Supplementary File for Debunking the Myth of Join Ordering: Toward Robust SQL Analytics

Anonymous

Due to the space limitation, we fail to show all the experiment results in our paper. So we add this appendix (supplementary) file as a supplement.

A PERFORMANCE WITH OPTIMIZER'S PLAN

In Appendix A, we show all the performance with the optimizer's plan of Robust Predicate Transfer compared to our baseline. Our baseline includes vanilla DuckDB, Bloom Join, and Predicate Transfer.

Figure 1 shows the execution time with the optimizer's plan for each query in TPC-H. Note that we omit Q1 and Q6 as they only scan and filter one table. On average (geometric mean), Robust Predicate Transfer outperforms vanilla DuckDB by 1.53×, Bloom Join by 1.33×. And Robust Predicate Transfer shares the same performance with Predicate Transfer. This is because TPC-H is simple and the transfer scheduling of Predicate Transfer and Robust Predicate Transfer does not differ a lot.

Figure 2 shows the execution time with the optimizer's plan for one result from each of the 33 query templates in JOB. On average (geometric mean), Robust Predicate Transfer outperforms vanilla DuckDB by 1.46×, Bloom Join by 1.29×. And Robust Predicate Transfer shares the same performance with Predicate Transfer.

Figure 3 and Figure 4 show the execution time with the optimizer's plan for each query in TPC-DS. On average (geometric mean), Robust Predicate Transfer outperforms vanilla DuckDB by 1.56×, Bloom Join by 1.48×, and Predicate Transfer by 1.23×. Note that for some queries (like 16, 61, and 69), Robust Predicate Transfer has a very poor performance compared to vanilla DuckDB and Bloom Join. This is because the result is empty in those queries. Thus, Robust Predicate Transfer has to scan more tables than vanilla DuckDB and Bloom Join as the query execution stops as soon as it meets empty intermediate results.

Figure 5 and Figure 6 show the execution time with the optimizer's plan for each query in DSB. On average (geometric mean), Robust Predicate Transfer outperforms vanilla DuckDB by 1.54×, Bloom Join by 1.45×, and Predicate Transfer by 1.23×. Note that due to the same reason in TPC-DS, for some specific queries, Robust Predicate Transfer has a very poor performance compared to vanilla DuckDB and Bloom Join as well.

B LEFT DEEP JOIN ORDER ROBUSTNESS

In Appendix B, we show the distribution of execution time with random left deep plans for each query of Robust Predicate Transfer compared to our baseline in Figure 7a (TPC-H), Figure 8 (JOB), Figure 9 (TPC-DS query 1-52), Figure 10 (TPC-DS query 53-99), Figure 11 (DSB query 1-52) and Figure 12 (DSB query 53-99). Our baseline includes vanilla DuckDB, Bloom Join, and Predicate Transfer.

We can see that for most acyclic queries, Robust Predicate Transfer and Predicate Transfer are both more robust than vanilla DuckDB and Bloom Join. However, in the specific acyclic query (JOB 32a and 32b, TPC-DS 54 and 83, DSB 54 and 83), Predicate Transfer is also not robust as well. This is because the transfer algorithm `Small2Large` used by Predicate Transfer has no theory guarantee.

And even for the cyclic queries, Robust Predicate Transfer can also improve the robustness to some degree. However, as there is no theory guarantee for cyclic queries, Robust Predicate Transfer failed to constrain their maximum execution time.

C BUSHY JOIN ORDER ROBUSTNESS

In Appendix C, we show the distribution of execution time with random bushy plans for each query of Robust Predicate Transfer compared to our baseline in Figure 7b (TPC-H) and Figure 13 (JOB), Figure 14 (TPC-DS query 1-52), Figure 15 (TPC-DS query 53-99), Figure 16 (DSB query 1-52) and Figure 17 (DSB query 53-99). Our baseline includes vanilla DuckDB, Bloom Join, and Predicate Transfer. Note that the figure of TPC-H has already been shown in the paper, so we do not display it here.

The conclusion is the same as the left-deep results. But we can notice that the robustness of Robust Predicate Transfer is worse than the left-deep. We have discussed the reason in our paper that this is due to the wrong probe-build side selection during the hash join.

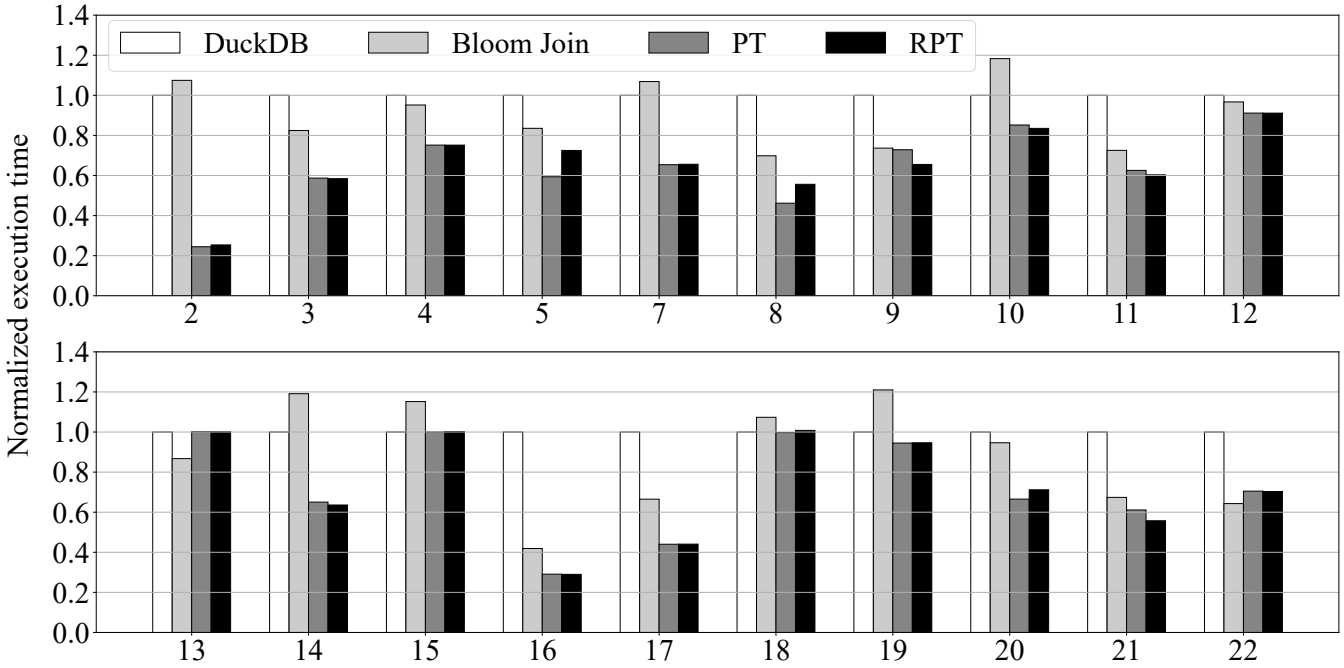


Figure 1: The execution time with optimizer's plans for each query in TPC-H – Normalized by the execution time of default DuckDB. We omit Q1 and Q6 as they are only the table scan and filtering.

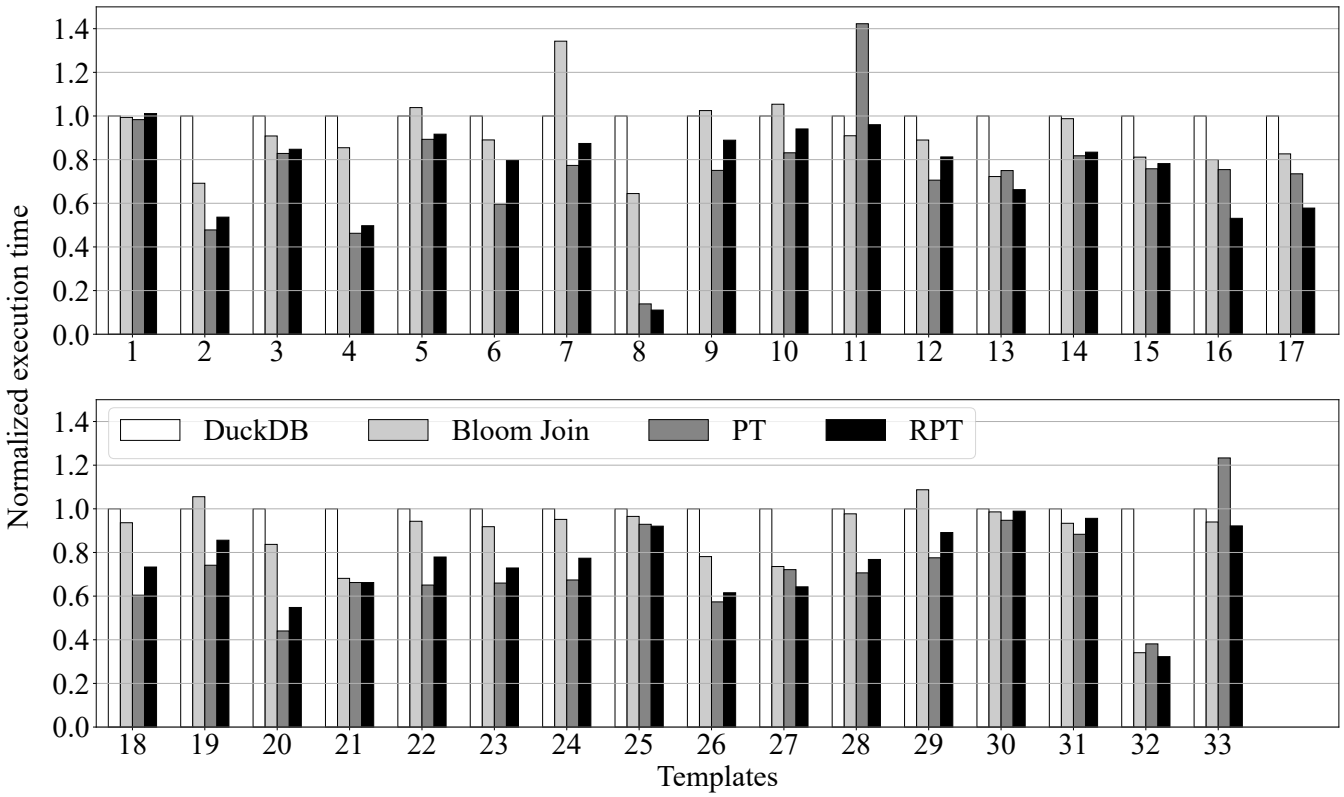


Figure 2: The execution time with optimizer's plans for each query in JOB – Normalized by the execution time of default DuckDB.

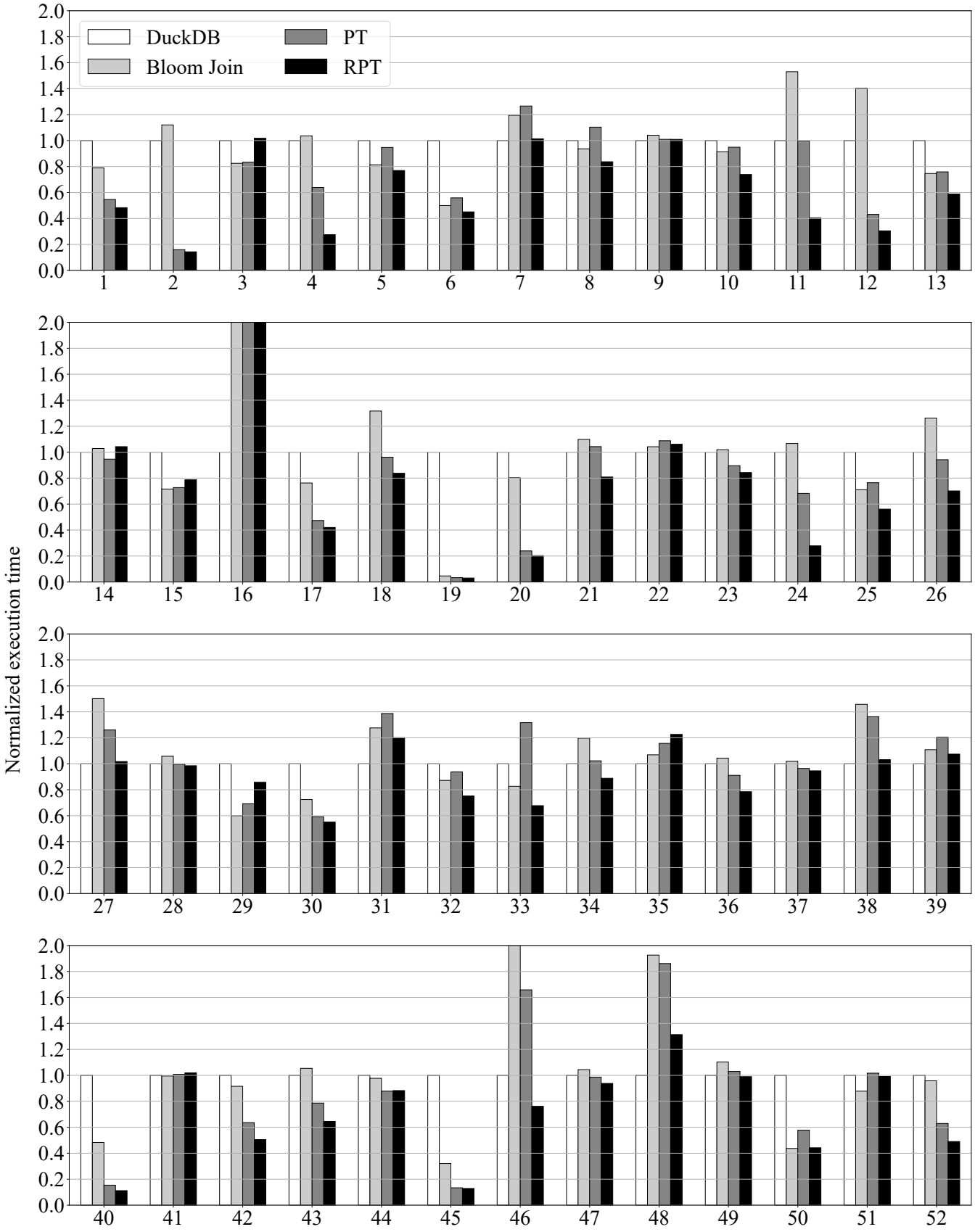


Figure 3: The execution time with optimizer's plans for each query (1 - 52) in TPC-DS – Normalized by the execution time of default DuckDB.

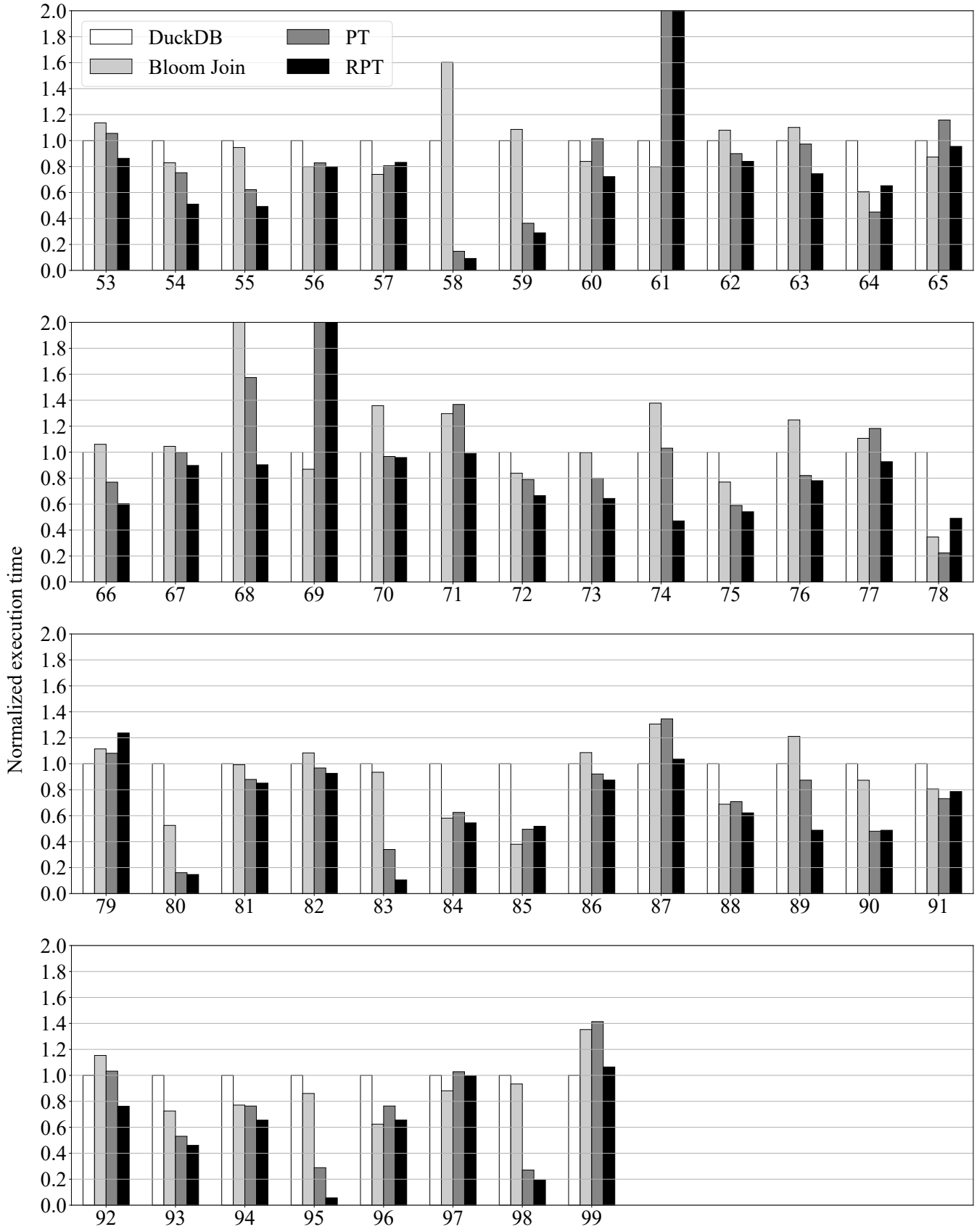


Figure 4: The execution time with optimizer's plans for each query (53 - 99) in TPC-DS – Normalized by the execution time of default DuckDB.

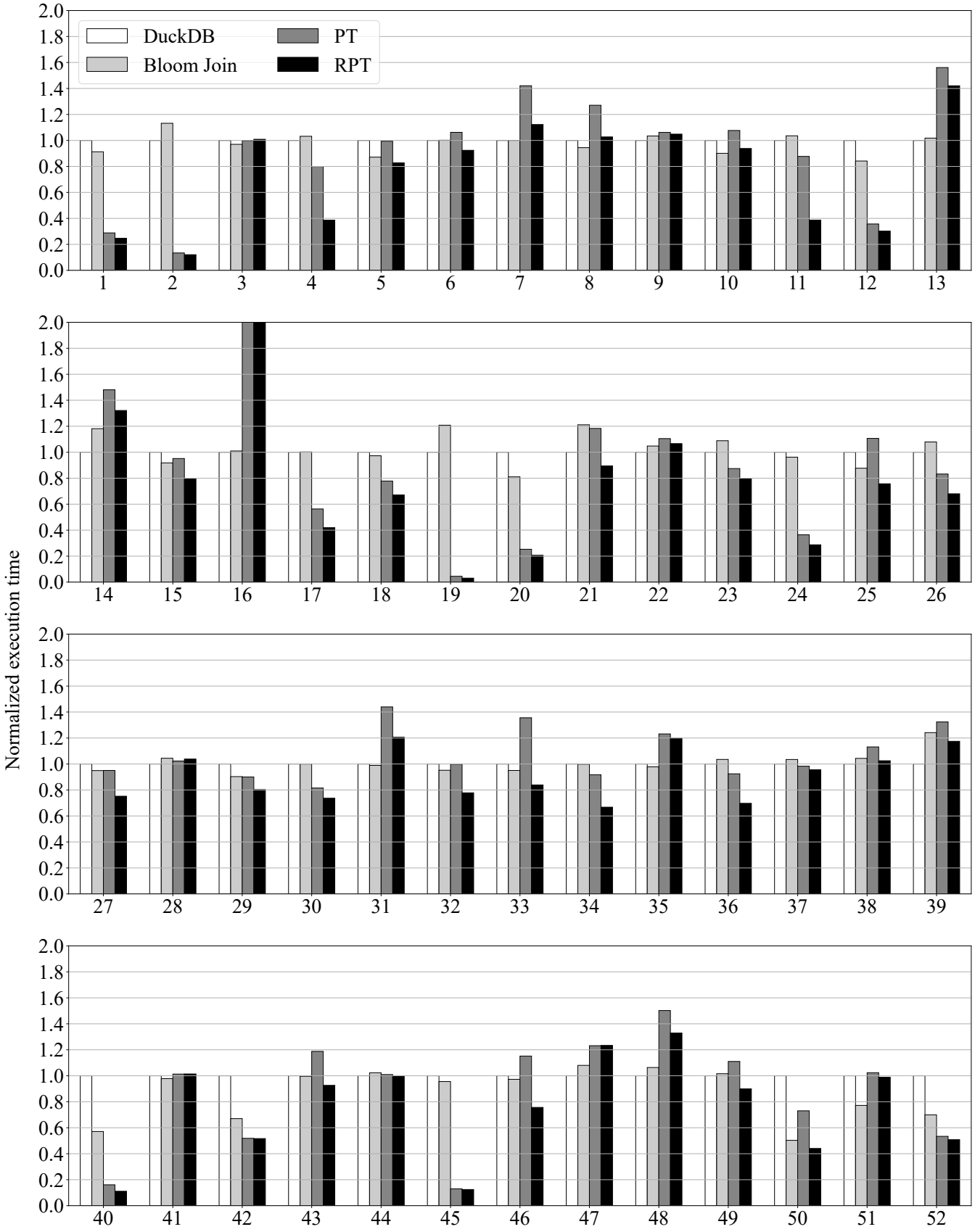


Figure 5: The execution time with optimizer's plans for each query (1 - 52) in DSB – Normalized by the execution time of default DuckDB.

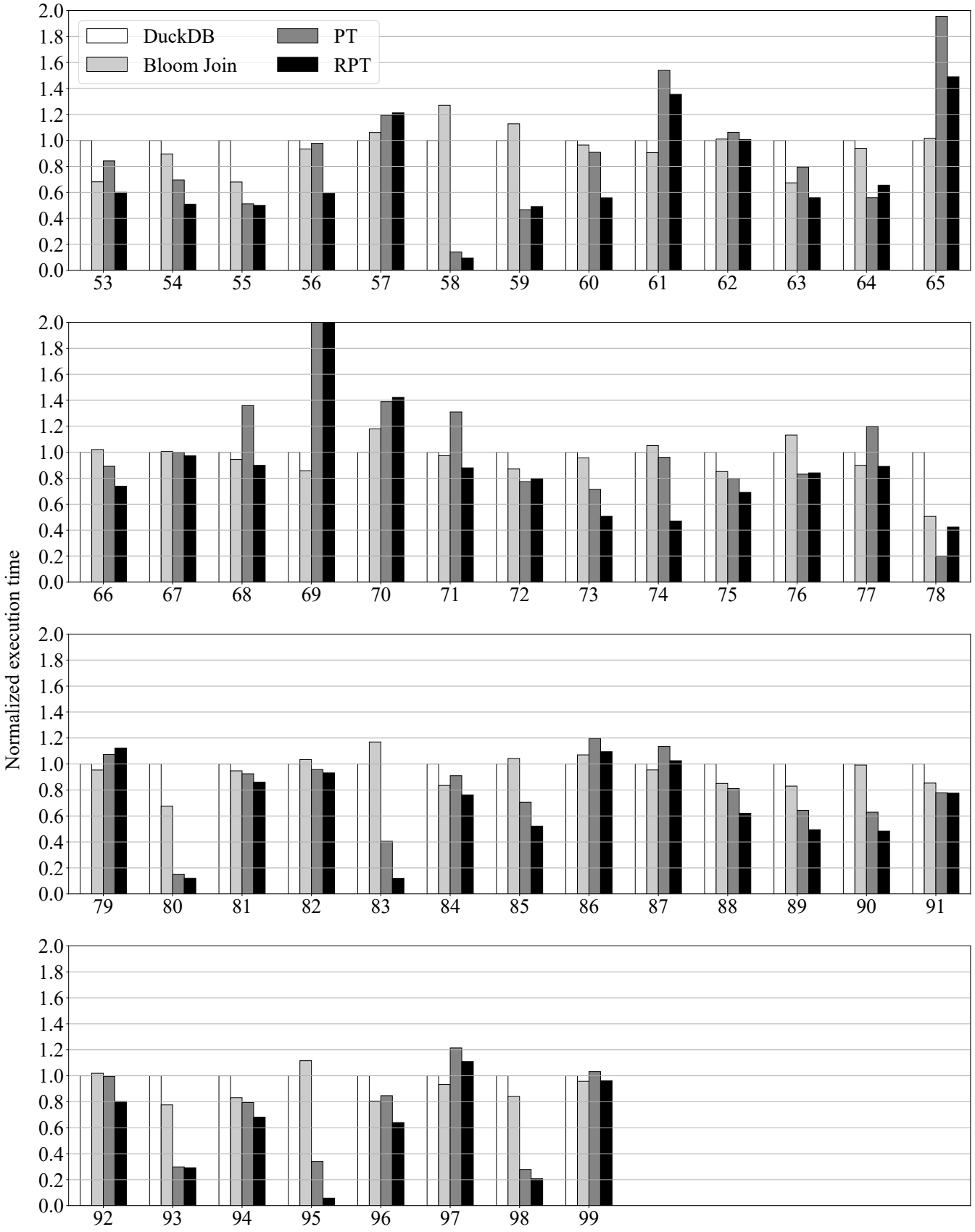
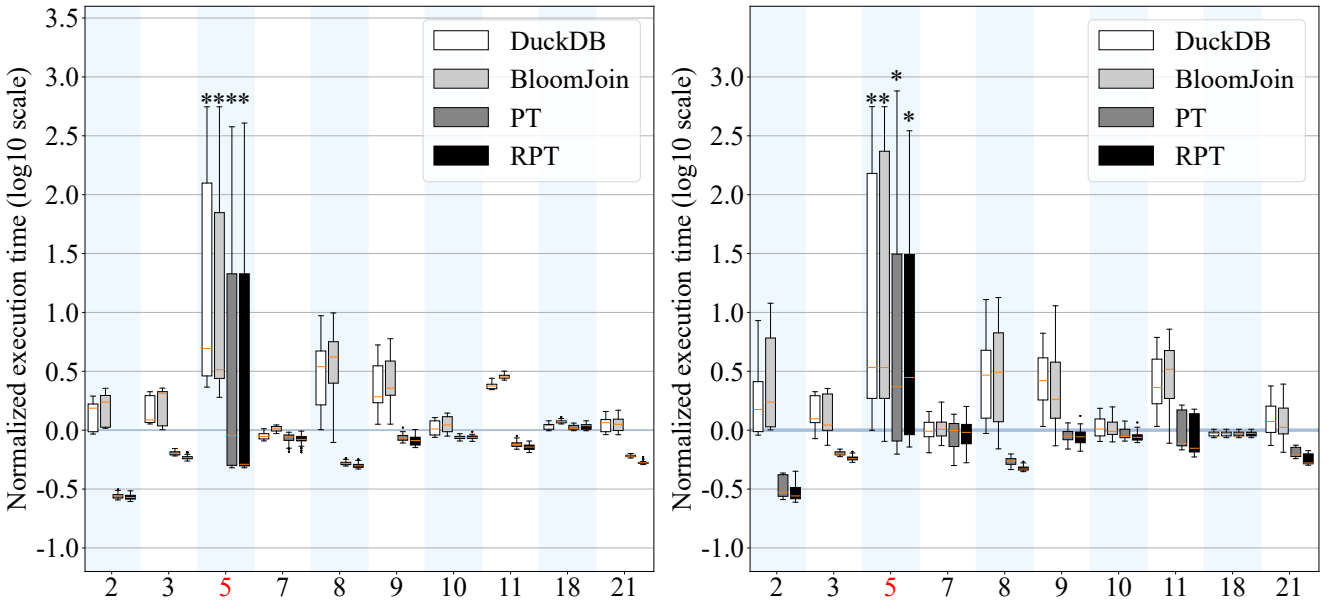


Figure 6: The execution time with optimizer's plans for each query (53 - 99) in DSB – Normalized by the execution time of default DuckDB.



(a) Only left deep **(b) Bushy**

Figure 7: The distribution of execution time with random left deep plans for each query in TPC-H – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

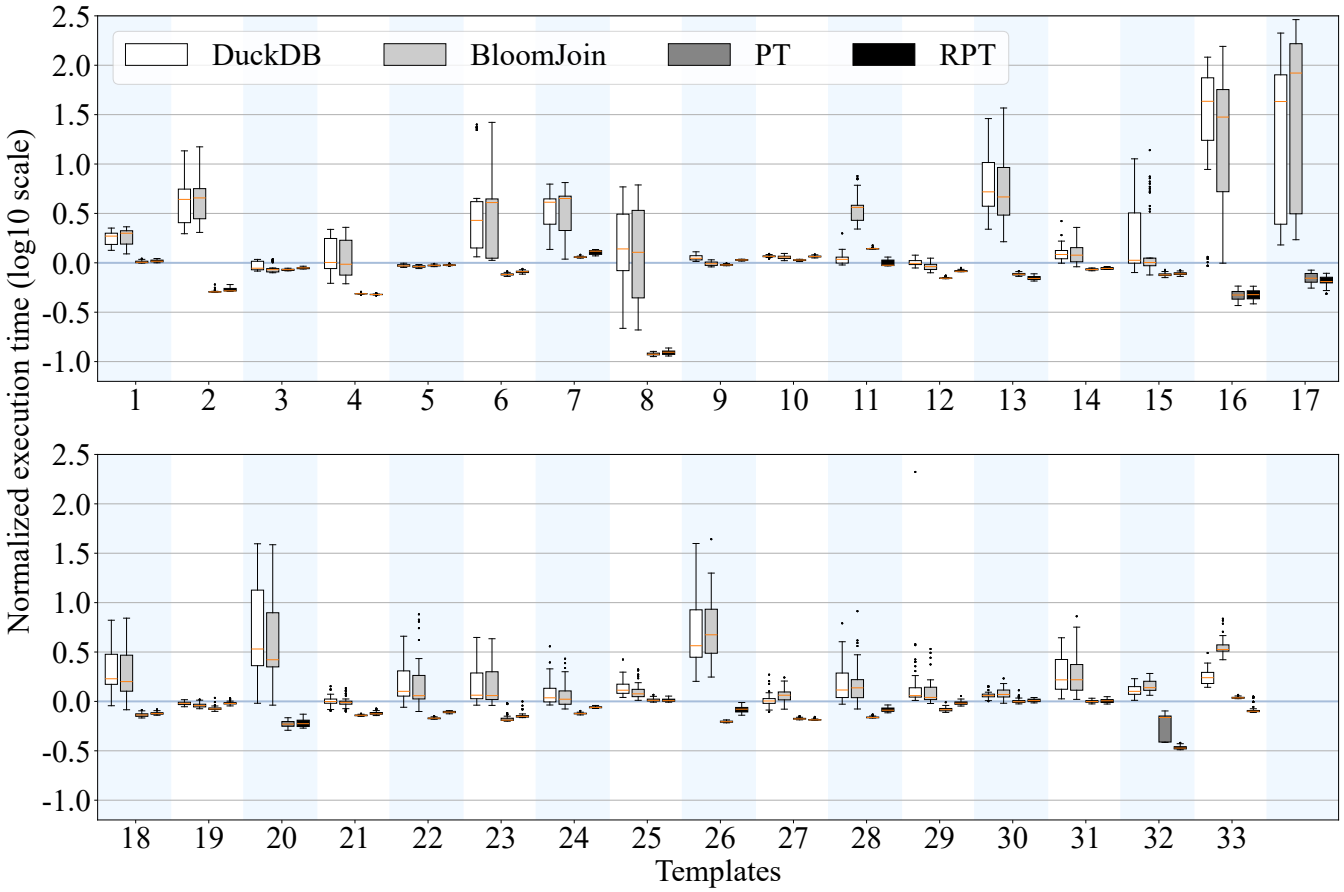


Figure 8: The distribution of execution time with random left deep plans for each template in JOB – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers).

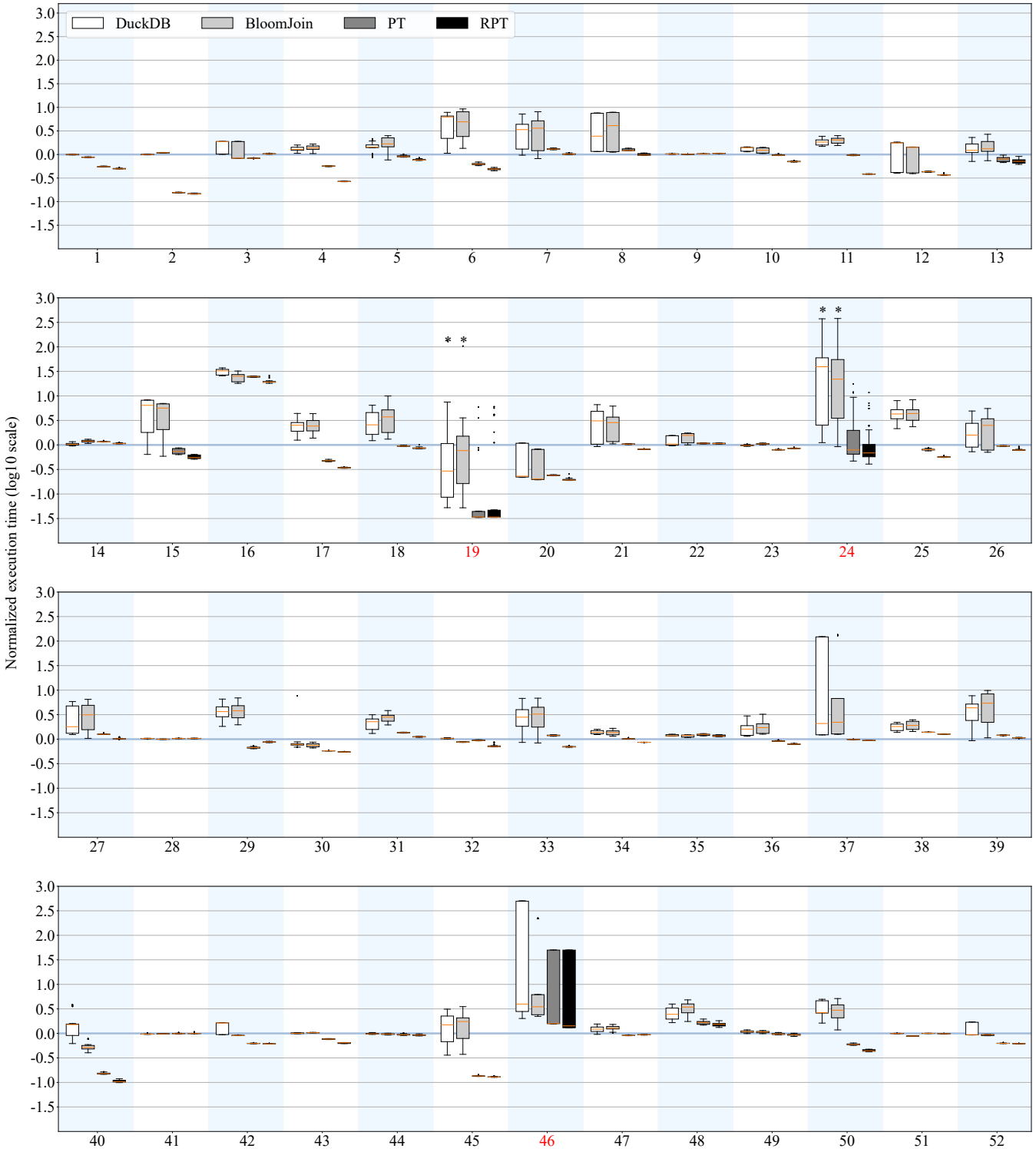


Figure 9: The distribution of execution time with random left deep plans for each query (1 - 52) in TPC-DS – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

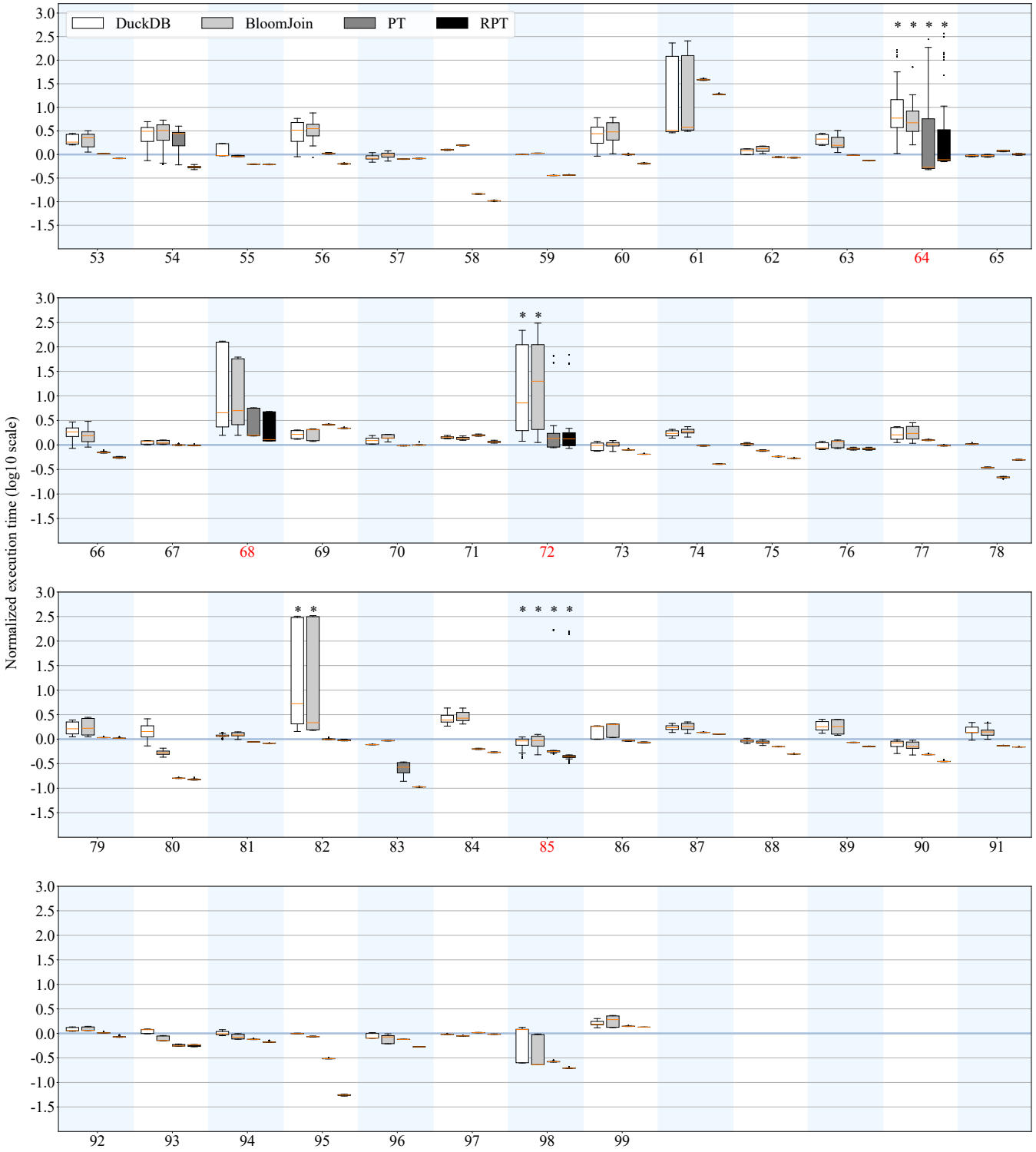


Figure 10: The distribution of execution time with random left deep plans for each query (53 - 99) in TPC-DS – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

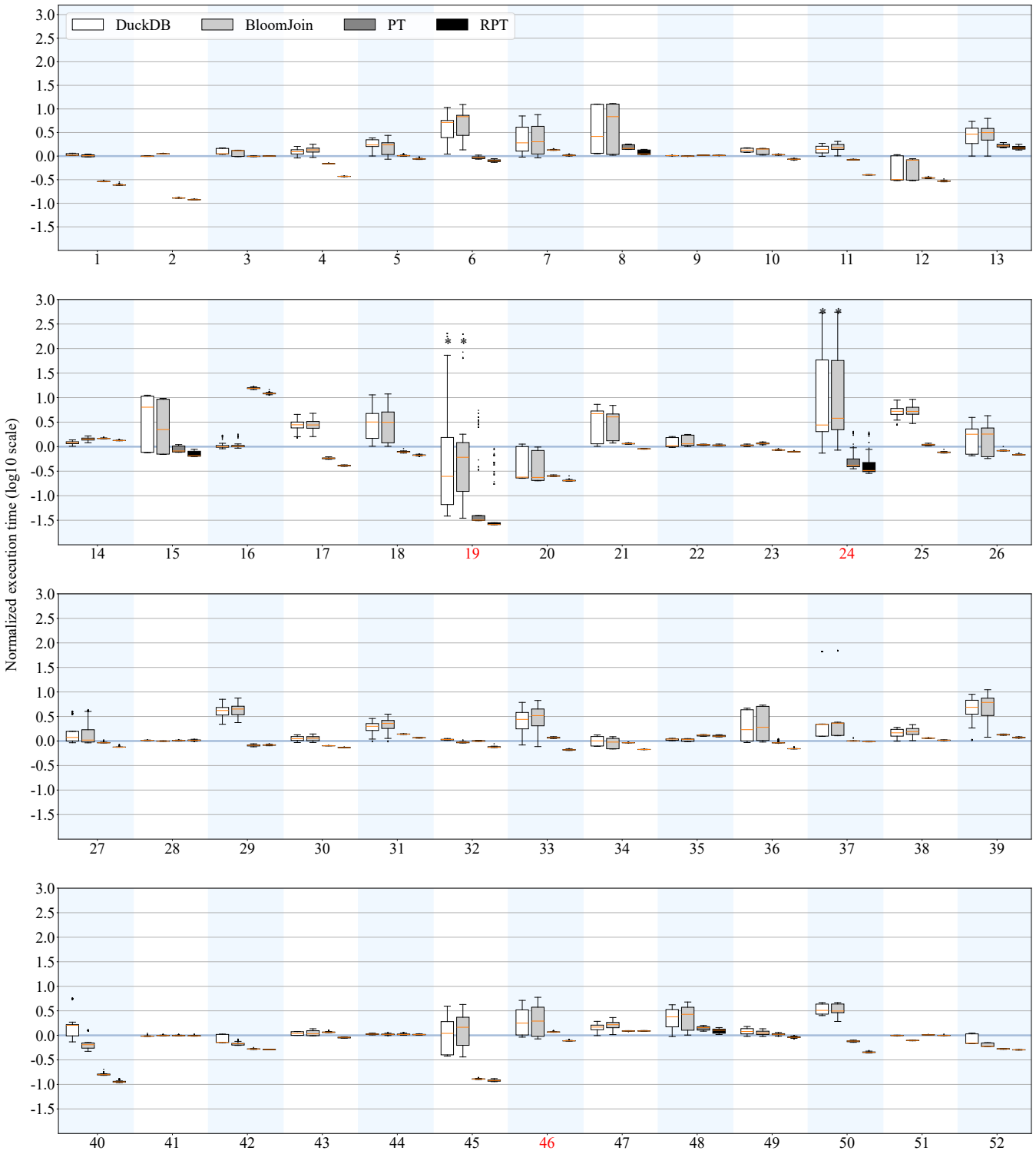


Figure 11: The distribution of execution time with random left deep plans for each query (1 - 52) in DSB – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

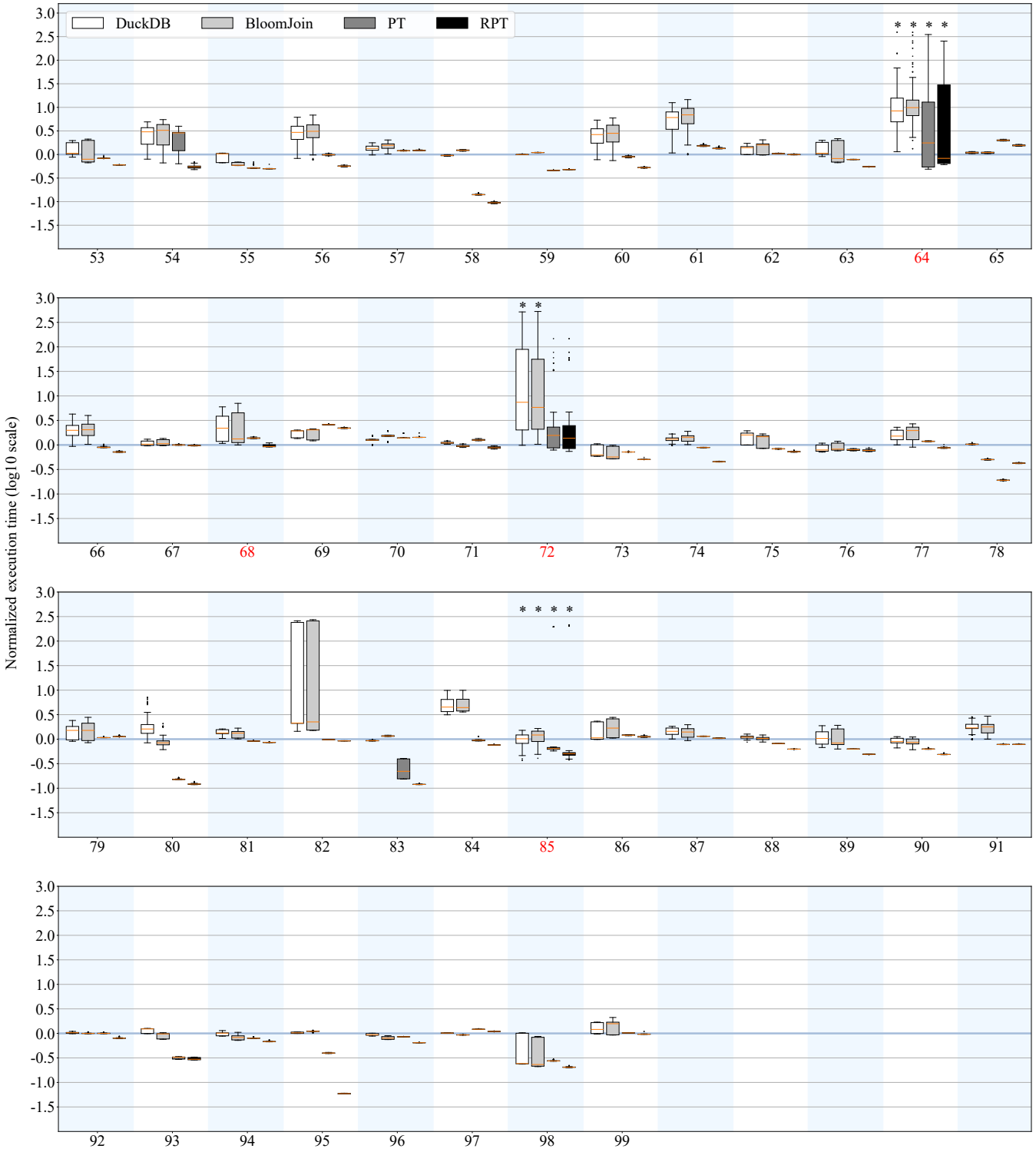


Figure 12: The distribution of execution time with random left deep plans for each query (53 - 99) in DSB – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

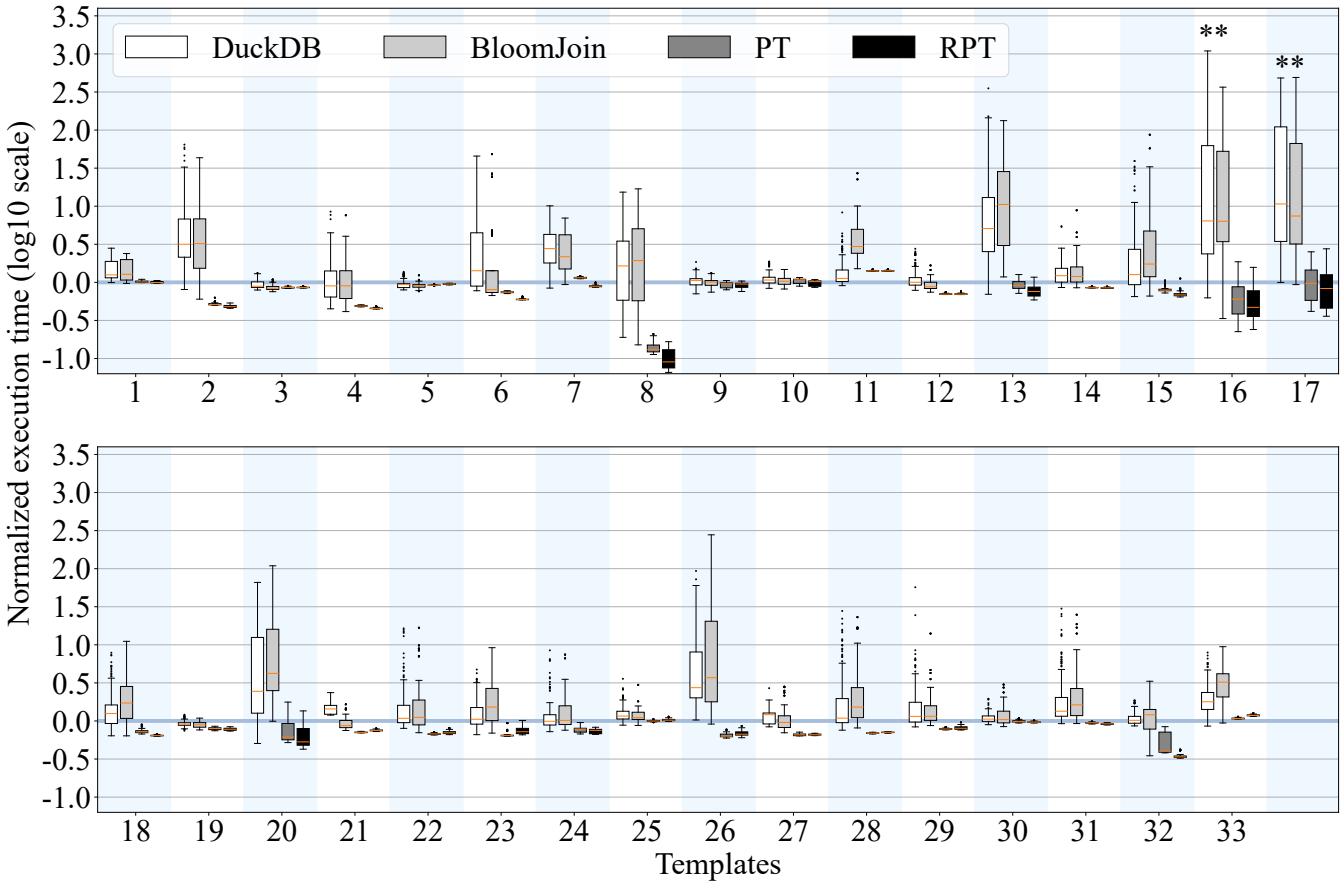


Figure 13: The distribution of execution time with random bushy plans for each query in JOB – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers).

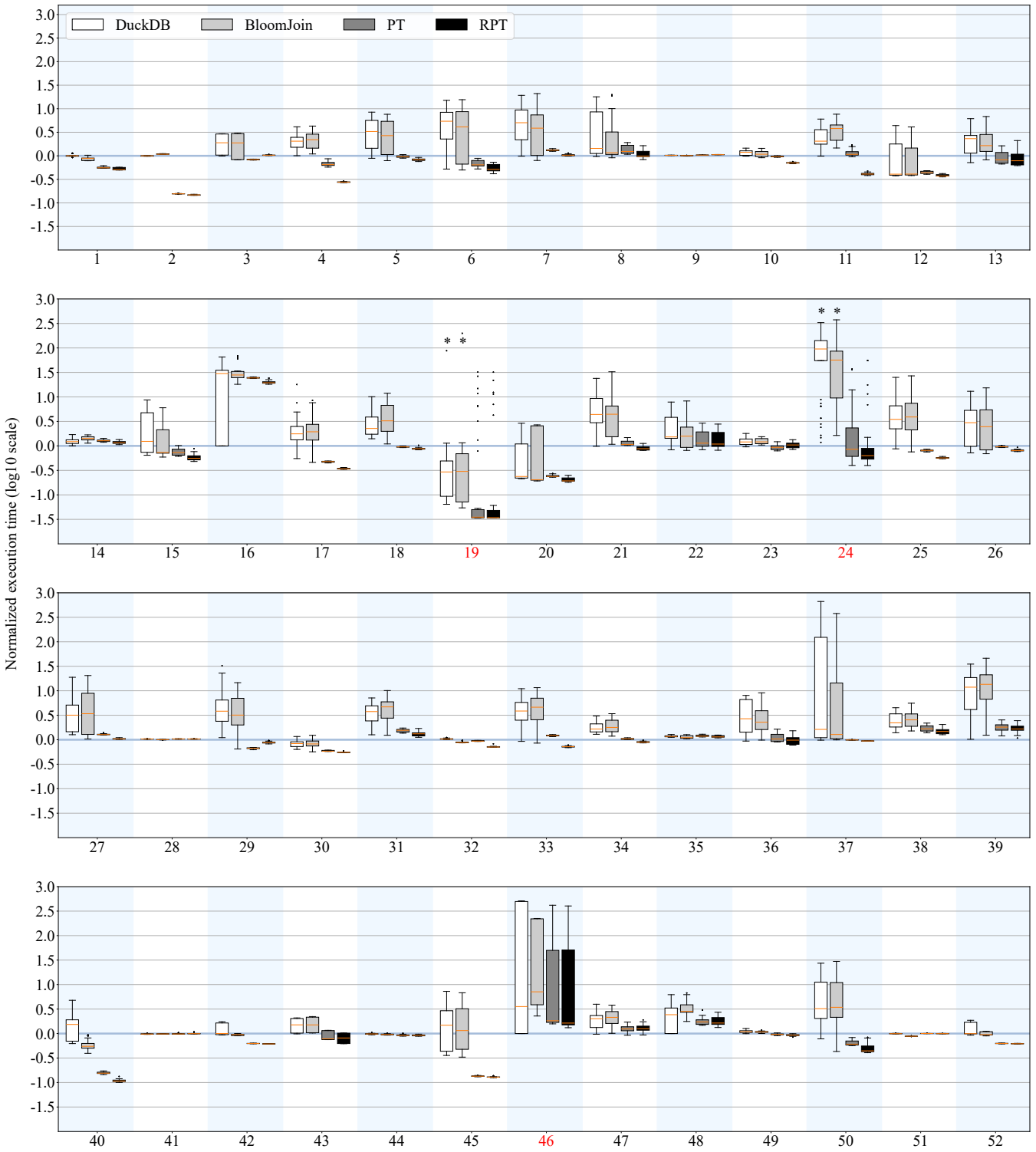


Figure 14: The distribution of execution time with random bushy plans for each query (1-52) in TPC-DS – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

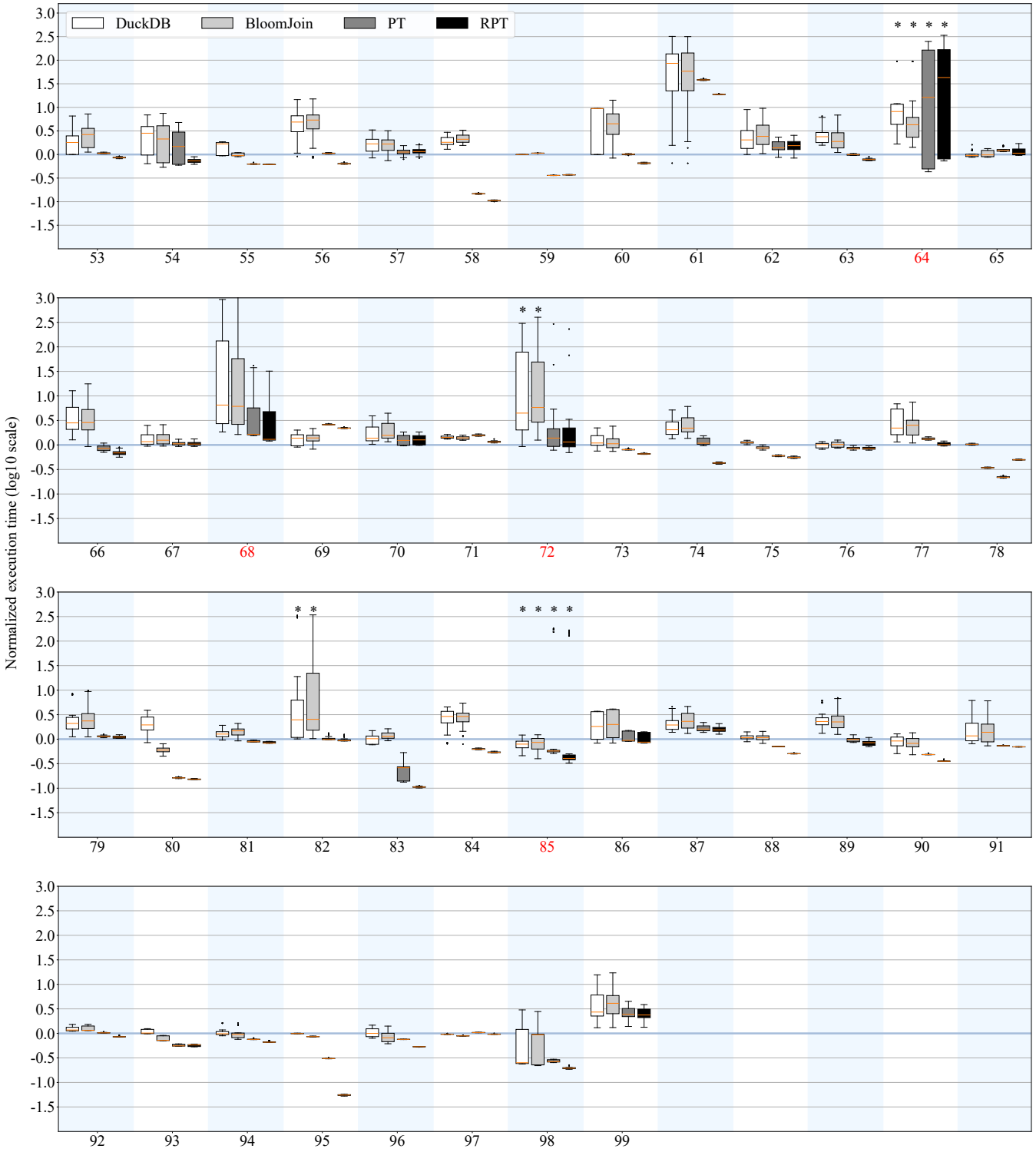


Figure 15: The distribution of execution time with random bushy plans for each query (53-99) in TPC-DS – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

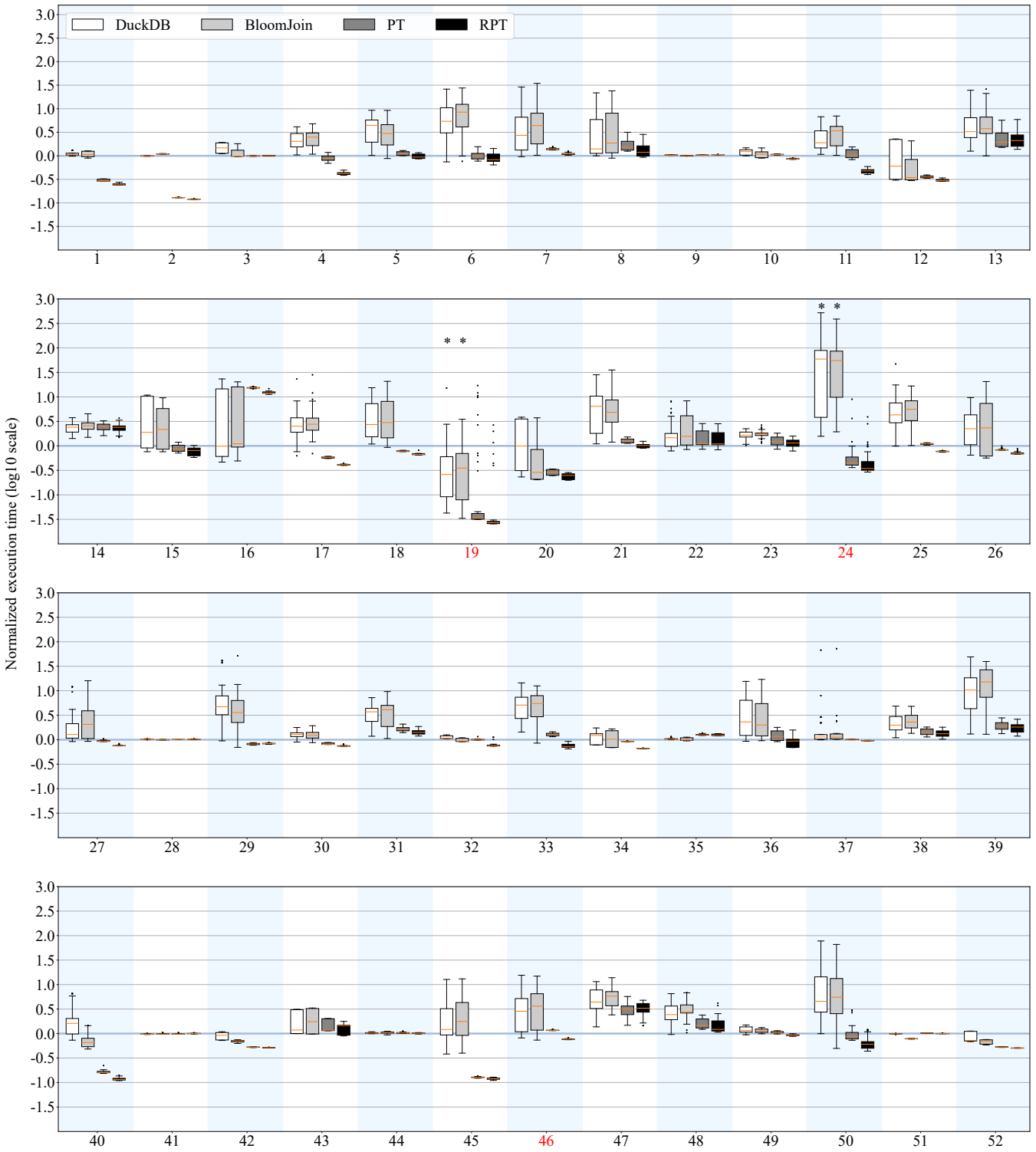


Figure 16: The distribution of execution time with random bushy plans for each query (1 - 52) in DSB – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.

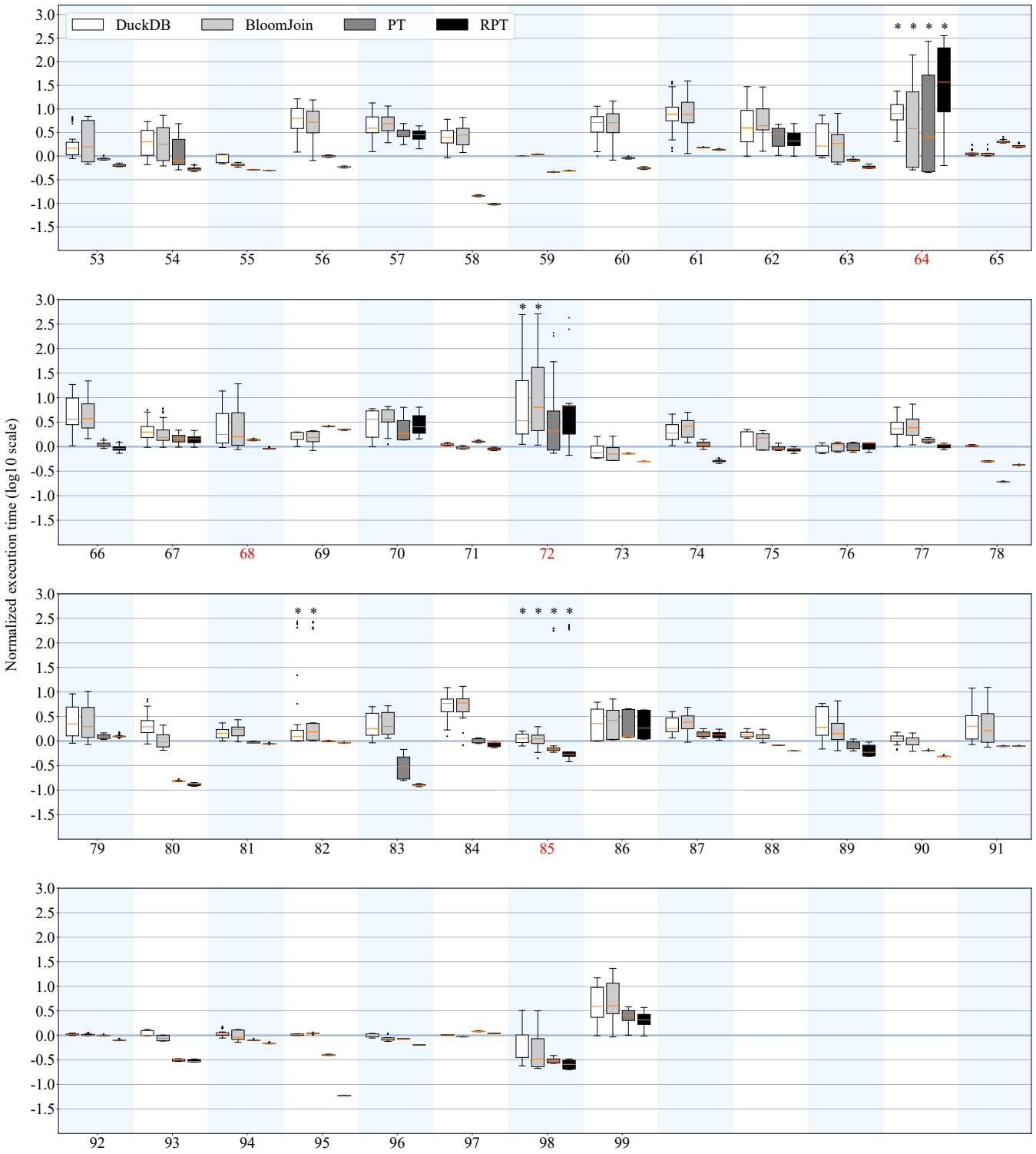


Figure 17: The distribution of execution time with random bushy plans for each query (53 - 99) in DSB – Normalized by the execution time of default DuckDB. The figure is log-scaled. The box denotes 25- to 75-percentile (with the orange line as the median), while the horizontal lines denote min and max (excluding outliers). '*' indicates timeouts. Cyclic queries are in red.