Supplementary Material – Hype or Heuristic? Quantum Reinforcement Learning for Join Order Optimisation

Maja Franz
Technical University of Applied Sciences Regensburg
Regensburg, Germany
maja.franz@othr.de

Tobias Winker
University of Lübeck
Lübeck, Germany
t.winker@uni-luebeck.de

Sven Groppe
University of Lübeck
Lübeck, Germany
sven.groppe@uni-luebeck.de

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens AG, Technology
Regensburg/Munich, Germany
wolfgang.mauerer@othr.de

***Abstract*—This document provides additional information and experiments conducted for the paper "Hype of Heuristic? Quantum Reinforcement Learning for Join Order Optimisation".**

I. Introduction

In particular, we address the following:

1) We provide an outline of the replication of the classical reinforcement learning (RL) approach by Marcus and Papaemmanouil [1] for the join order (JO) problem in Sec. II.
2) The process of finding good hyperparameters for each setting is described in Sec. III. We also list the parameters, which lead to the training runs depicted in the main study.

II. Classical Baseline Replication

While the main study focuses on the experimental results of the quantum-based version of *ReJoin*, this section is an extension of Sec. IV-C (methodology of the multistep QRL approach) and Sec. VB (experimental results) in the main study with a focus on the classical baseline replication.

A. Methodology

1) Classical Model: As in the original study [1], we utilise two fully-connected NNs, one for the actor-part and one for the critic part of PPO, using two hidden layers with 128 units each.

2) Training and Test Data: For classical *ReJoin*, we trained and evaluated the model using 113 queries from the join order benchmark (JOB). We applied a ten-fold cross-validation scheme [2], whereby the dataset is split into ten distinct parts. Each part is excluded from the training set once to be utilised for testing, leading to ten different train-test-splits. While *ReJoin* [1] selects ten test queries for evaluation and the remaining 103 queries for training on a random basis, we adopt ten-fold cross-validation, which is a commonly accepted to demonstrate generalisability [3].

As discussed in the main study, we also conduct experiments on a limited number of relations in a query, resulting in fewer queries in the dataset. Similar to Krishnan *et al.* [4], we generate new queries based on subplans to enlarge the dataset. However, instead of obtaining subplans from the traditional optimiser, we rely on a single *ReJoin* training run, generating over 12 000 subqueries. Note that the newly generated dataset is not used in our classical baseline replication experiments described in this supplementary material, but in the main study.

3) Setting the Join Order: To specify the selected join order for the *PG* cost model, we utilise the `pg_hint_plan` [5] extension that allows to control the execution plan of a query with hinting phrases. A join order can then be selected using RL, and the rest of the query plan choices are left to the traditional optimiser[1].

4) Reward Signal: In the original study [1], the agent is rewarded with zero for each intermediate step and with the reciprocal of the final join order's cost, that is, $R_t^{(\text{rec})} = \frac{1}{C_{\text{RL}}^{(\text{CM})}}$ in the final timestep T of an episode. Here, $C_{\text{RL}}^{(\text{CM})}$ represents the costs assigned to the join order selected by the RL agent according to the cost model CM. Since we initially achieved a subpar training convergence for $R_t^{(\text{rec})}$, inspired by the implementation of another RL technique for JO, *RTOS* [7], we used a different reward signal in our experiments:

$$R_t^{(\text{mrc})} = -\min\left[\log_{10}\left(\frac{C_{\text{RL}}^{(\text{CM})}}{C_{\text{DP}}^{(\text{CM})}}+1\right), 4\right] + \log_{10}(2)+1 \quad (1)$$

where $C_{\text{DP}}^{(\text{CM})}$ is the cost of the best join order determined by a dynamic programming (DP) exhaustive search according to cost model CM. The logarithm allows for expressing a large range of costs and additionally clipping the ratio to be at most 4 and shifting it with $log_{10}(2)+1$ to be at most 1 reduces the chances of steeper gradients during training, which is a known problem that can lead to suboptimal training [8].

5) Cost Models: We investigate the influence of two different cost models:

[1]For our replication of Ref. [1], we relied on explicit join clauses [6] using PostgreSQL V8.4, as in the original study

a) PG: The first model is the default cost model of PostgreSQL [6][2], which calculates a cost estimate for a selected JO and its correspondence query plan based on metadata, such as cardinalities and selectivities. PostgreSQL's built-in optimiser also uses these costs estimates to select a JO. However, because of errors in cost estimation, a selected JO is not necessarily optimal, even when using an exhaustive search based on DP. Additionally, since DP is resource-intensive, PostgreSQL switches from DP to a genetic optimiser for queries exceeding twelve relations per default. In the case of using (Q)ML for the JO selection, the complete query plan is established by sending the resulting JO to the PostgreSQL optimiser, which chooses the remaining optimisation decisions, such as whether to apply a hash or nested-loop join. Lastly, the PostgreSQL cost model assigns a cost to the resulting plan. Note that the *PG* cost model is mainly considered for comparability with the original study and is not used for our quantum-based approach in the main study.

b) OUT: To evaluate the selected join order, we use the established cost function C_{out} [9], which considers the cardinalities (*i.e.* the number of tuples in a query result set) as an approximation of query complexity:

$$C_{\text{out}}(T) = |T| + C_{\text{out}}(T_1) + C_{\text{out}}(T_2), \tag{2}$$

where n is the maximum number of joins in the query, a join tree is defined as $T = T_1 \bowtie T_2$, and $|T|$ represents the true cardinality of T ($C_{\text{out}}(T) = 0$ if $T \in \{r_1, r_2, \dots\}$ is a leaf). To obtain the optimal join orders for this reference cost model, we compute C_{out} for all possible orders (excluding cross joins) and select one with the lowest cost. Due to the high resource requirements of DP, we start with the join order suggested by PostgreSQL for more than twelve joins, and update it whenever a better order is found during training.

B. Results

Our attempt at replicating the approach and experiments of Ref [1] are illustrated on the left of Fig. 1. Despite some minor deviations from the findings in Ref. [1], which claim that *ReJoin* can on average reach 20% cheaper plans than PostgreSQL's genetic optimiser *GEQO* on the test dataset after approximately 10 000 sampled episodes, we believe that our results constitute a successful effort of replicating *ReJoin*. The evaluation of our replication confirms that *ReJoin* can learn near-optimal join orders, similar to *GEQO*, and even match the optimal orders of DP in some cases. The deviations from the original results could possibly be attributed to differing hyperparameters or settings. As unfortunately the hyperparameters are not specified in Ref. [1], we conducted an extensive hyperparameter search, for which individual runs and the resulting parameters can be found in Sec. III. Nevertheless, we cannot guarantee exact agreement with the original study.

[2]For our baseline comparison, we use PostgreSQL V8.4 to ascertain compatibility with the original experimental setting, while V16.0 is utilised in subsequent experiments.

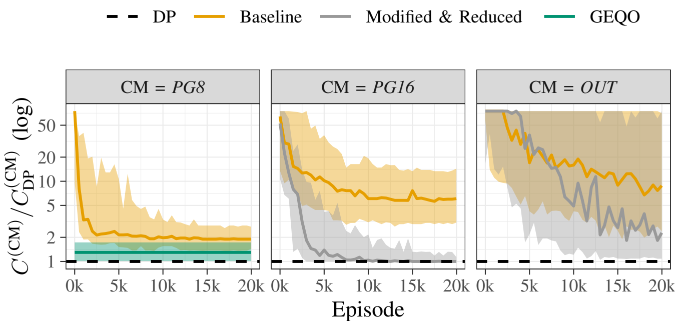

Fig. 1. Relative costs over all queries in the JOB during training of reimplemented baseline and the modified/reduced version of *ReJoin* using the cost models of PostgreSQL V8.4 (*PG8*) and V16.0 (*PG16*), as well as C_{out} (*OUT*). The models were trained according to ten-fold cross-validation, such that each query is in a test-set once. We executed PostgreSQL V8.4's genetic optimiser *GEQO* on every query in the JOB. The solid lines represents the medians, the areas corresponds to the first and third quartiles. To improve the visual display, results exceeding a value of 75 are cut off.

C. Baseline Modifications

To further enhance the outcomes of our replication, and to allow for the reduced encoding described in the main study in Par. IV-C2a, we combined methods from other RL approaches for JO [4], [10] to improve the learning convergence in cost training.

1) Incorporating Orders from the Traditional Optimiser: In our previous evaluations of our replication, we found that throughout the whole training process, actions were sampled for certain queries, resulting in suboptimal rewards and therefore in join orders leading to high costs. This indicates that the model may have become trapped in a local optimum. To address this, we select actions in the sampling stage of PPO based on the join order selected for a specific query by the traditional optimiser, which is easily obtainable and near-optimal. A similar technique is utilised in Ref. [4], which is based on Q-learning, an RL approach that assimilates knowledge from stored experience in a *replay buffer*, from which is sampled to train the model. To pre-fill this buffer, Krishnan *et al.* employ (sub-)join orders that occur in the query plan of the traditional optimiser. As a replay buffer is not present in PPO, we sample actions from the traditional optimiser's join orders with probability $p_O \in [0, 1]$ in the sampling stage; p_O requires hyperparameter tuning.

2) Updating the Training Dataset based on Rewards: As mentioned above, basic *ReJoin* fails to find good join orders for specific queries. One possibility for learning better join orders for these "hard" queries is to increase the amount of samples taken. Ref. [10] adds multiple copies of a "hard" query to the training set based on the reward associated with the join order that can be achieved with the current policy. We update the training dataset every $s_U \in [0, s_{\max}]$ episodes, where $s_{\max} \in \mathbb{N}$ is the maximum number of sampled episodes. In the update, a copy of each query achieving a reward below the threshold $th_U \in \mathbb{R}$ is added to the training dataset. This introduces hyperparameters s_U and th_U.

3) Multi-Step Reward Signal: In previous studies on RL for JO (*e.g.*, Refs. [1], [4], [10]) the reward, as function of cost, is only assigned at the end of each episode when the full join order is built by the RL policy. Intermediate steps receive a zero reward. This seems counter-productive, given that one property of RL is to determine an action based on a current state and reward signal. Assuming the cost difference $C_t^{(\mathrm{CM})}$ between timesteps t and $t-1$ with costs c_k for subtrees $T_k \in \mathcal{F}$ in a state S_t according to cost model CM is

$$C_t^{(\mathrm{CM})} = \begin{cases} \sum_{T_k \in \mathcal{F}} c_k - C_{t-1}^{(\mathrm{CM})} & \text{if } t > 0 \\ 0 & \text{if } t = 0 \end{cases}, \tag{3}$$

and the cost assigned to the order of the full query determined by DP is $C_{\mathrm{DP}}^{(\mathrm{CM})}$, we propose the clipped reward at t as

$$R_t^{(\mathrm{mult})} = \frac{1}{n-1}\left[-\min\left(\frac{C_t^{(\mathrm{CM})}}{C_{\mathrm{DP}}^{(\mathrm{CM})}}, n-1\right) + 2\right]. \tag{4}$$

This requires $(n-1)$ joins (and actions) to build the join tree for a query with n relations. As for $R_t^{(\mathrm{mrc})}$, shifting and normalising reduces the chance for steep gradients. We introduce whether the multi-step or single-step reward signal (cf. Sec. II-A4) should be used as an additional hyperparameter.

The experiments involving the application of modifications and reduction of input feature space, as discussed in Par. IV-C2a in the main study, were conducted using PostgreSQL V16.0's cost model due to its compatibility with the `pg_hint_plan` extension. This extension facilitates simpler manipulation of join plans than explicit join clauses. The outcomes of the modified and reduced *ReJoin*, as well as the baseline *ReJoin* on the two cost models *PG*, with PostgreSQL V16.0, and *OUT* are depicted in Fig. 1[3]. According to the figure, the modified and reduced *ReJoin* version can achieve lower costs than the baseline, with considerably fewer input components. This confirms that the reduced *ReJoin* can serve as classical basis for subsequent QRL calculations. Furthermore, we observed that training convergence is particularly sensitive to hyperparameters. For example, in our top-performing test of the baseline version in PostgreSQL V8.4, we utilised a low learning rate (*i.e.* optimiser stepsize) of 8e-5. However, costs for *OUT* using the modified version significantly varied with this rate, and we settled on a learning rate of 3e-4 (cf. Sec. III for hyperparameters for all configurations). Despite various efforts to find similar optimal hyperparameters for the *OUT* cost-model, both configurations are less stable in training. Higher costs for *OUT* compared to *PG* may be a reason for this observation. The maximum relative costs across all experiments in Fig. 1 were about 500 000 times higher for *OUT* than for *PG*. By clipping the calculated reward signal, we avoid steep gradients and retain small median costs for the modified and reduced version. Yet significant variations in cost range leave several outliers.

III. Hyperparameter Search

All training runs undertaken in this study underwent a comprehensive hyperparameter search. To achieve this, hyperparameters were fine-tuned in four steps:

1) Firstly, we determined hyperparameters of interest and a possible selection of values, listed in Tab. I.
2) As testing over 8 000 combinations of hyperparameters in a grid search is too extensive, we randomly selected 50 of all configurations on the specified training and test dataset in Ref. [1][4] for each cost model (*PG8*, *PG16*, *OUT*) and setting (Baseline, Modified & Reduced). .
3) While we can not guarantee that we identified the best hyperparameter constellation, as we were only able to test a part of the possible constellations, we collected the four most efficient hyperparameter combinations, meaning those that resulted in the lowest average costs over the last three validation steps. These top-performing combinations were subsequently utilised for the 10-fold cross-validation.
4) Finally, we selected the combination with the lowest average costs over the last three validation steps and all queries in the test sets as the optimal one.

The final parameters for each configuration are listed in Tab. II. The raw results can also be found in this corresponding reproduction package.

Acknowledgements MF, TW, SG and WM were supported by the German Federal Ministry of Education and Research (BMBF), funding program "Quantum Technologies—from Basic Research to Market", grants #13N15647 and #13NI6092 (MF and WM), and #13N16090 (TW and SG). WM acknowledges support by the High-Tech Agenda Bavaria.

References

[1] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration," in *Proc. of the First Int. Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2018, ISBN: 9781450358514. DOI: 10.1145/3211954.3211957.

[2] T. Fushiki, "Estimation of prediction error by using k-fold cross-validation," *Statistics and Computing*, vol. 21, pp. 137–146, 2011.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (Adaptive computation and machine learning). 2016, ISBN: 9780262035613.

[4] S. Krishnan *et al.*, "Learning to optimize join queries with deep reinforcement learning," 2018. DOI: 10.48550/arXiv.1808.03196.

[5] NTT OSS Center DBMS Development Team. "Pg_hint_plan PostgreSQL extension." (2023), [Online]. Available: https://pghintplan.osdn.jp/pg_hint_plan.html.

[6] PostgreSQL Developers. "PostgreSQL." (2023), [Online]. Available: https://www.postgresql.org.

[7] X. Yu *et al.* "Github repository: AI4DBCode." Commit: a8989bfa. (2022), [Online]. Available: https://github.com/TsinghuaDatabaseGroup/AI4DBCode.

[8] A. Laud and G. DeJong, "The influence of reward on the speed of reinforcement learning: An analysis of shaping," in *Proc. of the 20th Int. Conf. on Machine Learning (ICML-03)*, 2003, pp. 440–447. DOI: 10.5555/3041838.3041894.

[3]To assure a fair comparison, we used a similar number of parameters for both, the full and the reduced *ReJoin* versions, resulting in two fully-connected layers with each 128 or 384 hidden units, respectively.

[4]For the training runs in the main study on four relations, we conducted the initial hyperparameter search on the first train-test split

TABLE I
HYPERPARAMETERS SUBJECT TO SEARCH.

Hyperparameter	**Variable name**	**Description**	**Possible values**
p_O	`take_best_threshold`	Probability of sampling optimal actions from the traditional optimiser (modified *ReJoin* only, cf. Sec. II-C1).	{ 0.1, 0.2, 0.5 }
s_U	`take_best_frequency`	Number of episodes after which to update the dataset based on rewards (modified *ReJoin* only, cf. Sec. II-C2).	{ 10, 50, 100, 500, 1 000 }
th_U	`update_dataset_reward_threshold`	Threshold, below which rewards are added to the train set every s_U episodes (modified *ReJoin* only, cf. Sec. II-C2).	{ -1, 0, 0.5 }
η_{start}	`lr_start`	Initial value for the learning rate.	{ 5e-5, 8e-5, 9e-5, 1e-4, 2e-4, 3e-4 }
η_{duration}	`lr_duration`	Share of the training process for which the learning rate decays linearly until a learning rate of 1e-8 is reached. A value of zero corresponds to no decay in the learning rate.	{ 0, 0.5, 0.9 }
multistep reward	`multistep`	Whether to use a multi-step (modified *ReJoin* only, cf. Sec. II-C3) or single-step (cf. Sec. II-A4) signal.	{ True, False }
n_{episodes}	`batchsize`	Number of episodes per batch in PPO (cf. Ref. [11]).	{ 5, 10, 20, 50, 100 }

TABLE II
FINAL HYPERPARAMETER VALUES FOR THE TRAINING RESULTS IN THIS SUPPLEMENTARY MATERIAL AND IN THE MAIN STUDY.

Configuration	**Cost model**	**Results**	**Hyperparameter**	**Value**
Classical – Baseline	*PG8*	Fig. 1, left		
			η_{start}	9e-5
			η_{duration}	0.9
			n_{episodes}	20
Classical – Baseline	*PG16*	Fig. 1, mid		
			η_{start}	9e-5
			η_{duration}	0.9
			n_{episodes}	20
Classical – Modified & Reduced	*PG16*	Fig. 1, mid		
			p_O	0.1
			s_U	1
			th_U	0.5
			η_{start}	3e-4
			η_{duration}	0.9
			multistep reward	False
			n_{episodes}	20
Classical – Baseline	*OUT*	Fig. 1, right		
			η_{start}	5e-5
			η_{duration}	0.9
			n_{episodes}	20
Classical – Modified & Reduced	*OUT*	Fig. 1, right		
			p_O	0.1
			s_U	1
			th_U	0.5
			η_{start}	3e-4
			η_{duration}	0.9
			multistep reward	False
			n_{episodes}	20
Quantum & Classical – Modified & Reduced – Four Relations	*OUT*	Main study, Figs. 5 & 6		
			p_O	0.1
			s_U	1
			th_U	0.5
			η_{start}	3e-4
			η_{duration}	0.9
			multistep reward	True
			n_{episodes}	20

[9] S. Cluet and G. Moerkotte, “On the complexity of generating optimal left-deep processing trees with cross products,” in *Proc. of the 5th Int. Conf. on Database Theory*, 1995, pp. 54–67, ISBN: 3540589074. DOI: 10.1007/3-540-58907-4_6.

[10] X. Yu *et al.*, “Reinforcement learning with tree-lstm for join order selection,” in *2020 IEEE 36th Int. Conf. on Data Engineering (ICDE)*, 2020, pp. 1297–1308. DOI: 10.1109/ICDE48307.2020.00116.

[11] J. Schulman *et al.*, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].