# Solving the Pocket Cube with Deep Approximate Value Iteration: A Reproduction with Exact Ground Truth

**Gabriel Caballero**
School of Computer Science
McGill University
gabriel.caballero@mail.mcgill.ca

## Abstract

I reproduce the Deep Approximate Value Iteration (DAVI) algorithm of Agostinelli et al. [1] on the $2\times2$ Rubik's cube ("Pocket Cube"), a combinatorial puzzle small enough to admit exact value iteration as a tractable ground truth. The Pocket Cube has $3{,}674{,}160$ reachable states and a worst-case optimal solve length of 14 quarter-turn moves (*God's number*); a single breadth-first search from the goal computes the exact cost-to-go function $V^\star$ in seconds. I use $V^\star$ to (i) score how closely DAVI's neural estimate $V_\theta$ approaches the true value function and (ii) measure how close batch weighted A* search (BWAS) using $V_\theta$ gets to optimal solution lengths. These comparisons are infeasible at $3\times3$ scale, where $V^\star$ has $4.3 \times 10^{19}$ entries. I also compare against an uninformed forward breadth-first search baseline. With $60{,}000$ training iterations on a single laptop GPU (Apple M1 Pro / Metal Performance Shaders, with best-checkpoint selection on a held-out eval set), DAVI reaches a heuristic with mean absolute error of $0.83$ moves against $V^\star$ ($-0.5$ bias). Across 3 evaluation seeds (90 states per scramble depth, $1{,}260$ total), BWAS using this heuristic solves every state and recovers a shortest path on $99.6\%$ of them; the remaining $0.4\%$ are off by one move at depths 13–14 (mean excess $0.04$ and $0.02$ respectively). Weighted A* with $\lambda<1$ is not optimality-preserving in general, so the near-optimality is a property of the trained heuristic on this test distribution rather than a structural guarantee. BWAS expands roughly $730\times$ fewer nodes than uninformed BFS at the deepest depth where the latter is tractable.

## 1 Introduction

The Rubik's cube is a deterministic shortest-path problem with a vast state space, a single goal, and the property that random play almost never reaches the goal. Agostinelli et al. [1] introduced DeepCubeA, which trains a neural network $V_\theta$ to approximate the optimal cost-to-go function via Deep Approximate Value Iteration (DAVI) and uses $V_\theta$ as a heuristic in batch weighted A* search (BWAS). DeepCubeA solves the full $3\times3$ cube and produces a shortest path in $60.3\%$ of test states.

Two questions go unaddressed at $3\times3$ scale: *how accurate is the learned heuristic compared to the true cost-to-go?* and *how close to optimal are BWAS solutions on average?* On the $3\times3$, $V^\star$ has $4.3 \times 10^{19}$ entries and cannot be stored. Optimal solvers are pattern-database–based and limited to specific subsets of states. I address these questions by reproducing DAVI on the $2\times2$ Pocket Cube, whose state space is $3{,}674{,}160$ states, small enough that one breadth-first search from the goal yields $V^\star$ exactly in under ten seconds and fits in 4 MB of memory. Every claim about $V_\theta$ can therefore be checked directly, and every BWAS solution can be compared to a known optimum.

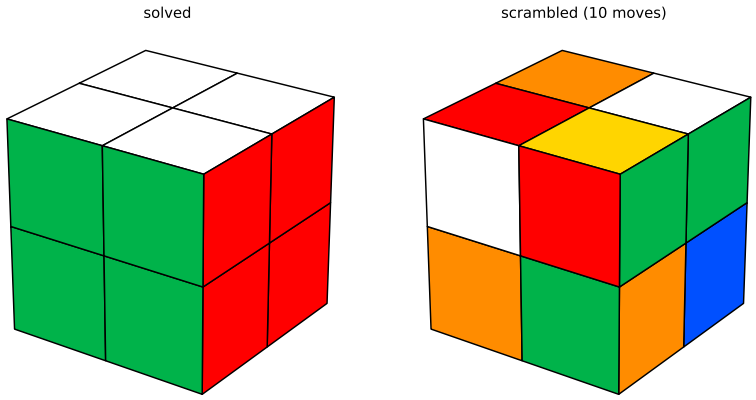


Figure 1: The 2×2 Rubik's cube ("Pocket Cube"). Left: the solved state, the unique goal of the search. Right: a state after 10 random quarter-turn moves. The puzzle has 3,674,160 reachable states (with one corner pinned in space), and the worst-case state is 14 moves from solved.

**Contributions.** A from-scratch reproduction of DAVI and BWAS specialised to the 2×2, with full code and exact $V^\star$; a direct comparison of three solvers on a per-scramble-depth test set (DAVI+BWAS, uninformed BFS, and an optimal-policy oracle following $\arg\min_a V^\star$); and a supervised baseline that trains the same architecture on $(s, V^\star(s))$ pairs, which gives DAVI a tight upper bound and measures the cost of bootstrapping.

## 2 Background

**The pocket cube.** The 2×2 cube has 8 corner cubies and no edges or centres (Figure 1). With one corner pinned in space (the standard "DLB-fixed" normalisation, factoring out whole-cube rotations), the remaining 7 corners give $7! \cdot 3^6 = 3{,}674{,}160$ reachable states. The action space is 6 quarter-turn moves: $\{U, U', R, R', F, F'\}$; the moves $\{D, L, B\}$ would move the fixed corner and are equivalent to the allowed moves under whole-cube rotation. I work throughout in the *quarter-turn metric* (QTM, where only 90° turns count as one move); under the alternative *half-turn metric* (HTM, where $U^2$ counts as one move) the action space and worst-case solve length differ. *God's number*, the worst-case optimal solve length over all states, is 14 in QTM and 11 in HTM for the 2×2 [2].

**Value iteration on the cube.** Solving the cube is a deterministic shortest-path MDP with unit move cost and a single absorbing goal. The optimal cost-to-go satisfies the Bellman equation $V^\star(s) = \min_a [1 + V^\star(\mathrm{succ}(s, a))]$ for $s \neq \mathrm{goal}$ and $V^\star(\mathrm{goal}) = 0$. For graphs of this kind, value iteration from the goal coincides with breadth-first search from the goal: each iteration of BFS finds states at the next distance level, and $V^\star(s)$ equals the BFS depth at which $s$ is first encountered.

**Deep approximate value iteration (DAVI).** When the state space is too large to tabulate (the 3×3), DeepCubeA replaces the table with a parameterised function $V_\theta : \mathcal{S} \to \mathbb{R}$ and trains it via the Bellman target

$$y(s) = \begin{cases} 0 & s = \mathrm{goal} \\ 1 + \min_a V_{\theta^-}(\mathrm{succ}(s, a)) & \text{otherwise} \end{cases} \tag{1}$$

where $\theta^-$ is a frozen target-network copy of $\theta$. Training data is generated in reverse: starting from the solved state, $k$ random moves are applied with $k$ uniform in $\{1, \ldots, K\}$. This distribution propagates information from the goal outward and avoids the dead-end problem of forward random play. The target network is periodically copied from the live network.

**Batch weighted A* search (BWAS).** At test time, $V_\theta$ is used as a heuristic in weighted A* with priority $f(s) = \lambda \cdot g(s) + h(s)$. The "batch" modification expands the $N$ best-priority nodes per iteration so that all $6N$ children can be evaluated by the network in a single batched forward pass. I use $\lambda = 0.6$ following [1] and $N = 10$ (see Appendix A for the choice of $N$).

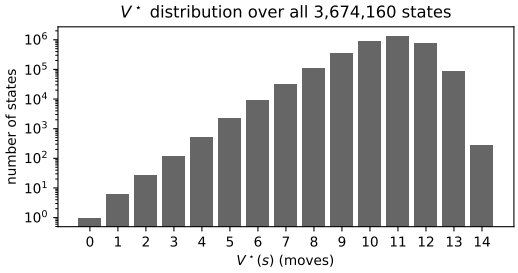


Figure 2: $V^\star$ distribution over all $3{,}674{,}160$ states of the Pocket Cube. Y-axis is logarithmic.

## 3 Methodology

**State representation and network.** The 24-element sticker representation (port from py222 [4]; one colour in $\{0, \ldots, 5\}$ per face) is converted to a 144-dimensional one-hot for the network. A canonical state ID in $[0, 3{,}674{,}160)$ is computed from the corner permutation and orientation. The network is a scaled-down DeepCubeA: a $144 \to 1024 \to 256$ stem with batch normalisation and ReLU, followed by 2 residual blocks of width $256$ and a single linear head.

**Training and evaluation.** DAVI trains for $60{,}000$ iterations at batch size $10{,}000$. Each iteration samples states by reverse-scrambling the solved cube $k$ moves, $k$ uniform in $\{1, \ldots, 15\}$, with no immediate inverse moves. The target network is hard-copied every $1{,}000$ iterations; Adam at $\eta = 10^{-3}$. I use *best-checkpoint selection*: the network state with the lowest evaluation MAE seen during training is saved and used downstream. The evaluation test set is $30$ random scrambles per depth $k \in \{1, \ldots, 14\}$ at one seed, $420$ states; for the headline numbers I run three independent test sets at distinct seeds, $1{,}260$ states total. I run three solvers on every state (the optimal-policy oracle following $\arg\min_a V^\star(\mathrm{succ}(s, a))$, uninformed BFS from start, and DAVI+BWAS), and report solve rate, mean solution length, mean excess over optimal, and mean nodes expanded.

## 4 Results

**Exact value function.** A single BFS from the goal traverses all $3{,}674{,}160$ reachable states with maximum depth $14$ in approximately $9$ seconds on a single CPU core. The depth distribution (Figure 2) is heavily concentrated at depths 10–11: $62\%$ of states have $V^\star(s) \in \{10, 11\}$. Only $276$ states are at the maximum depth, and only $1$ state (the solved state) is at depth $0$.

**Heuristic accuracy.** I track $V_\theta$'s mean absolute error against $V^\star$ during training on a fresh sample of $5{,}000$ states drawn uniformly over the $3{,}674{,}160$-state space (heavily weighted toward $V^\star \in \{10, 11\}$ by the natural distribution). The MAE starts near $10.7$ moves (the network's initial output is near zero), decreases sharply for the first $\sim 10{,}000$ iterations as the target horizon extends from the goal outward, then more slowly until $\sim 25{,}000$ iterations, and then enters a noisy plateau where the eval MAE oscillates between $0.83$ and $1.05$ for the rest of training. I use *best-checkpoint selection*: the model state with the lowest eval MAE seen during training is saved. Best MAE was $0.83$ at iteration $49{,}000$, with $-0.5$ bias and RMSE $\approx 1.07$. The plateau itself, with the live network never settling, is the expected fingerprint of bootstrapping from a moving target. End-of-training values would have been $\sim 1.0$ MAE. Figure 3 plots predictions against $V^\star$ on the $1{,}260$-state multi-seed test set; per-iteration training curves are in Appendix B.

**Supervised baseline (cost of bootstrapping).** Because $V^\star$ is tabulated, I can train an identical network *supervised* on $(s, V^\star(s))$ pairs. To make this a fair comparison, supervised uses the same training-state distribution as DAVI (reverse-scrambling $k$ uniform in $[1, 15]$); the only difference is the target ($V^\star(s)$ vs the Bellman target). With $20{,}000$ iterations at the same batch size, supervised reaches MAE $= 0.74$, RMSE $= 0.94$, an $\sim 11\%$ reduction in MAE over DAVI's best $0.83$. The gap is the *cost of bootstrapping*, the price DAVI pays for not having $V^\star$ in hand at training time. This comparison is unique to a domain where $V^\star$ fits in memory.

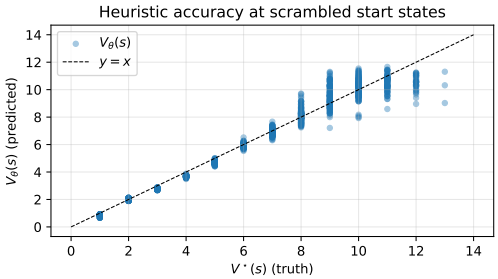


Figure 3: Predicted heuristic $V_\theta(s)$ against true $V^\star(s)$ on the 1,260-state multi-seed test set. The dashed line is $y = x$.

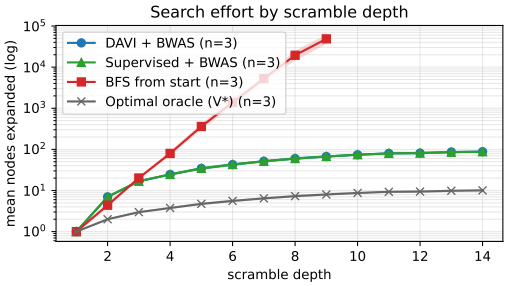


Figure 4: Mean nodes expanded vs scramble depth (log scale).

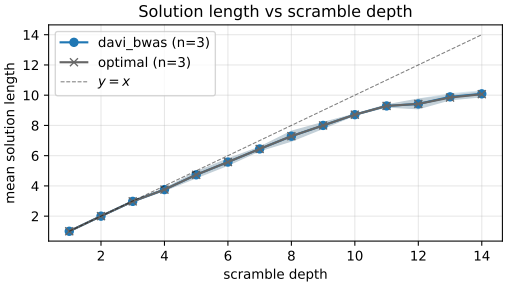


Figure 5: Mean solution length vs scramble depth.

I also report (Appendix E) on a naive supervised variant trained with uniform-over-states sampling: it achieves slightly lower aggregate MAE but is mis-calibrated on shallow states, illustrating that the training *distribution* matters as much as the target signal.

**Search performance by scramble depth.** Figure 4 compares nodes expanded by DAVI+BWAS, the supervised network used as a BWAS heuristic, and uninformed BFS-from-start, on a log scale, averaged over 3 evaluation seeds (1,260 states). BFS grows exponentially: by depth 9 it expands $\sim$49,000 nodes (mean across seeds); I do not run BFS-from-start at depths $\geq 10$ because uninformed forward search at this scale is intractable in a reasonable budget. DAVI+BWAS expands only $\sim$67 nodes at depth 9 and $\sim$88 at depth 14, a $\sim$730$\times$ reduction at depth 9, the deepest depth at which I can directly compare. The supervised heuristic produces near-identical search trajectories.

Figure 5 reports mean solution length. DAVI+BWAS solves every one of the 1,260 test states; mean excess over $V^\star$ is 0 at depths 1–12, $+0.04$ at depth 13, and $+0.02$ at depth 14 (Table 1). Supervised+BWAS achieves 0 excess at every depth. Weighted A* with $\lambda < 1$ does not guarantee optimality even with an admissible heuristic, and DAVI's small non-zero excess at the deepest scrambles is the empirical price: a handful of states where the heuristic's noise flips the priority ordering and BWAS commits to a one-move-longer path. The supervised heuristic, with its smaller noise, escapes this. On this scale, the heuristic's *ordering* of children matters more than its absolute calibration. The gap in raw MAE (0.83 vs 0.74) is much larger than the gap in search behaviour, which appears only at the very tail of the depth distribution.

**Related learning on the 2$\times$2.** Konen 2022 [2] reports near-100% solve rate on the same QTM 2$\times$2 using TD-n-tuple learning with MCTS-wrapped test-time search; the headline metric (policy reaches goal within 50 moves on 200 scrambled cubes per depth) and the search procedure (MCTS from the start) both differ from those used here, so I anchor against Konen's setup without claiming a direct comparison. The contribution of this work is the orthogonal $V^\star$ axis: at this scale I measure heuristic quality and BWAS optimality exactly, which neither Konen 2022 nor the original DeepCubeA do.

| | Optimal oracle | | BFS from start | | DAVI + BWAS | | |
|---|---|---|---|---|---|---|---|
| depth | solved | nodes | solved | nodes | solved | nodes | excess |
| 1 | 90/90 | 1 | 90/90 | 1 | 90/90 | 1 | +0.00 |
| 5 | 90/90 | 5 | 90/90 | 361 | 90/90 | 34 | +0.00 |
| 9 | 90/90 | 8 | 90/90 | 48,646 | 90/90 | 67 | +0.00 |
| 11 | 90/90 | 9 | – | – | 90/90 | 80 | +0.00 |
| 13 | 90/90 | 10 | – | – | 90/90 | 86 | +0.04 |
| 14 | 90/90 | 10 | – | – | 90/90 | 88 | +0.02 |

Table 1: Solver comparison on the per-depth test set, averaged over 3 evaluation seeds (90 states per depth, 1,260 total). BFS-from-start is not run at depths $\geq 10$ as uninformed forward search at this scale exceeds a reasonable budget. DAVI+BWAS solves every state; the small $+0.04/+0.02$ mean excess at depths 13/14 reflects rare suboptimal commitments under $\lambda$=0.6 (see §4). $N = 10$ for BWAS.

## 5 Conclusion and Future Work

I have reproduced DAVI on the Pocket Cube and shown that, even with a much smaller network than the original DeepCubeA ($\sim$679K parameters versus $\sim$9.2M, a 13$\times$ reduction), the learned cost-to-go $V_\theta$ tracks the exact $V^\star$ to within 0.83 moves (best-checkpoint MAE), and supports BWAS solutions that solve 100% of 1,260 test states with near-optimal length (mean excess $\leq 0.04$ moves at every depth). The Pocket Cube admits an explicit value-function ground truth, so every component of the algorithm can be checked against an exact reference.

**Limitations and future work.** My network plateaus at a non-zero MAE rather than fitting $V^\star$ exactly, with a state-independent $\sim -0.5$ bias. This is consistent with bootstrapping from a moving Bellman target. Hyperparameters (architecture, target-update period, training distribution) were not swept. Natural extensions include symmetry-aware training (exploiting the 24 colour-swap symmetries [2]), and scaling to 3$\times$3 subsets where pattern databases provide local ground truth.

## Acknowledgements

I thank Wolfgang Konen [2] for state-representation derivations and the maintainer of py222 [4] for an MIT-licensed reference implementation whose move tables and OP-hashing scheme I ported.

## References

[1] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi, "Solving the Rubik's Cube with Deep Reinforcement Learning and Search," *Nature Machine Intelligence*, vol. 1, pp. 356–363, 2019.

[2] W. Konen, "Towards Learning Rubik's Cube with N-tuple-based Reinforcement Learning," Technical Report, TH Köln, 2022.

[3] S. McAleer, F. Agostinelli, A. Shmakov, and P. Baldi, "Solving the Rubik's Cube Without Human Knowledge," *arXiv preprint arXiv:1805.07470*, 2018.

[4] MeepMoop, "py222: Python 2×2 Rubik's Cube representation & solver," GitHub repository, MIT License, 2022. `https://github.com/MeepMoop/py222`.

## A The choice of BWAS batch size $N$

DeepCubeA uses $N = 10{,}000$ on the 3$\times$3 cube. I initially followed this and found that BWAS expanded *identical* numbers of nodes regardless of which heuristic (DAVI, supervised, even $V^\star$ itself) was used. The cause: at $N = 10{,}000$ the BWAS inner loop pops effectively all of OPEN per iteration, then pushes $6N$ children. With the small initial frontier (the start state has only 6 successors), every iteration's pop is so wide that the heuristic plays no role in selection order; BWAS at this scale degenerates into pure BFS-from-start that simply terminates when it sees the goal. Lowering $N$ to 10

restores the heuristic's discriminating effect: at every step, only the ten lowest-$f$ nodes are expanded, so a heuristic that orders successors well prunes the search effectively. The order-of-magnitude reductions in nodes expanded reported in Section 4 are with $N = 10$.

## B Training curves

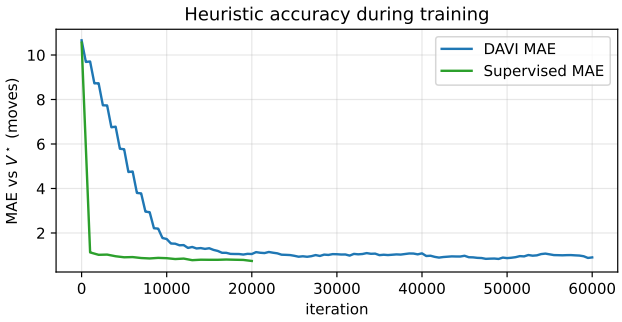


Figure 6: MAE of the heuristic against $V^\star$ across training iterations. The supervised baseline uses the same data distribution as DAVI but trains directly on $V^\star$ targets; it converges to a lower MAE.

DAVI's loss is noisier than supervised training because of target-network updates: each hard copy of the live network into the target network shifts the regression target, producing a small loss spike that the network catches up on over the next $\sim$1,000 iterations. The supervised baseline has no such spikes since its target ($V^\star$) is fixed.

## C Effect of the BWAS weight $\lambda$

I ablate the weighted-A* parameter $\lambda$ in $f(s) = \lambda g(s) + h(s)$ across $\{0.0, 0.3, 0.6, 1.0\}$ on the per-depth test set (30 states per depth, 1–14, $N = 10$, $10^5$-node budget). $\lambda = 0$ is greedy best-first on $h$ alone; $\lambda = 1$ is unweighted A*, which would be optimal if $h$ were admissible ($V_\theta$ is not, but the $\sim -0.5$ bias is roughly state-independent so $\lambda = 1$ behaves close to admissible in practice).

The headline result of the ablation is that on this test set, $\lambda$ **barely affects nodes expanded**: across all four values, DAVI+BWAS expands within a few nodes of $\sim$90 at depth 14. This is consistent with the main-text claim that the heuristic's ordering of children dominates BWAS behaviour at this scale. Once the heuristic is good enough to put the right child first, the $g$-cost reweighting affects which equally-good states get expanded, not how many. Optimality, on the other hand, does shift: DAVI+BWAS achieves 0-excess at every depth only at $\lambda = 1$; at $\lambda \leq 0.6$ it incurs $+0.07$ mean excess at depth 14. Supervised+BWAS achieves 0-excess at every $\lambda$, consistent with its lower bias keeping the heuristic close to admissible pointwise (not just on average).

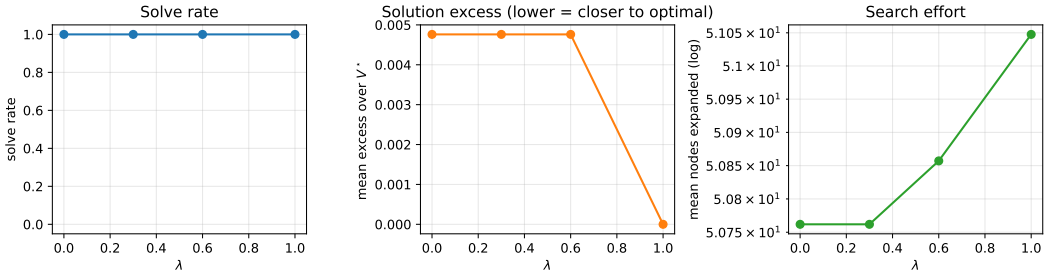


Figure 7: $\lambda$-ablation for DAVI+BWAS. Left: solve rate. Middle: mean excess over $V^\star$. Right: mean nodes expanded (log scale). Each point averages over the 420-state test set.

## D Example solve

Figure 8: An example DAVI+BWAS solve of a randomly scrambled state. Each panel is the cube state after applying one move from the recovered solution. The first panel is the scrambled start; the last is the solved cube. The accompanying supplementary video (`paper/figures/solve_animation.mp4`) shows the same process in 3D.

## E Naive supervised baseline

To check whether the training-state *distribution* affects the heuristic on its own, I trained a second supervised network that samples training states uniformly over all $3{,}674{,}160$ states (as opposed to the reverse-scramble distribution used by both DAVI and the main supervised baseline). With $20{,}000$ iterations at the same batch size, this naive variant reaches MAE $= 0.64$ on a uniform random sample, $\sim 13\%$ better than the reverse-scramble supervised baseline and $\sim 23\%$ better than DAVI's best checkpoint ($0.83$ MAE) on the same metric.

However, the aggregate MAE hides poor calibration on shallow states. Probing on hand-crafted inputs:

- $V_\theta(\text{solved}) = 2.33$ (true value: $0$).
- $V_\theta$ on the six depth-1 states ranges over $[1.48, 3.38]$ (true value: $1$).

The cause: a uniform sample over $3{,}674{,}160$ states almost never includes a shallow state ($V^\star \leq 4$ accounts for under $0.02\%$ of the state space), so the network never learns to predict small values. Reverse-scrambling places one-fifteenth of training mass on each scramble depth in $[1, 15]$, including the rare deep-shallow extremes. This illustrates that for value-iteration function approximation, the training-state distribution matters as much as the target signal. The *coverage* of the loss over depth is what matters, not the average loss.