Experiment Figures in Chipmink: Partial Object Store for Temporal Exploration on Notebooks

Supawit Chockchowwat, Sumay Thakurdesai, Zhaoheng Li, Matthew S. Krafczyk, Yongjoo Park
University of Illinois at Urbana-Champaign
Urbana, Illinois, United States
{supawit2,sumayst2,zl20,mkrafcz2,yongjoo}@illinois.edu

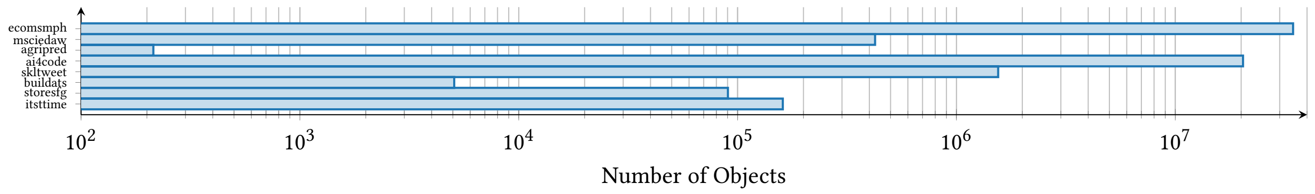
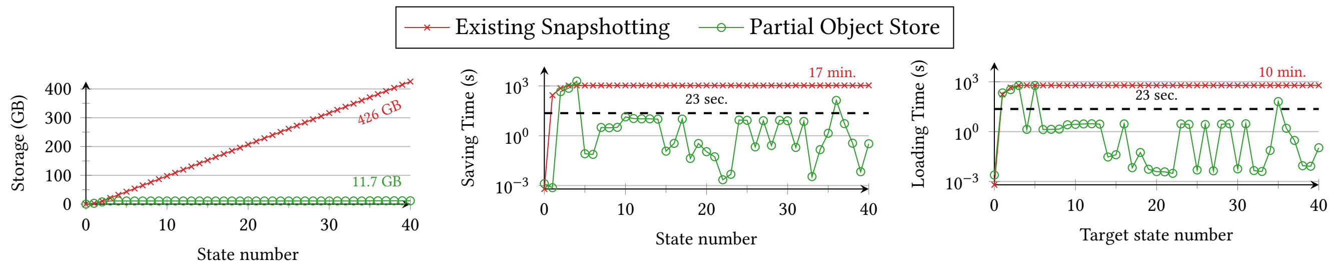


Figure 1: Numbers of objects from real notebooks.

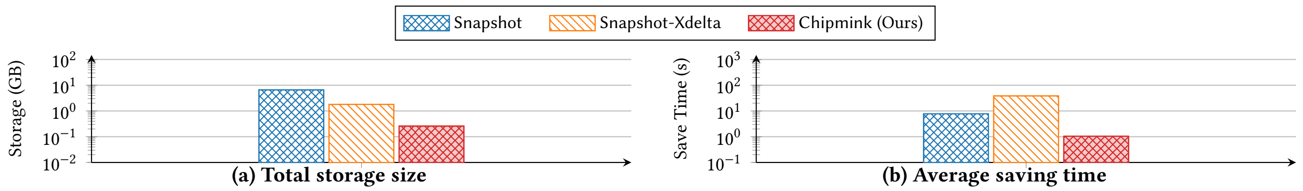


(a) Cumulative storage size

(b) Saving times

(c) Loading times

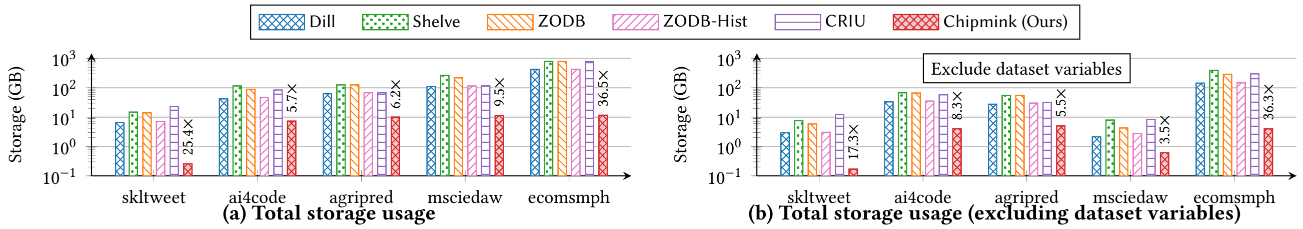
Figure 2: Temporal exploration is feasible on partial object stores; users set aside smaller storage space and perceive fewer saving and loading delays compared to temporal exploration on snapshotting. Plots show cumulative storage sizes and saving times as a user runs `ecomsmph` notebook as well as loading times when the user inspects variables referred by each cell.



(a) Total storage size

(b) Average saving time

Figure 3: Object-aware deltas reduce storage and time costs.



(a) Total storage usage

(b) Total storage usage (excluding dataset variables)

Figure 4: Chipmink stores all variables 5.7–36.5× smaller and selected variables 3.5–36.3× smaller than the best baselines. The plots show the total storage required when saving all variables (left) and saving all but read-only dataset variables (right).

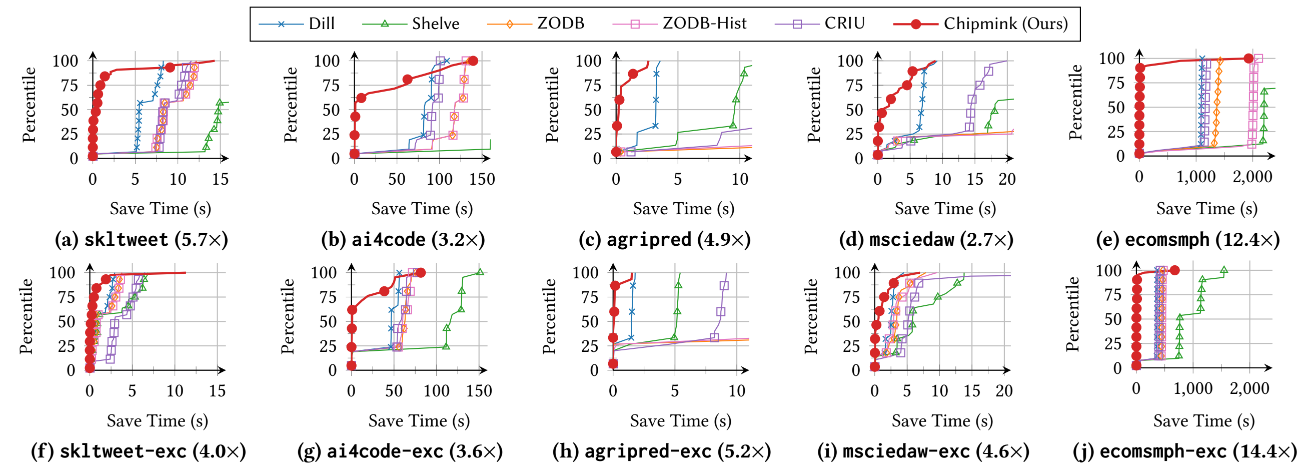


Figure 5: Empirical cumulative distributions (eCDFs) of the perceived saving latency (closer to the top-left corner are better). skltweet-exc refers to skltweet excluding dataset variables and so on. Numbers in parentheses are Chipmink's speedup over the best baseline; Chipmink stores all variables 2.7–12.4× faster and selected variables 3.6–14.4× faster than the best baselines.

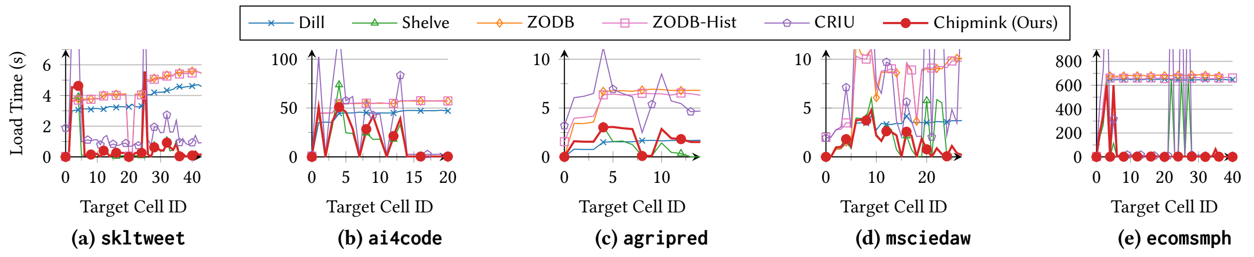


Figure 6: Partial loading time when users are interested in variables accessed at each cell. Chipmink quickly loads target variables proportionally to their sizes, whereas some baselines' performance depends on the entire namespace size.

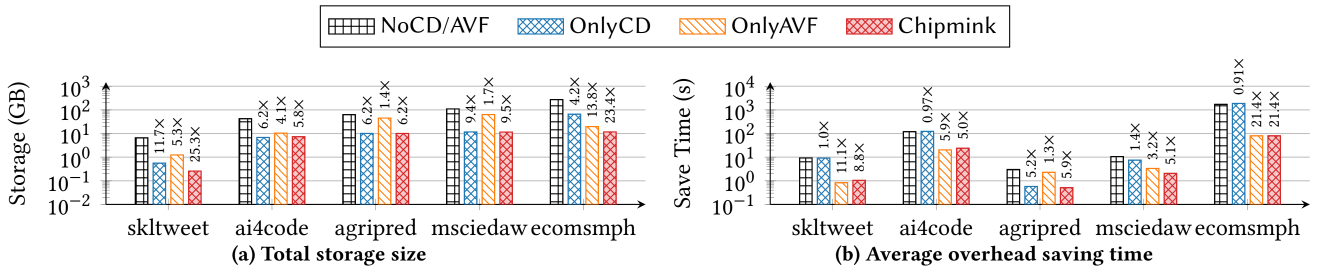


Figure 7: Ablation study: change detector (CD) and active variable filter (AVF) contribute to storage savings and speedups.

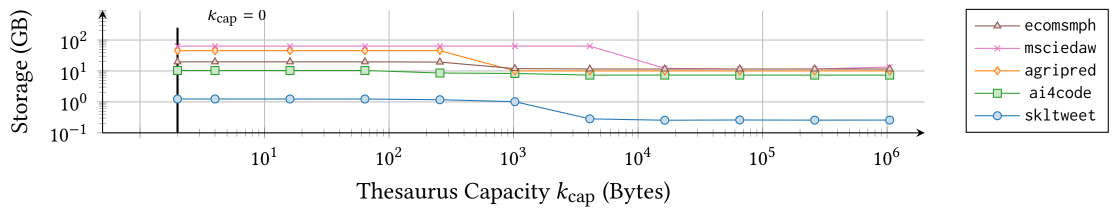


Figure 8: Higher thesaurus capacity reduces storage usage by detecting more synonymous pods. In these notebooks, storage usage converges with k_{cap} around 100 KB.

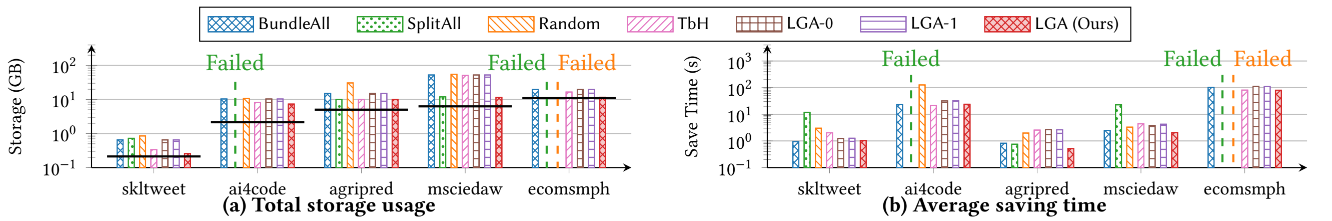


Figure 9: LGA is the most effective podding optimizer in discovering compact podding compared to naive methods (BundleAll, SplitAll, Random), manually derived heuristic (TbH), and LGA with inaccurate volatility models (LGA-0, LGA-1). Thick horizontal lines indicate loose theoretical lower bounds of the optimal storage costs. Exhaustive baseline is studied in Fig. 12.

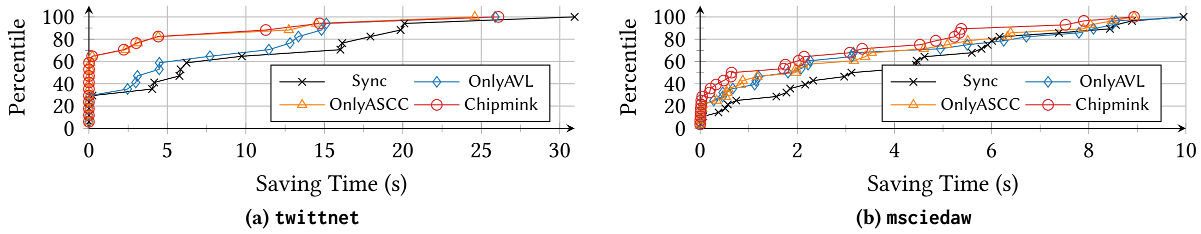


Figure 10: With parallel saving, active variable locking (AVL) and allowlist-based static code checker (ASCC) unblock user's cell executions, improving over synchronous saving.

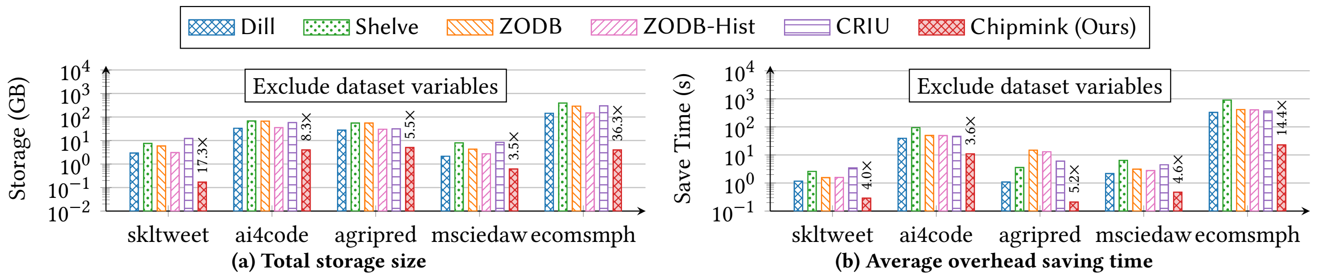


Figure 11: Even when users selectively save some variables and exclude read-only dataset, Chipmink still compactly and quickly stores them, 3.5–36.3× smaller and 3.6–14.4× faster than the best baselines.

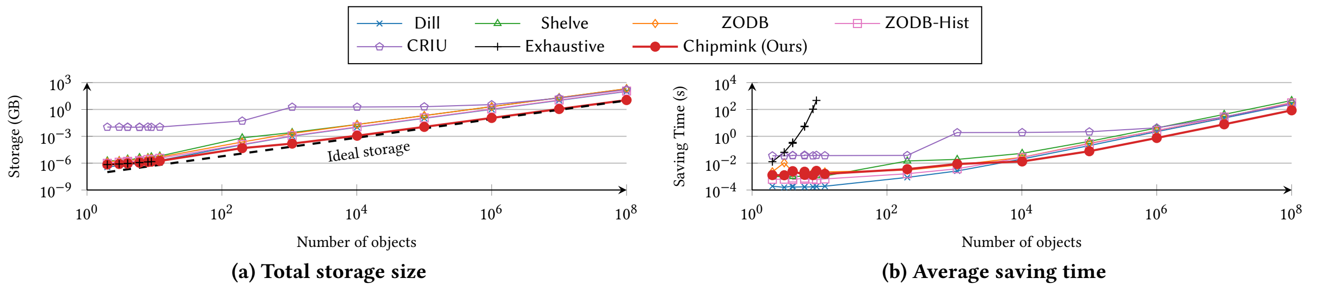
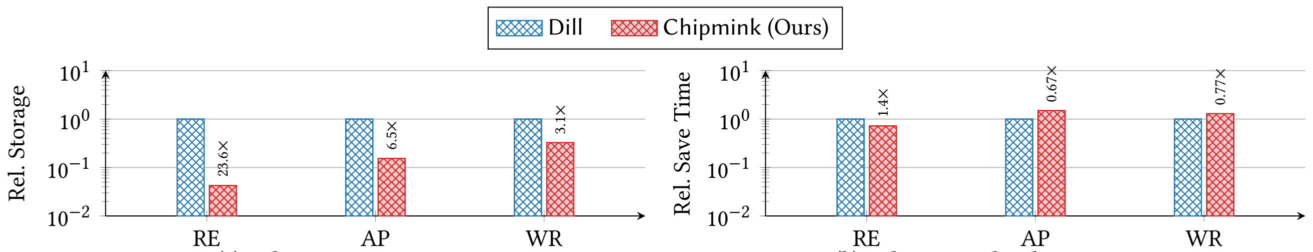
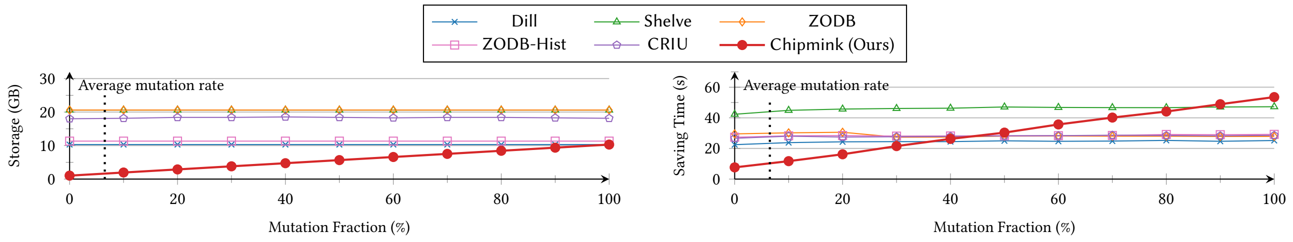


Figure 12: Storage and save time as namespace size scales.



(a) Relative storage size **(b) Relative overhead saving time**

Figure 13: **Chipmink storage size and overhead saving time relative to Dill across different notebook characteristics (RE: read-only cells, AP: cells appending new variables, WR: cells applying write operations on the same variable). Chipmink saves storage space even when the notebook heavily mutates its namespace; however, Chipmink can be slower in such cases.**



(a) Total storage size **(b) Average saving time**

Figure 14: **Storage and save time when the notebook mutates 1 GB of data over 10 cells at varied rates. Dotted lines displays the mutation fraction averaged over 5 real notebooks.**

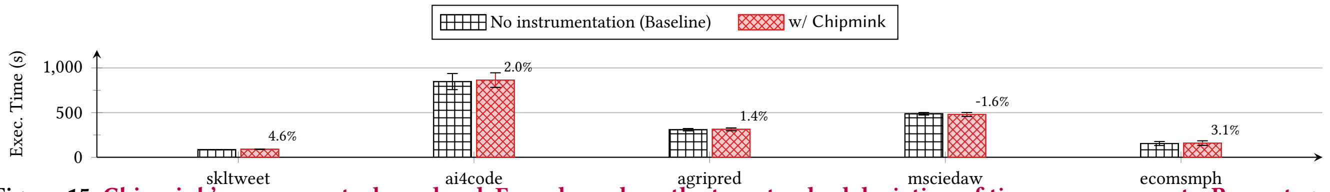


Figure 15: **Chipmink's access control overhead. Error bars show the two standard deviation of time measurements. Percentages refer relative overheads over the baseline.**