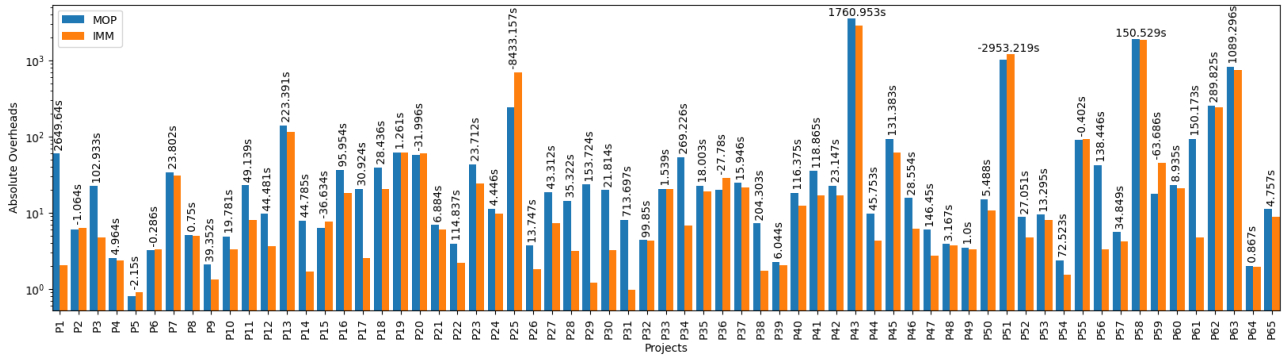

Fig. 1: Plot (in log scale) of JavaMOP and DEIMM absolute overheads, computed as time to monitor test runs with RV minus time to run tests w/o RV. Bars for the same project are labeled with the time that DEIMM saves if used instead of JavaMOP; negative numbers mean slowdowns. Similar to Figure 10 in the paper, but excluded code transformation time.

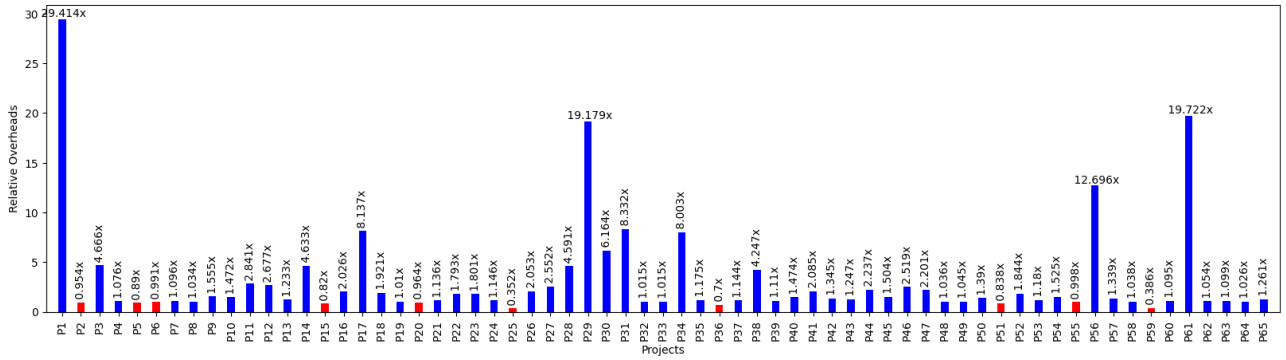

Fig. 2: MOP / DEIMM time. Projects with speedups are in blue; projects with slowdowns are in red. Similar to Figure 11 in the paper, but excluded code transformation time.

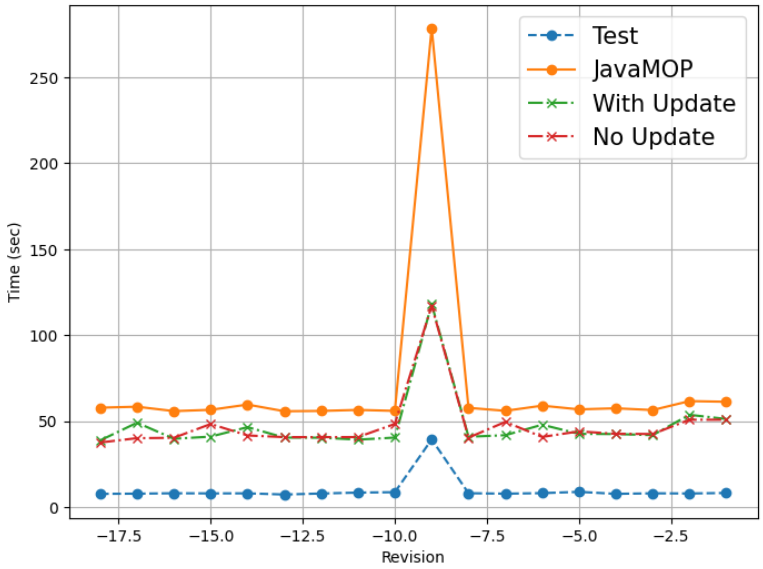

Fig. 3: Simulated RV overhead during software evolution for project conveyal/osm-lib

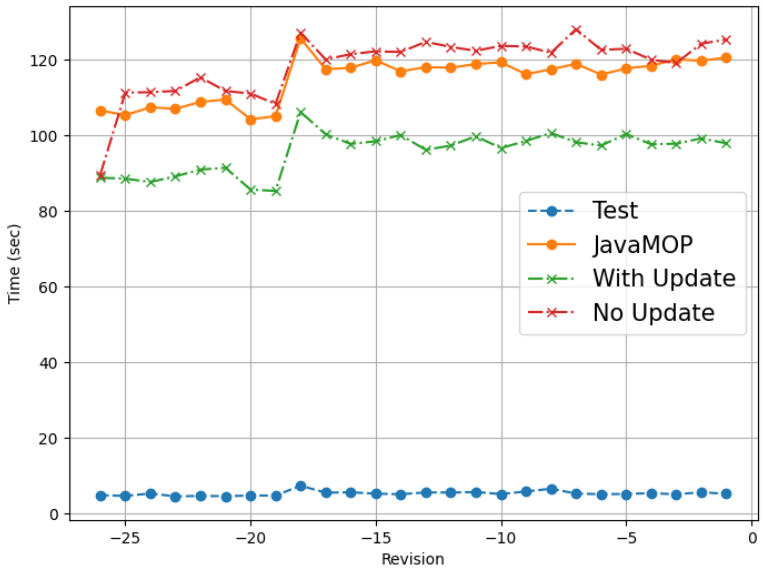

Fig. 4: Simulated RV overhead during software evolution for project davidmoten/rtree-multi

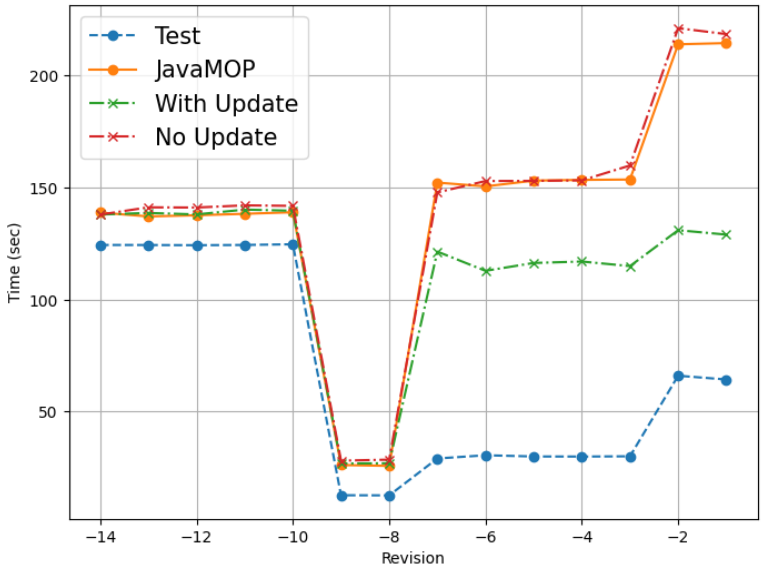

Fig. 5: Simulated RV overhead during software evolution for project flipkart-incubator/databuilderframework

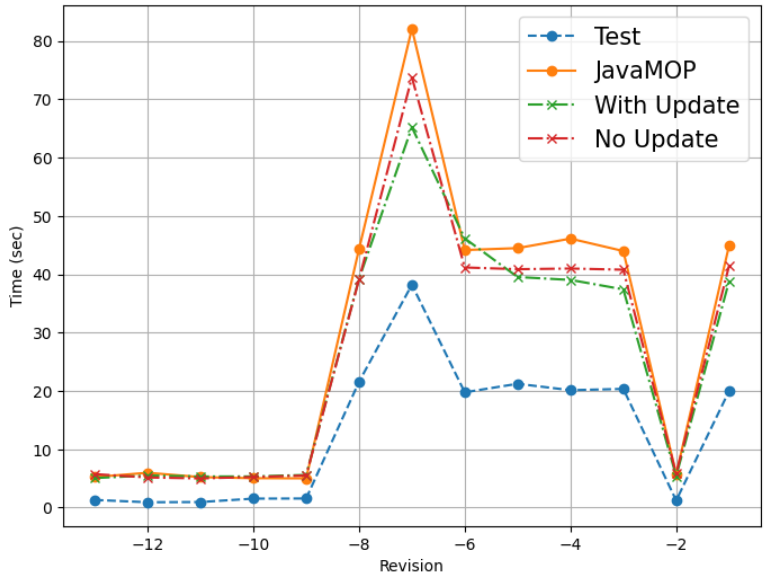

Fig. 6: Simulated RV overhead during software evolution for project Grundlefleck/ASM-NonClassloadingExtensions

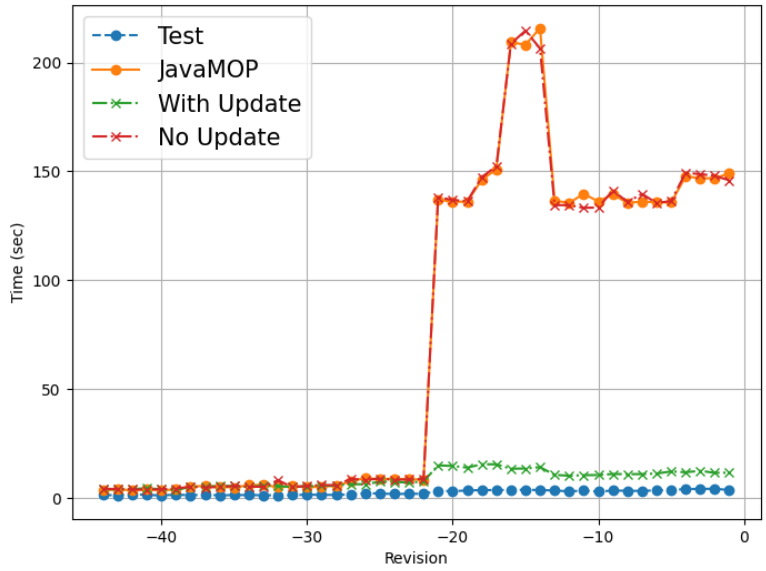

Fig. 7: Simulated RV overhead during software evolution for project houbb/sensitive-word

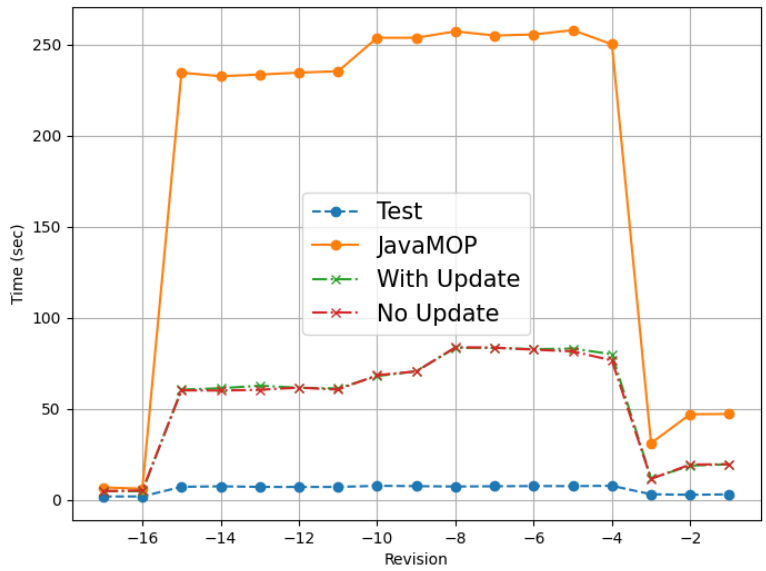

Fig. 8: Simulated RV overhead during software evolution for project huaban/jieba-analysis

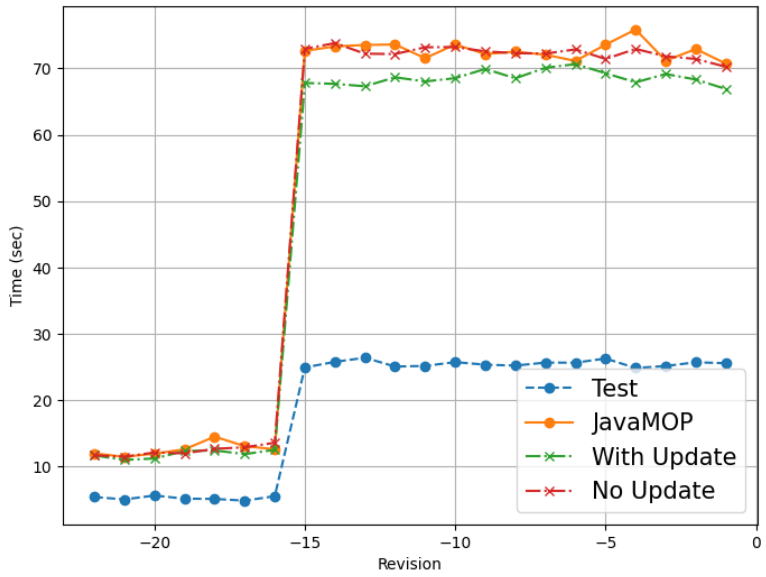

Fig. 9: Simulated RV overhead during software evolution for project myui/btree4j

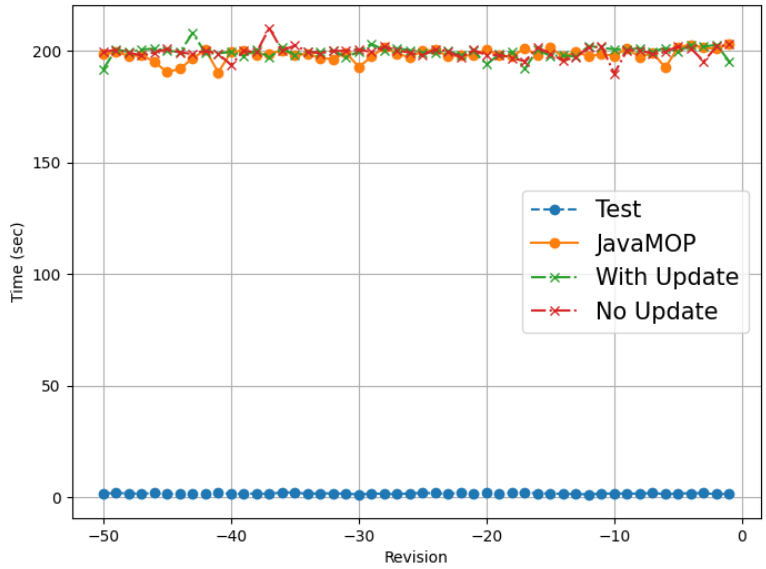

Fig. 10: Simulated RV overhead during software evolution for project solita/functional-utils

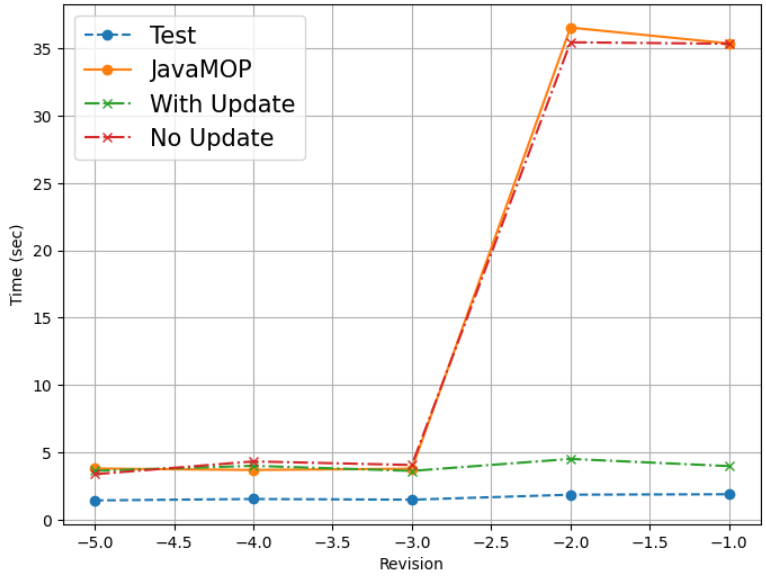

Fig. 11: Simulated RV overhead during software evolution for project StarlangSoftware/TurkishSentiNet

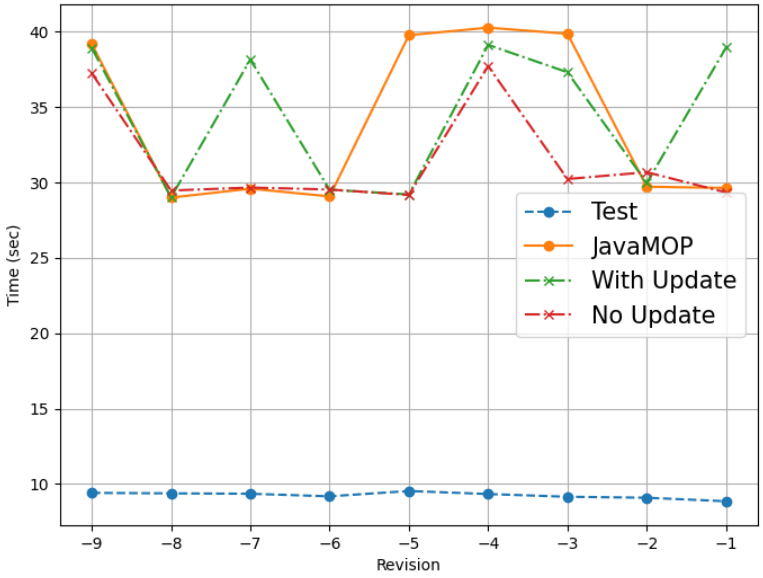

Fig. 12: Simulated RV overhead during software evolution for project wiqer/ef-redis