Experiment Figures in
Chipmink: Native Object Database for Data Science

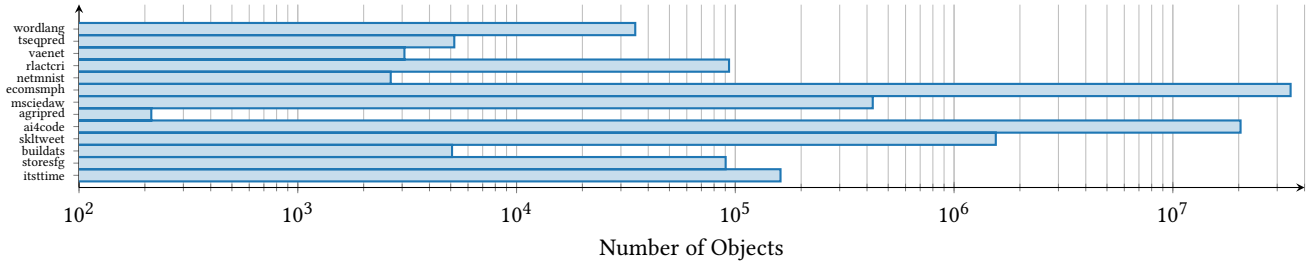
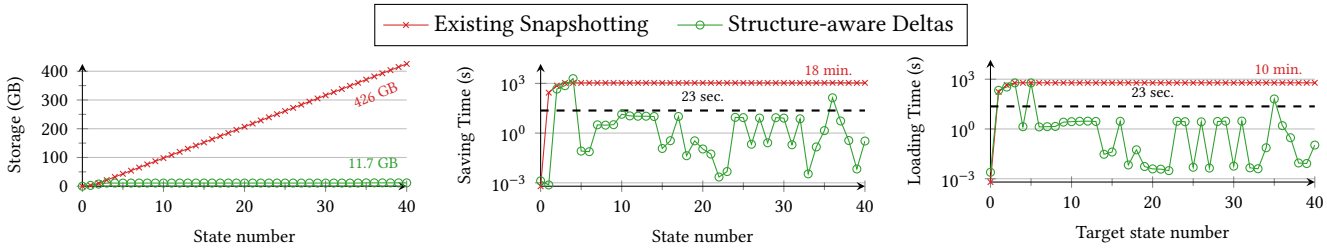
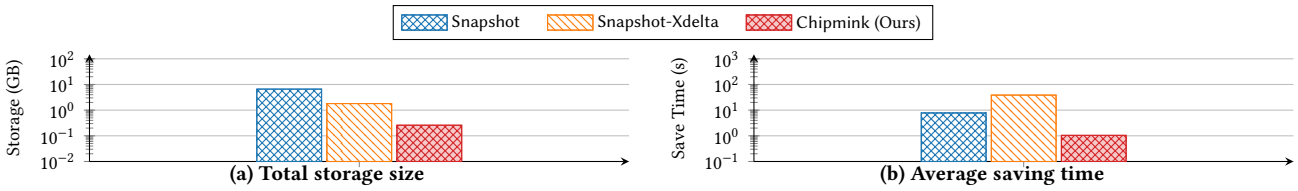


Figure 1: Numbers of objects from real notebooks.



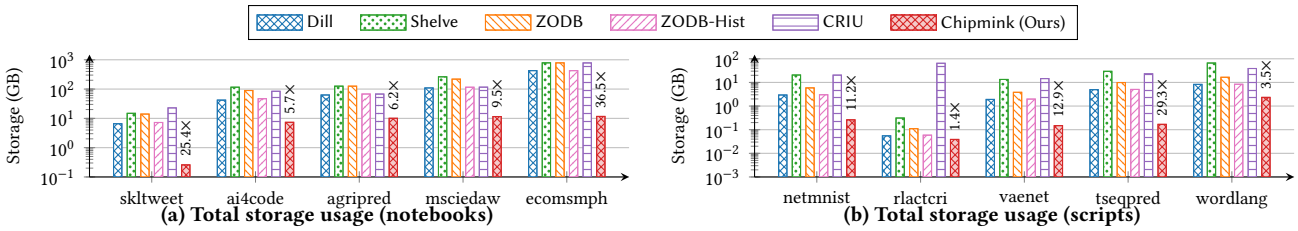
(a) Cumulative storage size **(b) Saving times** **(c) Loading times**

Figure 2: Graph-based deltas improve efficiency: smaller storage space and faster saving/loading compared to complete snapshotting. Plots show (a) cumulative storage sizes, (b) saving times as a user runs `ecomsmph` notebook, and (c) 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 (notebooks) **(b) Total storage usage (scripts)**

Figure 4: Chipmink stores all variables with 5.7–36.5× smaller storage on notebooks and 1.4–29.3× smaller on scripts than the best baselines. The plots show the total storage required when saving all variables in the namespace at different points in time.

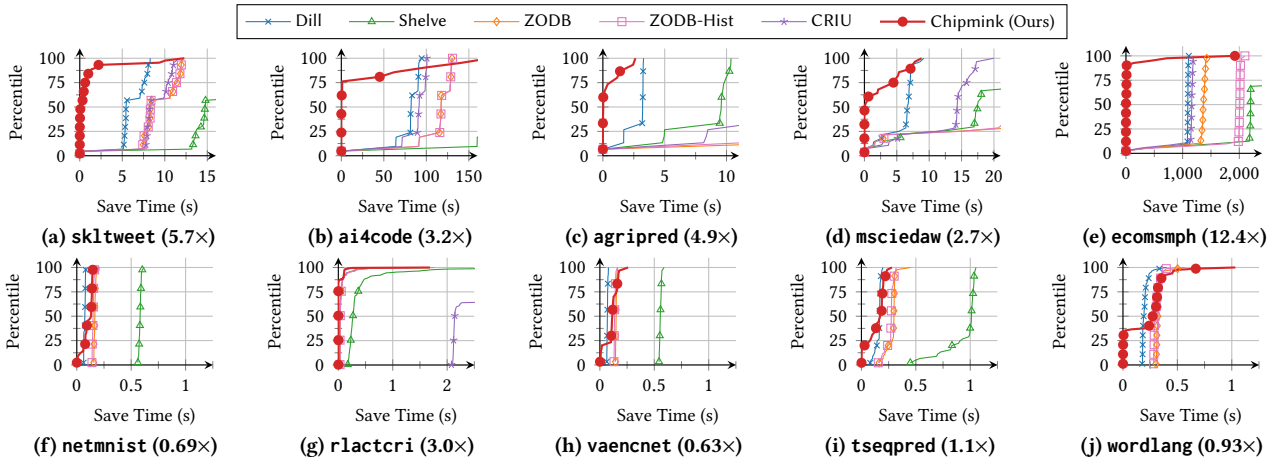


Figure 5: Empirical cumulative distributions (eCDFs) of the perceived saving latency (closer to the top-left corner are better). Numbers in parentheses are Chipmink's speedup over the best baseline; Chipmink stores all variables 2.7–12.4× faster than the best baselines on notebooks while remaining competitive with the fastest baselines on scripts.

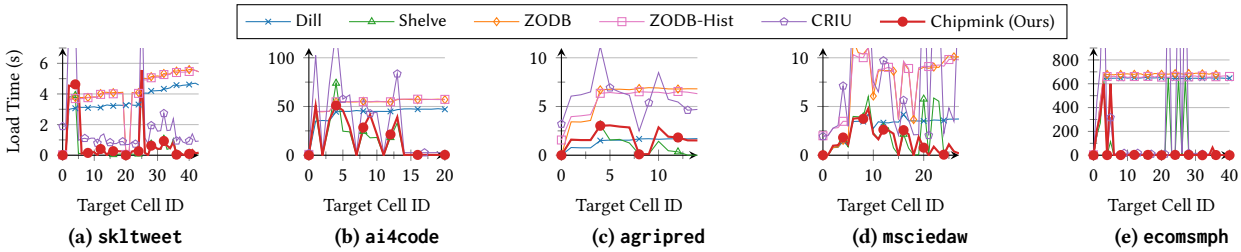


Figure 6: Partial loading time when users are interested in variables accessed at each cell. Chipmink quickly loads target variables proportionally to their sizes, whereas some baselines' performance depends on the entire namespace size.

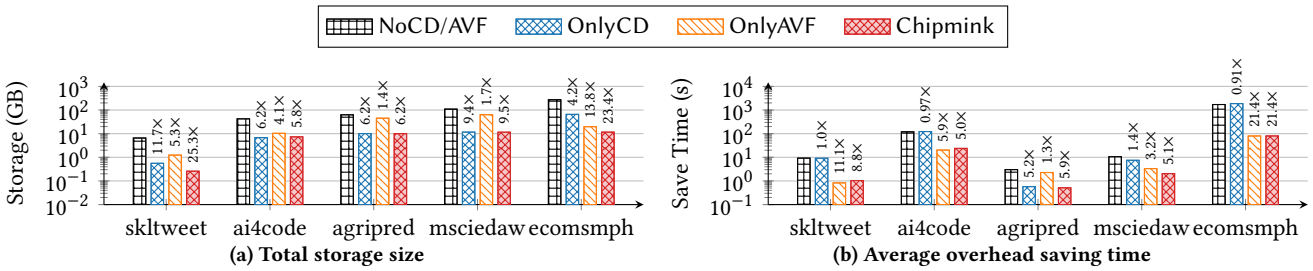


Figure 7: Ablation study: change detector (CD) and active variable filter (AVF) contribute to storage savings and speedups.

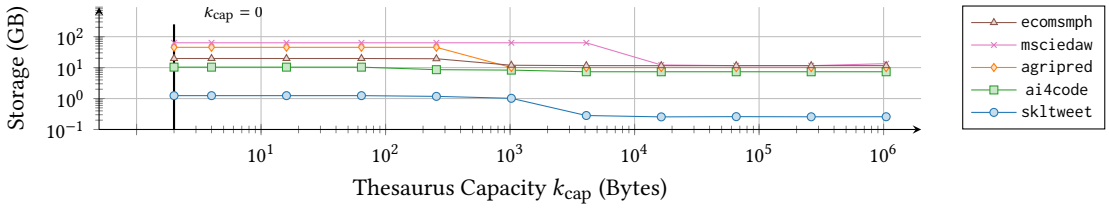
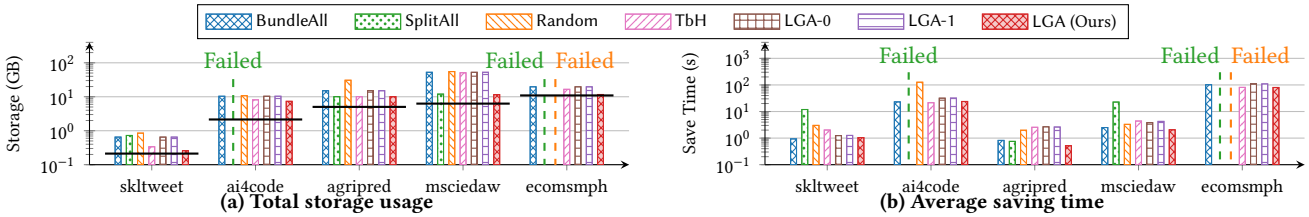
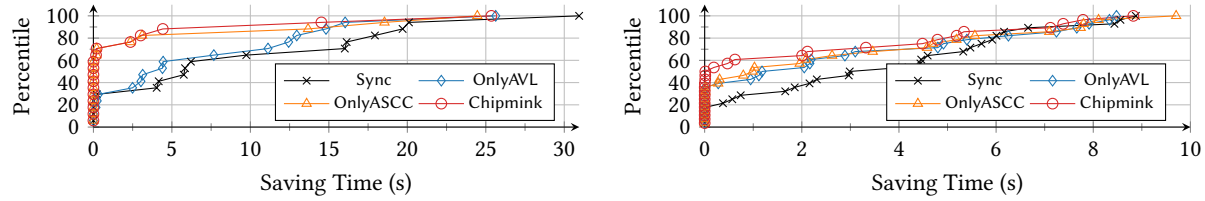


Figure 8: Higher thesaurus capacity reduces storage usage by detecting more synonymous pods. In these notebooks, storage usage converges with k_{cap} around 100 KB.



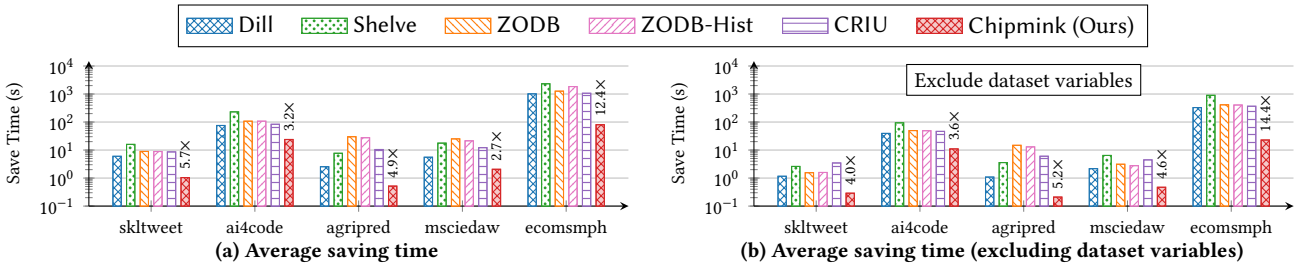
(a) Total storage usage (b) Average saving time

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.



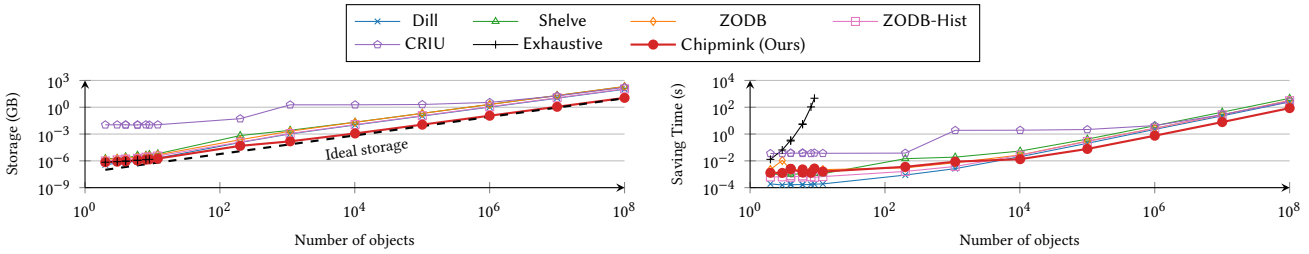
(a) `twittnet` (b) `msciedaw`

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.



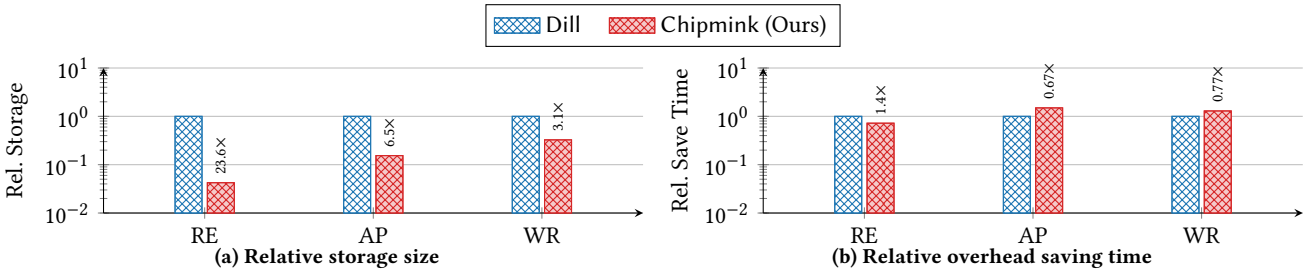
(a) Average saving time (b) Average saving time (excluding dataset variables)

Figure 11: Chipmink stores all variables 2.7–12.4× faster and selected variables 3.6–14.4× faster than the best baselines. The plots show the delay incurred for each cell when saving all variables (left) and saving all but read-only dataset variables (right).



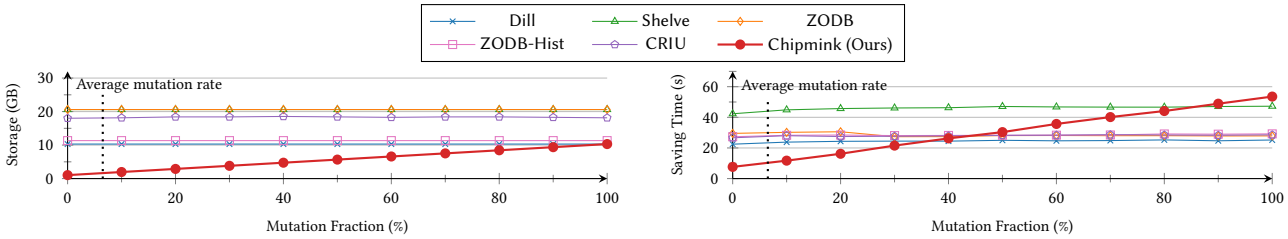
(a) Total storage size (b) Average saving time

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. The dotted lines display the mutation fraction averaged over 5 real notebooks.

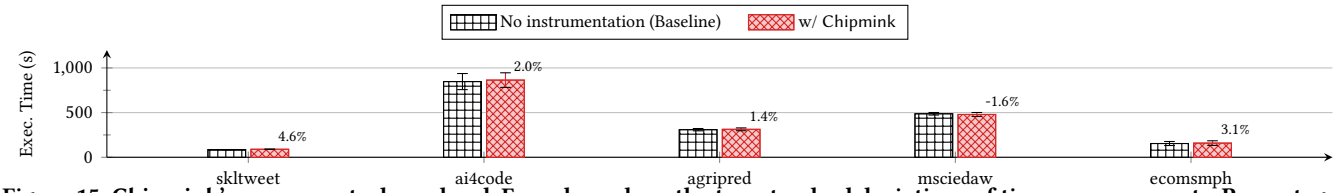


Figure 15: Chipmink's access control overhead. Error bars show the two standard deviations of time measurements. Percentages refer to relative overheads over the baseline.