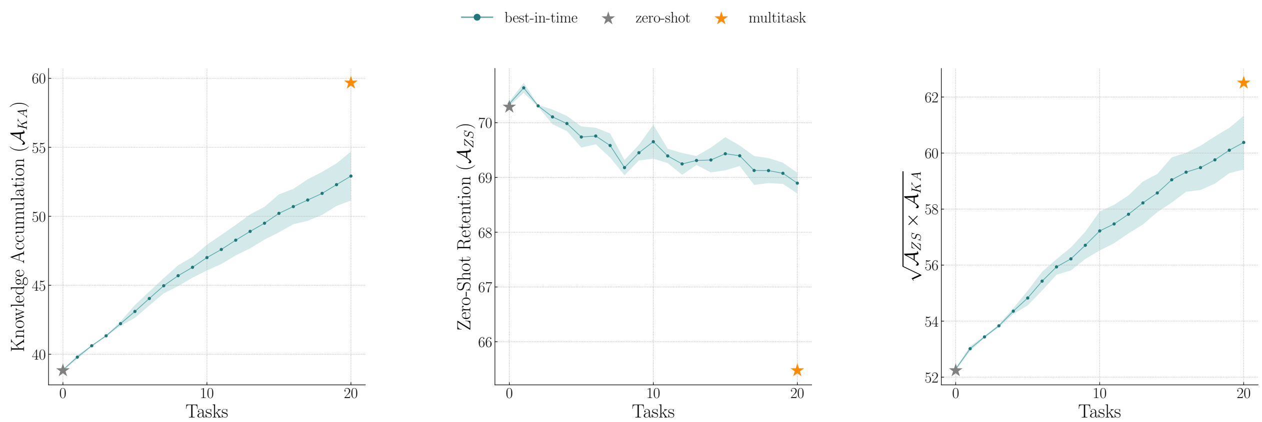

Figure 13. The mean and standard deviation across three runs of *Best-in*-TIME.

To put our results from Sec. 4.5 in perspective, we quantify the variance across runs for a single merging method. Specifically, we run *Best-in*-TIME three times and show the mean and standard deviation across runs in Fig. 13. Comparing this to Fig. 6 reveals that the best results for different methods fall within the standard deviation of multiple runs of the same method. In particular, for the last task, the standard deviation of the geometric mean of knowledge accumulation and zero-shot retention is 0.96.

F. Hyperparameter Details

In an effort to remove any confounding factors, we conduct an extensive hyperparameter sweep, to the best of our abilities, for each individual merging technique for Figs. 3, 4 and 6. We list the hyperparameter ranges swept over for each technique below:

- **Weight Averaging.** For the offline merging, we use a standard merging coefficient of $\frac{1}{N}$, where N is the number of task checkpoints to merge.
- **SLERP.** In SLERP, as we can only merge two checkpoints at a time, we sweep over the following weight-coefficients: {0.1,0.3,0.5,0.7,0.9}.
- **Task-Arithmetic.** We sweep over the scaling factor: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0}
- **TIES.** We sweep over the scaling factor: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0} and the pruning-fraction: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0}.
- **DARE-TIES.** We sweep over the scaling factor: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0} and the pruning-fraction: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0}.
- **Breadcrumbs-TIES.** We sweep over the scaling factor: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0} and the pruning-fraction: {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0}.
- **MagMax.** We sweep over the scaling factor: {0.2,0.4,0.8,1.0}.
- **LiNeS-TIES.** We keep α fixed to 0.5, and sweep β: {0.2,0.5,0.8} and prune-fraction: {0.2,0.5,0.8} as recommended in the original paper [84].