EFFICIENT CHARGING STRATEGIES FOR ELECTRIC TRUCKS USING ACTOR-CRITIC REINFORCEMENT LEARNING WITH TREE ENSEMBLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-haul electric trucking requires charging strategies that balance delivery efficiency against battery health, under stochastic energy consumption. We formulate this as a profit-maximizing sequential decision problem: At each charging station, an agent chooses how long to charge for and with what power, trading delivery revenue against driver wages and battery degradation. The environment is inherently non-stationary since the agent's own decisions determine how battery capacity evolves over time. Addressing this problem well calls for methods that remain effective under drifting dynamics and can run on constrained, embedded, in-vehicle compute. To address this challenge, we propose tree ensembles combined with reinforcement learning. Tree ensembles excel on tabular low-dimensional data typical of such control problems, but remain largely unexplored for actor-critic reinforcement learning. The only existing method, which grows ensembles without bound during training, incurs continuously increasing memory and compute costs. As an alternative, in this paper we introduce Tree Ensemble Reinforcement Learning (TERL), which instead fixes the ensemble size. To update the model, TERL periodically rebuilds trees with off-policy reinforcement learning, and refits leaf values with on-policy methods between rebuilds. This requires no incremental tree-growing interface, while still letting the ensemble continuously learn and adapt to non-stationary dynamics. We instantiate TERL with XGBoost and demonstrate its competitiveness against existing methods. We evaluate both on standard benchmark and our electric truck charging environments, offering a realistic testbed for EV charging optimization and a general, resource-efficient framework for tree ensemble reinforcement learning.

1 INTRODUCTION

Electric vehicles are expected to play a vital role in the transition away from fossil fuels, towards a more sustainable transportation system. Although the world has seen a significant increase in battery electric passenger cars and smaller distribution vehicles, long-haul road transport still depends on diesel as the primary source of energy. Among other challenges such as large upfront costs and the heavy weight of the batteries, electric trucks suffer from short range and long charging times compared with refueling conventional trucks, and the charging infrastructure available to them is far behind that of both their diesel-fueled counterparts and smaller electric vehicles.

A crucial aspect of operating heavy-duty trucks efficiently for long-haul is therefore to have an optimal charging strategy. Such a strategy must account for the energy consumption of the vehicle and for the restrictions on the driver and mission, such as planned breaks and rest times. Energy consumption is hard to predict accurately, as it depends critically on driver behavior, terrain, weather conditions and traffic patterns, so regardless of the level of modeling detail the observed consumption will always appear stochastic. The problem also contains a non-trivial trade-off between charging efficiency and battery health. High charging power fills the battery faster, shortening the time required to reach the state-of-charge (SoC) needed for the next part of the trip, but it imposes greater stress on the materials inside, which may cause the battery to lose capacity faster.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Recently, vehicle chargers with a maximum power output in the megawatt range have started appearing along our roads. To protect their batteries, vehicles that accept megawatt charging rely on sophisticated battery management systems (BMS), which track internal temperature and resistance to avoid severe damage. Despite such systems, larger power is still associated with higher risks and costs, and unless strictly needed for a quick departure, charging with lower power may be the wiser option in the long term. In a recent study of real-world battery health data from more than 22,700 electric vehicles across 21 models, the single most influential factor in battery degradation was found to be the charging power rate (Argue & Geotab Inc., 2026), with the rate of degradation roughly doubled for vehicles that frequently use high power charging. That said, fast charging is also identified there as a key enabler for operational efficiency, which further highlights the need for clever charging strategies. In short, the ability to charge at high powers and to use a large window of the battery's SoC gives better efficiency and more flexibility, making electric trucks competitive for more missions. However, for such vehicles to stay profitable over their lifetime, one must use and charge them wisely.

Reinforcement learning (RL) has been applied to electric trucks with degradation in mind, but for other decision variables. Jia et al. (2025) optimize velocity in car-following with a semi-empirical calendar and cyclic aging model. Closer to our setting, charging strategies for long-haul electric trucks have been studied outside reinforcement learning. Zähringer et al. (2024) plan charging duration and rest allocation along a corridor with dynamic programming, including EU driving-time rules and realistic nonlinear charging curves, and re-plan at every charging point as predictions drift. That approach optimizes against a forecast model of consumption and charger availability, and it does not model battery degradation, so the trade-off between charging speed and battery health plays no part in its decisions. To the best of our knowledge, no existing work learns a charging strategy that trades delivery efficiency against long-term battery health, under stochastic energy consumption and drifting dynamics, while respecting the driving-time regulations a real mission is subject to.

To fill this research gap, we propose a reinforcement learning framework in which the agent decides where to charge, for how long, and at what maximum power. The reward is based on realistic objectives, where the positive term comes from the value of moving the goods and the negative ones from driver salary, electricity cost, battery degradation and failure to complete the mission. To solve this problem we also propose a novel way of using tree ensemble methods for reinforcement learning, an under-explored approach compared with deep reinforcement learning. Our contributions are twofold, as we provide:

i. A charging-strategy environment for electric heavy-duty trucks: We formulate long-haul EV charging as a profit-maximizing Markov Decision Process (MDP) in which an agent decides, at each charging station, how long to charge and at what power. The environment combines a physically motivated battery degradation model, separating charging-induced degradation from driving-induced degradation, with a stochastic energy-consumption model calibrated on real-world electric-truck field data. Further, it explicitly incorporates EU driving-time regulations (mandatory breaks and daily rest). This provides a realistic, reproducible testbed for evaluating RL-based charging strategies under the joint uncertainty and constraints that make the problem hard in practice.

ii. Tree Ensemble Reinforcement Learning (TERL), an actor-critic framework built on fixed-size tree ensembles: Unlike Gradient Boosting RL (GBRL) (Fuhrer et al., 2025), which grows ensembles without bound during training, TERL fixes the number of trees and alternates between an off-policy macro stage, periodically rebuilding the ensemble, and an on-policy micro stage that refits only leaf values between rebuilds. This keeps memory and per-step compute low over arbitrarily long training runs, requires no incremental tree-growing interface, and remains compatible with standard gradient boosting libraries. This makes it a practical fit for resource-constrained, embedded deployment settings such as those available in modern vehicles. A more detailed discussion on the works related to our novel RL framework can be found in Appendix B.

2 Problem Formulation

In this work we consider the real-world problem of planning strategic charging stops for electric trucks, modeled as a sequential decision-making problem with the aim of maximizing the long-term profit. The environment simulates the progression of an electric heavy-duty truck along a predetermined route, where the agent must balance the immediate demands of route completion

against the long-term economic impacts of its actions. At each viable decision point, which come at the charging stations the agent passes, the action space consists of two continuous choices. The stop duration $T_{\text{charging}} \in [0, 11]$ h is the time spent at the station. If it is 0 the agent continues driving without stopping, and the upper bound follows the minimum uninterrupted daily rest required of drivers in the EU. The target power $C_{\text{target}} \in [0, 2]$ is the requested charging power expressed as a capacity rate, or C-rate, which relates the charging rate to the battery's maximum capacity. The upper bound of 2C corresponds to fully charging the battery in 30 minutes.

The reward is formulated as a profit, which is the difference between the value of the delivery made and the costs of electricity, driver wages, and the quantified battery degradation.

By optimizing its policy in this environment, the agent inherently learns to maximize operational efficiency while preserving the long-term health of the vehicle's most expensive component. As the battery capacity decreases, the agent becomes more constrained with limited driving range and so battery degradation induces costs both directly as a value asset and indirectly through reduced operational efficiency.

2.1 Battery degradation and energy consumption

A critical component of the profit-based environment is the battery degradation model, which bridges the gap between short-term charging decisions and long-term asset depreciation. Battery aging is a highly complex process, but for the purposes of this operational framework we model it through two mechanisms: Charging-induced degradation, which grows with the square of the requested C-rate, and driving-induced degradation, which is smallest near 50% SoC and grows quadratically away from it. Both are expressed as lost energy capacity, and a simplified BMS restricts fast charging to the 10–80% SoC window with a linear ramp outside it. The model is not aimed at capturing every detail of battery aging, it is designed to be computationally cheap enough to serve large-scale experiments while covering the overall effects of charging and discharging at the high C-rates and wide SoC windows common in today's electric vehicles. Energy consumption is drawn from a low-resolution model fit to real-world data from electric trucks operating in Germany (Corguillé et al., 2025), as a function of average speed, ambient temperature, total weight and altitude difference, with the realized consumption sampled around that mean to reflect the predictive uncertainty that remains even for high-resolution models (Nilsson et al., 2026). Both models are given in full in Appendix C.

2.2 Reward function and environment

To formalize the problem, we model the truck's journey as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. The environment tracks the vehicle's progress across a sequence of discrete nodes along the route, corresponding to available charging stations or the final destination.

State Space
At each decision step t, the state $s_t \in \mathcal{S}$ encapsulates all necessary information required for the agent to make its charging decisions. We define the state vector as:

$$s_t = (S_t, Q_t, \tau_t, \delta_t, p_t, x_{d,t}, x_{c,t}, a_{d,t}, a_{c,t}, h_t, w_t, \varphi_t, \psi_t) \tag{1}$$

where $S_t \in [0, 1]$ is the current State-of-Charge, Q_t represents the current maximum battery capacity (which decays over time as degradation accumulates). τ_t and δ_t tells the agent how long time is left until the next mandatory break or rest time, and how long the next stop must be at minimum respectively, based on EU-regulation for drivers. p_t is the local electricity price at the current charging node, $x_{d,t}$, $x_{c,t}$, $a_{d,t}$ and $a_{c,t}$ correspond to the remaining distance and altitude difference to the destination and next charger respectively. h_t refers to the current ambient temperature, w_t the total weight of the vehicle, φ_t the day of the year and ψ_t the time of the day.

As can be noted, the speed variable v present in equation 21 is missing from the state vector, as we assume a constant average speed of 80 km/h. For the temperature, h_t is sampled from historical recordings from Meteostat (Lamprecht) over time, to reflect realistic fluctuations and seasonality. As for the weight w_t, it is updated by uniform sampling between 30 and 65 tonnes each time the truck reaches one of the destination nodes.

Action Space and Transitions
The action $a_t \in \mathcal{A}$ comprises the chosen charging duration and target power: $a_t = (T_{\text{charging}}, C_{\text{target}})$, and an action is required each time the agent reaches a charging station. The transition dynamics $\mathcal{P}(s_{t+1}|s_t, a_t)$ update the environment state based on the energy consumed to reach the next station, the time elapsed, the new State-of-Charge, the battery capacity lost, and the characteristics of the next charger and destination nodes. Specifically, the updated capacity reflects the degradation models detailed above:

$$Q_{t+1} = Q_t - Q_0(D_{\text{charge}} + D_{\text{drive}}), \tag{2}$$

where Q_0 is the original non-degraded battery capacity.

Reward Function
The reward function $\mathcal{R}(s_t, a_t)$ translates the objectives into a scalar profit signal. The agent receives a reward r_t at each step corresponding to the value of the goods moved subtracted by the operating costs, after taking action a_t in state s_t, calculated as:

$$r_t = M \cdot w_l \cdot \Delta x - (c_{\text{e}} \cdot E_{\text{charge}} + c_{\text{w}} \cdot \Delta\tau + c_{\text{b}} \cdot (D_{\text{charge}} + D_{\text{drive}})) - P \cdot \mathbb{I}_{\text{fail}} \tag{3}$$

Here M is the monetary value per tonne-kilometer of moving the load, applied to the distance driven Δx and the load weight w_l, c_{e} is the unit price of electricity, applied to the energy actually charged E_{charge}, c_{w} is the hourly cost of the driver, applied to the elapsed time $\Delta\tau$, which covers both charging and driving, c_{b} converts modeled capacity loss into a depreciated asset value, and P is a heavy penalty applied if the mission fails ($\mathbb{I}_{\text{fail}} = 1$), which occurs if the State-of-Charge drops below 0.

By formulating the reward as a direct reflection of profit and operational costs, the reinforcement learning agent is continuously incentivized to find a holistic balance. Over time, it must learn to avoid unnecessarily high charging powers when not justified and strategically plan stops to avoid operating the battery at extreme SoCs, all while ensuring the operations run smoothly and reliably to secure the revenue.

3 TREE ENSEMBLE REINFORCEMENT LEARNING

Most modern RL algorithms have been developed with deep neural networks, and are often not ideal and easy to implement with tree ensembles as the function approximators. Therefore, we propose a generic framework to adapt commonly used reinforcement learning methods to be utilized with tree ensembles, while considering memory and compute efficient training. In particular, our framework utilizes both on- and off-policy reinforcement learning algorithms combined.

Our approach, which we call Tree Ensemble Reinforcement Learning (TERL) provides an alternative to the recent GBRL framework (Fuhrer et al., 2025). Instead of fitting new trees to the ensemble at every optimization step, we propose to update the leaf values of the existing trees using functional gradient ascent. Concretely, the policy gradient theorem (Appendix A) is applied in function space rather than parameter space, with the ensemble F parametrizing the policy, and the advantage A in place of the action-value,

$$\nabla_F J(\pi_F) \propto \mathbb{E}_{\pi_F}\left[\nabla_F \log \pi_F(a \mid s)\, A(s, a)\right], \tag{4}$$

which is the same quantity GBRL uses, differing only in how it is realised. GBRL projects equation 4 onto a newly grown tree and appends that tree to the ensemble. TERL instead projects it onto the partition the current ensemble already defines, so the step changes the leaf values rather than the number of trees. This way, we prevent the tree ensembles from growing ever larger, which would make the program require more memory and take longer to run. With our method the ensemble size can be predetermined, while still allowing for continuous improvements. This is particularly useful for systems with highly limited resources, such as those found in modern vehicles, autonomous robots and edge devices. Furthermore, without regularly having to build new trees and search for optimal splitting criteria, our method also allows faster training.

One issue with simply tuning the leaf output values is that the trees built early on during training may not contain the most optimal structure. This could be due to either a dynamic and ever-changing environment, or simply that the trees are built before a significant amount of data has been collected. To mitigate these potential issues, we let the tree ensembles be refitted from scratch occasionally,

using off-policy RL on the historical data collected so far. During the iterations between the re-building of the whole ensemble, the leaf values are updated using on-policy learning. Algorithm 1 summarizes this loop. Since TERL does not require continuous appending of new trees it works efficiently with existing libraries for building tree ensembles. Furthermore, it is flexible enough to allow for any combination of on- and off-policy RL algorithms.

Algorithm 1 Tree Ensemble Reinforcement Learning

$\pi_{0,0} \leftarrow$ a random initial policy.
$\mathcal{D} \leftarrow \emptyset$.
Add trajectories sampled via $\pi_{0,0}$ to $\mathcal{D}$.
for rebuild $u = 1, \ldots, U$ **do**
  Build critic $V_{u,1}$ and actor $\pi_{u,1}$ from $\mathcal{D}$.
  for refit $j = 1, \ldots, L$ **do**
    Sample N pairs (s, a) with policy $\pi_{u,j}$.
    Compute advantages $\hat{A}_1, \ldots, \hat{A}_N$.
    Refit leaves with $\{(s_i, a_i, \hat{A}_i)\}_{i=1}^N$ to get $V_{u,j+1}$, $\pi_{u,j+1}$.
    Add $\{(s_i, a_i, R_i)\}_{i=1}^N$ to $\mathcal{D}$, with R_i the observed return.
  end for
end for

In our framework, the output of each leaf in any individual tree of an ensemble is considered as a conditional mean contribution to the full ensemble output, given the feature values that project an input down to that leaf. The mean is calculated using the historical data samples that end up in the specific leaf. This way of modeling the leaf outputs as conditional mean contributions was introduced in (Nilsson et al., 2024) and allows for calculating the corresponding uncertainty based on the sample variance, which is often needed for the actor ensembles, depending on the particular RL algorithm used. The individual predictions from all trees are then added up to form the total ensemble output.

3.1 TERL WITH AWR AND A2C/PPO

To instantiate the off-policy stage of TERL, we employ Advantage-Weighted Regression (AWR) (Peng et al., 2019), and we implement A2C (Mnih et al., 2016) and PPO (Schulman et al., 2017) separately for the on-policy learning in-between tree re-builds. As of the particular tree ensemble method, we use XGBoost.

The actor is a single gradient-boosted ensemble of K trees with vector-valued leaves, where every leaf stores one value per action dimension. This way, one partition of the state space parametrizes the whole policy. The critic is a separate scalar-valued ensemble, also of K trees. Because K is fixed and trees are never appended, no incremental tree-growing interface is required, which is one of the practical difficulties and drawbacks of GBRL.

Following (Nilsson et al., 2024), each leaf is read as a conditional mean contribution to the ensemble output rather than as an estimate of the output itself. Tree i is fit to the stage residual

$$z_i^{(d)}(s,a) = a_d - \mu_{1:i-1}^{(d)}(s), \qquad \mu_{1:i-1}^{(d)}(s) = \sum_{j<i} m_j^{(d)}[\ell_j(s)]\,, \tag{5}$$

where $\ell_i(s)$ denotes the leaf that state s reaches in tree i, and $z_i^{(d)}$ is the part of action dimension d still unexplained when tree i enters the ensemble. The stored statistics $m_i^{(d)}[l]$ and $v_i^{(d)}[l]$ are the weighted sample mean and variance of $z_i^{(d)}$ over the buffer samples reaching leaf l of tree i.

Each tree's contributions are then summed to form the policy, a diagonal Gaussian

$$\pi(a \mid s) = \mathcal{N}\big(\mu(s),\ \mathrm{diag}\,\sigma^2(s)\big)\,, \qquad \mu_d(s) = \sum_{i=1}^{K} m_i^{(d)}[\ell_i(s)]\,, \qquad \sigma_d^2(s) = \nu^2 \sum_{i=1}^{K} v_i^{(d)}[\ell_i(s)]\,. \tag{6}$$

Summing the variances assumes the per-tree terms are independent, as in Eq. (9) of (Nilsson et al., 2024). As the successive trees are fit to a common sequence of residuals, this is not entirely true, and the exploration factor ν is meant to correct for the covariance induced. Empirically, we find $\nu = 1/\sqrt{K}$ a good default. For the environments with bounded actions, we sample from the unbounded Gaussian and clip the value on execution. We also regress on these clipped values when updating the tree ensembles.

Off-policy stage: Rebuilding the ensembles with AWR.
Every T_{AWR} environment steps the ensembles are discarded and refit from the replay buffer $\mathcal{D}$ of B transitions (s, a, R), where R is the observed return. Advantages are recomputed against the current critic, $\hat{A} = R - V(s)$, rather than reused from collection time, the buffer spans many past policies and stale advantages carry little signal about the current one. After standardizing $\hat{A}$, the AWR weights are

$$w(s,a) = \min\Big\{\exp\Big(\hat{A}(s,a)/\beta\Big)\,,\; w_{\max}\Big\}\,, \tag{7}$$

and the actor is refit by weighted regression of the executed actions on the states, the critic by unweighted regression of R on the states, as in AWR.

Once the trees are built, the leaf tables are populated by a single pass over $\mathcal{D}$. $m_i[\,l\,]$ is the sample mean, and $v_i[\,l\,]$ the sample variance of the regression residuals over the data reaching leaf l of tree i. This makes $\sigma^2(s)$ in equation 6 a state-dependent estimate of the spread of good actions in that region, so the policy explores where the data are inconsistent and commits where they agree.

On-policy stage: Refitting leaf values with A2C or PPO.
Between rebuilds the tree structures are frozen and only the tables m_i, v_i are updated from freshly collected on-policy rollouts. Because the partition is fixed, the leaf assignments $\ell_i(s)$ are computed once per batch and reused across epochs, which makes the inner loop cheap relative to a rebuild.

For a diagonal Gaussian the policy-gradient contributions are

$$\frac{\partial \log \pi}{\partial \mu_d} = \frac{a_d - \mu_d}{\sigma_d^2}, \qquad \frac{\partial \log \pi}{\partial \sigma_d^2} = \frac{1}{2}\left(\frac{(a_d - \mu_d)^2}{\sigma_d^4} - \frac{1}{\sigma_d^2}\right), \tag{8}$$

weighted by an effective advantage $\tilde{A}$ that distinguishes the two variants. Algorithm 2 states the full procedure. A2C uses $\tilde{A} = \hat{A}$ directly. PPO uses the gradient of the clipped surrogate, $\tilde{A} = \rho\,\hat{A}$ where $\rho = \pi_{\mathrm{new}}/\pi_{\mathrm{old}}$, zeroed on samples where the clip is active.

Algorithm 2 Tree Ensemble Reinforcement Learning (AWR + A2C/PPO)

$\mathcal{D} \leftarrow$ a FIFO buffer of capacity B.
Add transitions from a random policy to $\mathcal{D}$.
for rebuild $u = 1, \ldots, U$ **do**
  Build critic V (unweighted regression of R on s).
  $\hat{A} \leftarrow R - V(s)$ on $\mathcal{D}$, standardized.
  $w \leftarrow \min\{\exp(\hat{A}/\beta),\, w_{\max}\}$.
  Build actor π (w-weighted regression of a on s).
  Set m_i, v_i by equation 5.
  for refit $j = 1, \ldots, \lceil T_{\mathrm{AWR}}/N \rceil$ **do**
    Collect N transitions with π ($\hat{A}$ by GAE).
    for epoch $e = 1, \ldots, E$ **do**
      $\tilde{A} \leftarrow \hat{A}$ (A2C) or clipped $\rho\hat{A}$ (PPO).
      Update π, V leaves by equation 9.
    end for
    Add the N transitions to $\mathcal{D}$ (drop the oldest if $|\mathcal{D}| > B$).
  end for
end for

Per-sample gradients are aggregated to the leaves they reach, giving a raw per-leaf gradient $g_i[l]$ for every tree and leaf. This aggregation is the least-squares projection of the functional gradient equation 4 onto the indicator functions of the fixed partition, which is what replaces GBRL's projection onto a newly grown tree. Two corrections are then applied. First, the magnitude of $g_i[l]$ depends on the advantage scale, the action scale and the current σ^2, so a fixed step size is unusable across tasks, we normalise each leaf's step by its own second-moment estimate, as Adam does per parameter, and express the learning rate as a fraction of the action range. Second, since μ_d, σ_d^2 and V are computed by summing over K trees, applying the full step to every tree would move the ensemble output by K times the intended amount. We distribute each of these leaf-update steps with weights ω_i that sum to one and are inversely proportional to the mean absolute leaf value of the corresponding tree. Early boosting rounds carry most of the signal and have large leaf values, while later rounds fit small residuals, so normalising by magnitude gives every tree a step of comparable relative size. Consider for instance the update of the leaf l mean value $m_i^{(d)}$ of action dimension d and tree i:

$$m_i^{(d)}[l] \leftarrow m_i^{(d)}[l] + \eta\,(a_d^{\max} - a_d^{\min})\,\omega_i\,\widehat{g}_i^{(d)}[l], \qquad \sum_{i=1}^{K} \omega_i = 1, \tag{9}$$

where $\widehat{g}$ denotes the normalized gradient. The variance tables receive the analogous update from the second expression in equation 8. The critic's leaf tables are refit in the same way, with their own ω_i, from the value residual $R - V(s)$, so that V tracks the policy between rebuilds.

4 Experimental Results

We apply the framework using XGBoost with AWR for the off-policy ensemble building, and PPO/A2C for the on-policy leaf updates (TERL-PPO and TERL-A2C) respectively. The experiments ask three questions: (**Q1**) does bounding the ensemble at K trees cost performance compared with an ensemble that grows without bound, (**Q2**) does TERL learn good charging strategies when the environment changes during training, and (**Q3**) what does each method cost in training time, in memory, and in the time it takes to produce one action?

Baselines.
We compare against GBRL (Fuhrer et al., 2025) (A2C, PPO and AWR, using the authors' GBRL package), whose ensembles grow by one tree per update and never shrink, and against neural-network A2C and PPO from Stable-Baselines3 (Raffin et al., 2021) and AWR on the same code base. TERL offers a different approach to using gradient boosted tree ensembles with actor-critic RL methods than GBRL, so the comparison provides insights into the effects of the bounded rebuilt ensembles. The neural networks are the standard reference for deep RL.

Benchmark: Pendulum (Q1).
In the Pendulum-v1 environment (Towers et al., 2025) the agent swings a pendulum upright with a torque $u \in [-2, 2]$ and keeps it there. The reward is dense, $-(\theta^2 + 0.1\dot{\theta}^2 + 0.001u^2)$. The difficulty lies in precision rather than exploration. A good policy is a smooth function of the state that must be accurate near the upright position. Trees approximate this function piecewise constantly, so the task tests whether a capped ensemble can represent a precise controller as well as a growing one. Pendulum is part of the GBRL evaluation, so the baselines use the published per-task hyperparameters. TERL's were chosen by a hyperparameter search. Every agent is trained for 10^6 steps on 5 seeds.

Application: Charging strategy for electric trucks (Q2).
A truck with a 600 kWh battery and a total weight of 30–65 t drives the roughly 1600 km corridor from Gällivare to Gothenburg, or vice versa, with 13 available chargers along the route. The EU driving- and rest-time rules are enforced by the environment. At each station the agent chooses a charging duration $T_{\text{charging}} \in [0, 11]$ h and a C-rate target $C_{\text{target}} \in [0, 2]$. The reward is the mission profit of equation 3, and the observations come from equation 1. Electricity prices are synthetic, where each station draws a base price from 0.15 €/kWh $\pm$30 % uniform jitter, fixed for the run, modulated by a diurnal factor $1 + 0.25\cos(2\pi(\psi - 18)/24)$ that peaks at 18:00 and troughs at 06:00. Prices therefore span 0.079–0.244 €/kWh across stations and hours. Temperatures are fetched from historic hourly Meteostat data. This problem challenges the agents with inputs on mixed scales, threshold effects and non-stationarity. The battery persists across missions and ages according to Appendix C.1. Below 70% state of health it is replaced, so each run combines slow drift with abrupt change. The temperature follows a calendar that runs through training, which shifts energy consumption with the seasons. A growing ensemble keeps trees fitted to conditions that no longer hold. TERL rebuilds its ensemble from recent data only. Since the charging problem is first introduced in this work, no published hyperparameters exist for it, and so all eight agents are tuned from scratch (See Appendix D for details). Final runs use 10^6 steps and 10 seeds, disjoint from the tuning seeds. All agents are evaluated deterministically on the same 120 missions: 10 per calendar month, each with a fixed seed and a starting state of health of 100, 85 or 72%. This removes seasonal and battery sampling noise from the comparison. We report normalized profit per mission and the corresponding learning curves, as means over seeds with a spread of one standard deviation across seeds, as in the benchmark figure.

For Q3, we report the time to produce one deterministic action (latency), the size of the policy that a vehicle would have to carry (model size), and wall-clock training time on the utilized hardware (steps per second). Latency is measured on an unloaded compute node with every agent pinned to a single thread, one observation at a time, which is how a vehicle would call the policy, and we report the fastest of five passes over 500 observations.

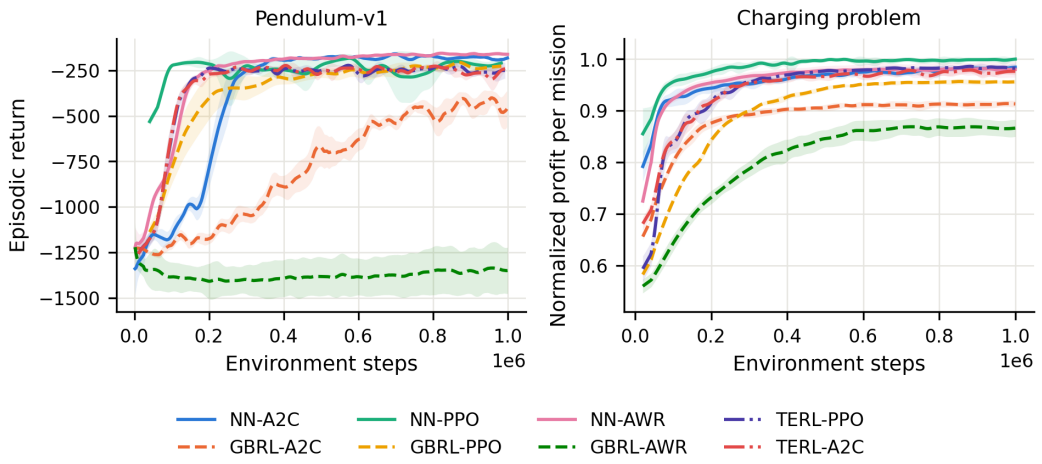

Figure 1: Learning curves, mean ± 1 standard deviation over seeds, with line style marking the family: Solid for neural networks, dashed for GBRL, dash-dot for TERL. **Left:** training return on Pendulum-v1 over 5 seeds, smoothed over 25k steps. **Right:** year-round evaluation on the charging problem over 10 seeds, with profit normalized by the best agent's final profit. The normalization is a single scale factor shared by every curve, so the ordering and the shape of the curves are unchanged.

Table 1: Results on the charging problem. Profit is normalized by the best agent's mean profit per mission, mean over 10 seeds ± 1 standard deviation across seeds, on the same 120 evaluation missions. Latency is one deterministic action on an unloaded CPU, single-threaded, best of five passes over 500 observations. The deployed policy counts the observation-to-action parameters only. Training throughput is reported with the device used, and is comparable only within a device.

Agent	Norm. profit	Latency (ms)	Model size (KiB)	Steps / s	Training device
NN-A2C	0.986 ± 0.004	0.151	**20.3**	3833	H100 GPU
GBRL-A2C	0.916 ± 0.008	1.230	1581.3	3298	H100 GPU
NN-PPO	$\mathbf{1.000 \pm 0.005}$	0.153	**20.3**	2662	H100 GPU
GBRL-PPO	0.957 ± 0.005	0.237	223.2	4084	H100 GPU
NN-AWR	0.981 ± 0.011	0.176	273.0	705	H100 GPU
GBRL-AWR	0.869 ± 0.016	2.687	8560.4	838	H100 GPU
TERL-PPO	0.981 ± 0.009	0.045	71.3	3562	CPU
TERL-A2C	0.977 ± 0.006	**0.041**	34.4	**4454**	CPU

4.1 Results

Pendulum (Q1).
Figure 1 (left) shows that bounding the ensemble does not cost performance relative to GBRL. Averaged over the last 10^5 steps, TERL-PPO ends at -258 ± 41 and TERL-A2C at -266 ± 34, against GBRL-PPO's -232 ± 22; the three are within one standard deviation of each other, so capping the ensemble at K trees costs nothing measurable on this task. The other two GBRL agents are clearly behind. GBRL-A2C is still improving at 10^6 steps, at -439 ± 63, and GBRL-AWR does not learn the task with its published hyperparameters, ending at -1347 ± 183. It is worth stating that our experiments here agree with the results reported in the GBRL paper for all GBRL- and NN-based agents. TERL-PPO and TERL-A2C are almost indistinguishable, which suggests that the overall methodology and AWR rebuilds, not the specific choice of leaf update, drive performance on this task. The neural-network agents remain the reference on this smooth, low-dimensional control problem. NN-AWR ends at -160 ± 3, NN-A2C at -183 ± 8 and NN-PPO, at -216 ± 31. The remaining gap is consistent with the piecewise-constant approximation discussed above.

Charging electric trucks (Q2).
Table 1 reports final performance and Figure 1 (right) the learning curves. Profit is reported normalized by the best agent's mean profit per mission, so the comparison is one of relative profitability

rather than of absolute revenue, which depends on the freight rates and electricity prices assumed by the environment. TERL is the best of the tree-ensemble methods by a clear margin. TERL-PPO reaches 0.981 ± 0.009 of the best agent's profit against GBRL-PPO's 0.957 ± 0.005. Hence, it is 2.5% more profitable per mission than GBRL-PPO, which is also the strongest GBRL variant, so this holds against all of the baseline tree ensemble agents. Against the neural agents TERL is competitive, as TERL-PPO is level with NN-AWR (0.981 ± 0.011) and indistinguishable from NN-A2C (0.986 ± 0.004). It trails, however, the strongest agent NN-PPO by 1.9%, placing it between the best neural and GBRL models for this task. As on Pendulum, the two TERL variants land within half a percent of each other, which again points to the rebuild schedule rather than the choice of leaf update as what matters.

Cost (Q3).
Table 1 also report what each method costs in terms of memory and time. The deployed policy counts the numbers required to map an observation to a deterministic action, at four bytes each. For the networks, feeding every parameter from the observation to the action mean. For the tree methods, every internal node, and the outputs stored at each leaf.

On the speed measure, TERL is the fastest agent at inference, at 0.041–0.045 ms per action against 0.151–0.176 ms for the neural policies and 0.237–2.687 ms for GBRL. On model size, TERL is in the same range as the neural policies, and far below the growing ensembles. TERL-A2C carries 34.4 KiB, 1.7 times the smallest neural policy and 7.9 times smaller than NN-AWR, while GBRL spans 223 KiB to 8.4 MiB. Against GBRL specifically, TERL-A2C is 6.5 times smaller than GBRL-PPO, 46 times smaller than GBRL-A2C, and 249 times smaller than GBRL-AWR. The two TERL variants differ by a factor of two in size, 34.4 against 71.3 KiB. This comes at a difference in profit of less than half a percent, so the smaller variant may well the better deployment choice.

What the fixed ensemble buys is not only a lower cost, but a stable one. TERL's inference cost is set by K and the tree depth and does not change over training, whereas a growing ensemble becomes slower with every update. Training time is reported with the device each agent used. The TERL agents were trained on the CPU and everything else on an H100 GPU, so those columns are not directly comparable. We note only that TERL reaches 3562 and 4454 environment steps per second on CPU, which is on par, or faster, than the baselines training on the GPU.

5 Conclusion

State-of-the-art reinforcement learning methods have been developed for neural networks specifically, and are not optimal for use with tree ensembles as the function approximators. At the same time, tree ensembles have shown great success in many applications of machine learning, and can offer benefits over neural networks. Furthermore, our solution draw upon the benefits of both on- and off-policy methods, allowing for active interaction with the environment for efficient exploration, while staying sample efficient. Our results show that tree ensemble reinforcement learning performs similar to the neural-network agents in profit and in deployed memory, while producing actions faster. Moreover, it improves on GBRL with more profit per mission, a smaller memory footprint, and less time required for inference, against all of the GBRL variants.

Several directions remain open. We have considered continuous actions only, but TERL carries over to discrete action spaces with little change. Hybrid action spaces would fit the charging problem naturally, with a discrete choice of whether to stop and which charger power level to use alongside a continuous duration. More broadly, TERL, like GBRL, adapts algorithms that were designed for neural networks. Algorithms designed for tree ensembles from the outset could exploit what the partition already provides, using per-leaf counts and variances for uncertainty-driven exploration or per-leaf trust regions, and could trigger rebuilds when incoming data drift away from the current partition rather than on a fixed schedule.

AI use statement

In this work, we used generative AI tools, specifically large language models, for several tasks that fall under the required disclosure categories. We used them to implement methods: The tree-ensemble agents, the charging environment, the hyperparameter-search harness, the evaluation protocol and the plotting code were written and revised with AI assistance, on top of the existing

stable-baselines3, GBRL and XGBoost libraries, following designs and specifications set by us. We used them to obtain feedback on research methodology and experimental design, in particular on enforcing a single tuning protocol across all agents so that no agent receives a favorable treatment, on the configuration of the search's early-stopping criterion and on how inference cost should be measured to be comparable across agent families. We used them to help interpret results and to refine diagnostic hypotheses about behavior we observed during training, such as late-training performance decay and measurement artifacts in our own timing instrumentation. Each such hypothesis was then tested by a controlled experiment of ours rather than accepted on the strength of the explanation. Finally, AI assistance was used in checking that our description of the proposed method agrees with what the released code actually computes. This process identified and corrected several discrepancies between text and implementation, and a human verified all necessary code updates found in the process.

We have not used generative AI tools to formulate the mathematical claims of this work. The method proposed here, its objective, and the conceptual framing of the problem as a profit-maximizing Markov decision process were conceived by the authors. Of the remaining required disclosure categories, translation, qualitative and thematic data analysis, mathematical proof-writing and the generation of synthetic data sets are not applicable to this work. The environment's stochastic components were specified by us, with AI assistance confined to implementing them in code. Data cleaning and reformatting was limited to the processing of publicly available historical weather records, for which AI-assisted code was used.

Additionally, among the tasks for which disclosure is recommended, we used generative AI tools for editing and rewriting prose, for producing figures and tables from our own result files, for debugging, and for summarizing related work as an aid to our literature search.

We have reviewed all AI-assisted work. All code was read by the authors and verified by execution. The agents were checked against reference behavior on standard continuous-control benchmarks, and changes to the inference path were accepted only after confirming that they reproduce the previous outputs exactly on the trained models we report. Every AI-assisted summary of prior work was checked against the original publication, and at least one inaccurate characterization of a cited study was found and corrected in this way. No experimental result, numerical value or table entry reported in this paper was produced by a generative AI tool. All reported numbers come from our own executed runs and their logs, and we re-derived the tables and figures from those logs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. pp. 2623–2631, 07 2019. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330701.

Charlotte Argue and Geotab Inc. Ev battery health study: New data on fast charging & degradation. Technical report, Geotab Inc., January 2026. URL https://www.geotab.com/press-release/ev-battery-health-degradation-fast-charging-study/. Updated analysis of 22,700 electric vehicles across 21 models.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2022. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3229161.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, 2016. doi: 10.1145/2939672.2939785.

Juliette Le Corguillé, Florian Hacker, Katharina Göckeler, Moritz Mottschall, and Theresa Dolinga. Real-world data analysis of energy consumption, activity and charging patterns of battery electric trucks operating in germany. Technical report, 2025. URL https://www.oeko.de/fileadmin/oekodoc/Real-world-data-analysis-of-battery-electric-trucks.pdf.

Benjamin Fuhrer, Chen Tessler, and Gal Dalal. Gradient boosting reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=URmGywyt59.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, pp. 18932–18943. Curran Associates, Inc., 2021.

Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, pp. 507–520. Curran Associates, Inc., 2022.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Qun8fv4qSby.

Ruo Jia, Kun Gao, Shaohua Cui, Jing Chen, and Jelena Andric. A multi-objective reinforcement learning-based velocity optimization approach for electric trucks considering battery degradation mitigation. *Transportation Research Part E: Logistics and Transportation Review*, 194:103885, 2025. ISSN 1366-5545. doi: https://doi.org/10.1016/j.tre.2024.103885. URL https://www.sciencedirect.com/science/article/pii/S1366554524004769.

Vivek N. Lam, Xiaofan Cui, Florian Stroebl, Maitri Uppaluri, Simona Onori, and William C. Chueh. A decade of insights: Delving into calendar aging trends and implications. *Joule*, 9(1):101796, 2025. ISSN 2542-4351. doi: https://doi.org/10.1016/j.joule.2024.11.013. URL https://www.sciencedirect.com/science/article/pii/S2542435124005105.

Christian Sebastian Lamprecht. Meteostat python. URL https://github.com/meteostat/meteostat-python. Retrieved November 5, 2025.

Lorenzo Magnino, Kai Shao, Zida Wu, Jiacheng Shen, and Mathieu Laurière. Solving continuous mean field games: Deep reinforcement learning for non-stationary dynamics, 2025. URL https://arxiv.org/abs/2510.22158.

Volodymyr Mnih, Adrià Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 02 2016. doi: 10.48550/arXiv.1602.01783.

Hannes Nilsson, Rikard Johansson, Niklas Åkerblom, and Morteza Haghir Chehreghani. Tree ensembles for contextual bandits. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=59DCkSGw8S.

Hannes Nilsson, Rafael Basso, Balázs Kulcsár, and Morteza Haghir Chehreghani. Probabilistic physics-aware machine learning predictions of electric truck energy consumption with field data, 2026. URL https://arxiv.org/abs/2607.19054.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6638–6648. Curran Associates, Inc., 2018. URL https://neurips.cc.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063. MIT Press, 2000.

Mark Towers, Ariel Kwiatkowski, John Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, J Terry, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-4916. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/d7ff1795e8527f6443371c3933bdb52b-Paper-Datasets_and_Benchmarks_Track.pdf.

Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst lifelong non-stationarity, 2020. URL https://arxiv.org/abs/2006.10701.

Maximilian Zähringer, Olaf Teichert, Georg Balke, Jakob Schneider, and Markus Lienkamp. Optimizing the journey: Dynamic charging strategies for battery electric trucks in long-haul transport. *Energies*, 17(4):973, 2024. doi: 10.3390/en17040973.

A Reinforcement learning

A reinforcement learning problem is formalized as an iterative loop, where at each timestep t an agent observes the current state s_t and takes an action a_t drawn from its policy π, a probability distribution over the actions available in that state. Then, the environment the agent operates in returns a reward r_{t+1} and a new state s_{t+1}, drawn from an unknown distribution $p(s', r|s, a)$, which completes the current loop. The goal for the agent is then to maximize the discounted future rewards, also known as the return G_t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{j=0}^{\infty} \gamma^j R_{t+j+1}, \tag{10}$$

where γ is the discount factor that controls the short-sightedness of the agent. In an episodic task, such as the one studied here, the sum terminates at the end of the episode, which is equivalent to treating the terminal state as absorbing and reward-free. The way that the agent can control the future rewards it receives is to update the policy.

In order to guide the agent during the optimization process, it is common for reinforcement learning methods to use value functions. State-value function $V^\pi(s)$ is formalized as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[\sum_{j=0}^{\infty} \gamma^j R_{t+j+1} \middle| S_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^\pi(s') \right], \end{aligned} \tag{11}$$

and the action-value function $Q^\pi(s,a)$ like

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi \left[\sum_{j=0}^{\infty} \gamma^j R_{t+j+1} \middle| S_t = s, A_t = a \right] \\ &= \sum_{s',r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right]. \end{aligned} \tag{12}$$

Lastly, the advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ is a useful concept commonly utilized by modern reinforcement learning methods. It represents the expected additional value gained from taking action a in state s, relative to an action drawn from π, and is therefore positive exactly for the actions that improve on the current policy.

Although often useful in practice, value functions are not strictly needed to maximize the return. For an episodic task the objective is the expected discounted return from the initial state distribution d_0,

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t R_{t+1} \right] = \mathbb{E}_{s_0 \sim d_0} \left[V^{\pi_\theta}(s_0) \right], \tag{13}$$

and the policy gradient theorem (Sutton et al., 2000) states that

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a_t|s_t)\, Q^{\pi_\theta}(s_t,a_t) \right], \tag{14}$$

where the expectation is over states drawn from the on-policy state distribution and actions drawn from π_θ. Since the observed return G_t is an unbiased estimate of $Q^{\pi_\theta}(s_t,a_t)$, it may be substituted for the action-value,

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a_t|s_t)\, G_t \right], \tag{15}$$

so the agent can improve on its current policy using gradient ascent from the observed rewards directly, without a learned value function. This estimator is unbiased but has high variance, and the variance grows with the length of the episode, since G_t accumulates the noise of every subsequent transition. In order to reduce that variance, and the latency with which a late reward reaches an early action, modern techniques subtract a state-dependent baseline. Taking V^π as that baseline replaces the return with the advantage,

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a_t|s_t)\, A^{\pi_\theta}(s_t,a_t) \right], \tag{16}$$

and estimating A^{π_θ} by bootstrapping from a learned value function gives the actor-critic methods used in this work. The price is bias, as the gradient is now only as accurate as the value estimate.

Reinforcement learning methods can be broadly divided into on-policy and off-policy methods. The former class of techniques requires that the agent learns from data collected only using the current policy. Off-policy methods, on the other hand, can make use of old data as well. On-policy methods are known to be relatively simple and stable, but highly sample inefficient. Off-policy methods can benefit from re-using old data and often do not require as many interactions with the environment to learn efficiently. However, this often comes at the expense of increasing complexity and unstable behavior.

B Related work

Most previous reinforcement learning applications depend on Deep RL. In this type of reinforcement learning, (deep) neural networks are used as function approximators. Either to estimate the state- and/or action-value functions (value-based RL), or to parameterize the policy directly (policy-based RL). This type of machine learning models have shown tremendous performance on unstructured data, common in fields like computer vision and natural language processing. However, for tabular data, defined as information that can be effectively formalized in the form of a table, tree ensemble methods often outperform neural networks (Borisov et al., 2022), (Gorishniy et al., 2021), (Grinsztajn et al., 2022). Despite the performance of tree ensembles on supervised machine learning tasks on tabular data, and the fact that many reinforcement learning problems tend to involve such data, the application of tree ensemble methods to reinforcement learning is greatly underexplored, especially for policy-based and actor-critic reinforcement learning.

The recently introduced Gradient Boosting Reinforcement Learning (GBRL) (Fuhrer et al., 2025) provides a framework for applying any actor-critic reinforcement learning algorithm using gradient boosted tree ensembles as the function approximators. The authors apply it to both on-policy methods like A2C (Mnih et al., 2016) and PPO (Schulman et al., 2017), and the off-policy algorithm AWR (Peng et al., 2019)). At each training iteration, GBRL uses the policy gradient theorem and employs functional gradient descent to build and add new trees to the ensemble to correct and refine the agent's actions. By utilizing gradient boosting decision trees as the function approximators of policy-based and actor-critic reinforcement learning methods, the GBRL framework is able to provide the same benefits that tree ensembles have over neural networks for many supervised machine learning tasks for RL problems. In particular, GBRL has demonstrated a significant advantage on RL tasks involving structured data, where it tends to be more robust under out-of-distribution scenarios, spurious correlations and state-space perturbations.

Despite the impressive performance and promising features of GBRL, the specific way in which it utilizes functional gradient descent leaves the technique with a handful of issues. In particular, due to the continuous generation of new trees, the ensemble grows indefinitely, which limits the application of the method to real-time online adaptation. Additionally, the unbounded ensemble size may lead to computational inefficiency and memory issues. Lastly, the authors of GBRL note that the method is not well-suited for traditional libraries for building gradient boosted tree ensembles, like XGBoost (Chen & Guestrin, 2016) and CatBoost (Prokhorenkova et al., 2018), which are optimized for supervised learning and tend to scale poorly when used in this incremental learning context.

Apart from the specific issues related to GBRL, many reinforcement learning algorithms struggle with non-stationary environments, where the agents fail to unlearn previously successful behavior and keep up with the changing dynamics. For neural network based methods, a number of solutions have been proposed to handle such situations (Magnino et al., 2025; Igl et al., 2021; Xie et al., 2020). When it comes to GBRL, it is likely that trees in the ensembles become misaligned in a changing environment. Although building new trees can correct for this effect, it likely leads to redundancy and unnecessarily complex models. Considering the charging strategy problem we aim to solve, the environment is indeed non-stationary and the optimal actions depend heavily on the state of health and remaining capacity of the battery.

C Environment details

C.1 Battery degradation model

A critical component of the profit-based environment is the battery degradation model, which bridges the gap between short-term charging decisions and long-term asset depreciation. Battery aging is a highly complex process, but for the purposes of this operational framework, we model it through two distinct mechanisms: Charging-induced degradation and driving-induced degradation. By this simplification we assume that the truck is either driving or charging, so that the effect of calendar aging, i.e. the degradation that takes place in the battery while it is at rest with no current flowing in or out of it (Lam et al., 2025) can be considered as baked into the total degradation during charging or discharging. For simplicity, we only consider battery degradation in terms of lost energy capacity, and assume that the internal resistance stays constant over time.

Charging-Induced Degradation
To ensure realistic charging behavior, the environment assumes the presence of a simplified Battery Management System (BMS). The BMS actively modulates the requested C_{target} based on the current State-of-Charge ($S \in [0, 1]$) to protect the battery cells. Fast-charging is exclusively permitted within the optimal operational window of 10% to 80% SoC. Outside of this window, the BMS enforces a linear ramp-up and ramp-down of the charging current. The actual applied C-rate, $C(S)$, is defined piecewise as

$$C(S) = \begin{cases} 10 \cdot C_{\text{target}} \cdot S, & \text{if } 0 \leq S < 0.10 \\ C_{\text{target}}, & \text{if } 0.10 \leq S \leq 0.80 \\ 5 \cdot C_{\text{target}} \cdot (1 - S), & \text{if } 0.80 < S \leq 1.0 \end{cases} \tag{17}$$

Under these modeling assumptions, despite the varying actual charge rate $C(S)$, the BMS maintains a constant rate of degradation per SoC percentage point for any given session, dictated solely by

the selected target C-rate. We assume here that charging infrastructure is never power-limited, so the requested target rate is always achieved when allowed by the BMS. In our model, the severity of the degradation incurred during a charging session, based on the excess heat generated, scales as C^2_{target}. Therefore, the degradation accumulated during a charging event, D_{charge}, over a change in SoC (ΔS), is expressed as

$$D_{\text{charge}} = \alpha \cdot C^2_{\text{target}} \cdot \Delta S \tag{18}$$

where α is a scaling coefficient representing the degradation rate.

Driving-Induced Degradation
Battery degradation is not isolated to charging events. The state of the battery while discharging during transit also heavily influences its lifespan. Operating at extremes of the SoC spectrum induces higher mechanical and chemical stress on the battery internals. To capture this, the instantaneous driving-induced degradation rate dD_{drive}/dS is modeled as a second-order polynomial dependent on the current SoC, with the minimum degradation occurring at an optimal 50% SoC ($S = 0.50$):

$$dD_{\text{drive}}/dS = \beta \cdot (S - 0.50)^2 + \gamma \tag{19}$$

where β dictates the severity of the degradation when operating at different SoC than the ideal 50%, and γ represents the unavoidable baseline degradation occurring during operation. The total degradation cost subtracted from the agent's reward is the continuous integration of dD_{drive}/dS during transit, incentivizing the agent to maintain the battery charge within a medium range whenever the active mission allows:

$$D_{\text{drive}} = -\int_{S_{high}}^{S_{\text{low}}} \left(\beta \cdot (S - 0.50)^2 + \gamma\right) dS \tag{20}$$

We again want to stress that our battery degradation model is not aimed at perfectly capturing all the intricate details involved in battery aging. Instead, the model is designed to be computationally efficient enough to serve large-scale experiments, while covering the overall effects on degradation from charging and discharging the batteries common in today's electric vehicles with high C-rate and large SoC window.

C.2 Energy consumption model

The underlying aspect that necessitates charging is of course the trucks' energy consumption. Here, we employ a simple low-resolution model based on real-world data from electric trucks operating in Germany (Corguillé et al., 2025). In this model, the energy consumption in kWh over a distance x kilometers driven, is given by a function of average speed v (km/h), ambient temperature h $(^\circ C)$, total weight w (kg) and altitude difference a (m):

$$E_{\text{drive}} = x \cdot (m_1 e^{-k_1 v} + m_2 h + m_3 w + m_4 a + m_5), \tag{21}$$

where the best fit coefficients, based on their data are reported as: $k_1 = 0.174$, $m_1 = 5.13$, $m_2 = -0.0131$, $m_3 = 1.83 \times 10^{-5}$, $m_4 = 0.001$ and $m_5 = 0.710$. In reality, the specific coefficients vary between different trucks and driving conditions, and a more detailed model that takes specific physical effects of vehicle dynamics and more detailed information such as acceleration profiles and wind into account may generate more realistic numbers. However, such detailed information is difficult to know beforehand, and more involved models suffer from heavier computational load. Furthermore, as demonstrated in (Nilsson et al., 2026), a certain level of predictive uncertainty remains even for high-resolution models with detailed inputs, which reflects the unavoidable stochasticity related to energy consumption of electric vehicles. Hence, in accordance with this previous work, we sample the realized energy consumption over a stretch of road from a normal distribution $\mathcal{N}(\mu_{E_{\text{drive}}}, \sigma_{E_{\text{drive}}})$, with $\mu_{E_{\text{drive}}} = E_{\text{drive}}$ and $\sigma_{E_{\text{drive}}} = 0.05 E_{\text{drive}}$, with E_{drive} obtained from equation 21. This relatively simple model provides a fast and sufficiently accurate model of energy consumption for the purposes of our study, taking the most influential factors of energy consumption and the inherent uncertainty into account.

D Hyperparameter tuning

Benchmark: Pendulum.
Pendulum-v1 is part of the evaluation published with GBRL, so the three GBRL agents and the three

neural-network agents use the per-task hyperparameters released by Fuhrer et al. (2025). Reusing them rather than re-tuning keeps our numbers directly comparable with the ones reported there, which is what lets us state that our benchmark results agree with theirs. The two TERL agents have no published configuration for this task, so theirs were selected separately.

Charging problem.
The charging environment is introduced in this work, so no published hyperparameters exist for any of the eight agents and all of them are tuned from scratch under one protocol. We use Optuna (Akiba et al., 2019) with the TPE sampler (Bergstra et al., 2011) and median pruning, each trial averaging 2 seeds over 10^6 environment steps. Trials are run at the same budget as the final runs rather than at a shortened one. Configurations that look good early and drift later are otherwise selected, because the search never sees the drift.

A trial is scored on the year-round evaluation described in Section 4, as the mean return over the last 4 of 10 evaluation points, minus the gap between that tail and the run's best evaluation point. The first term rewards a configuration that ends well rather than one that peaks once. The second penalizes a configuration that reaches a high point and then decays, which a tail mean alone still rewards as long as the decay is slow. Pruning uses the same quantity, so a trial is stopped only when it is behind the median of the trials before it, and no trial is stopped before it has trained through a fixed fraction of the budget — without that, the search systematically discards configurations that start slowly and improve, which are exactly the ones the drift penalty is meant to favor.

Every tree-based agent is searched over the same depth range, 3 to 5. The upper limit is set by GBRL rather than by us. Its CPU backend cannot construct a greedy depth-6 ensemble at all, so a depth-6 configuration could be trained on a GPU but never run on the hardware this work targets. Applying the same range to TERL keeps the tree-based comparison symmetric.

For TERL, the ensemble size is itself a hyperparameter, and it was searched over $K \in \{50, 100, 200, 400\}$. Ensemble size and tree depth trade off against each other, so the selected K varied between agents, and the cost we report would then have described a different model for each. We therefore report both TERL agents with K fixed at 200, so that the cost figures in Section 4.1 correspond to a stated ensemble size.

The per-agent search spaces differ, because the families share few parameters. The neural agents vary the learning rate, rollout length, entropy and value coefficients, gradient clipping, the initial policy standard deviation and the network width. PPO additionally varies the number of epochs, the clip range and the minibatch size, and AWR the actor and critic learning rates, the temperature β, the weight ceiling, the replay capacity and the update cadence. The GBRL agents vary the discount, tree depth, growth policy, the boosting learning rates for policy and value, the minimum data per leaf, and the initial and learned policy standard deviation. The TERL agents vary the discount, GAE parameter, tree depth, the boosting learning rate used to build the ensembles, the leaf-step learning rate, the AWR temperature, the rebuild interval T_{AWR}, the replay capacity B, the rollout length, the number of on-policy epochs and the entropy coefficient.

Final runs use 10^6 steps and 10 seeds, disjoint from the tuning seeds. The selected configurations are given in Tables 2–4.

Table 2: Selected hyperparameters for the Neural network agents on the charging problem.

Parameter	NN-A2C	NN-PPO	NN-AWR
actor lr	–	–	$2.59 \cdot 10^{-5}$
batch size	–	256	256
β	–	–	0.2202
buffer size	–	–	20000
clip range	–	0.3	–
critic lr	–	–	$1.93 \cdot 10^{-5}$
entropy coef.	0.00461	$1.87 \cdot 10^{-5}$	$1.08 \cdot 10^{-4}$
λ_{GAE}	0.9	0.9	0.98
γ	0.995	0.99	0.995
learning rate	$2.24 \cdot 10^{-4}$	$3.70 \cdot 10^{-4}$	$1.83 \cdot 10^{-5}$
learning starts	–	–	10000
$\log \sigma_0$	-1.56	-1.294	-1.331
max grad. norm	0.3	5	–
epochs	–	10	–
n_{steps}	8	128	–
net arch.	small	small	–
normalize advantage	true	–	true
policy gradient steps	–	–	1000
reward mode	–	–	gae
train freq.	–	–	4000
use rms prop	false	–	–
value gradient steps	–	–	500
value coef.	0.6403	0.4082	–
$w_{\max}$	–	–	20

Table 3: Selected hyperparameters for the GBRL agents on the charging problem.

Parameter	GBRL-A2C	GBRL-PPO	GBRL-AWR
batch size	–	256	1024
β	–	–	0.2449
buffer size	–	–	20000
clip range	–	0.05	–
entropy coef.	0.001	0.01	–
fixed std	false	false	true
λ_{GAE}	1	0.98	0.95
γ	0.995	0.999	0.99
gradient steps	–	–	50
grow policy	greedy	oblivious	greedy
$\log \sigma_0$	-0.5	-0.5	-2
$\log \sigma$ lr	$2.40 \cdot 10^{-4}$	$7.87 \cdot 10^{-4}$	–
max depth	4	4	3
max policy grad norm	1	1	1
max value grad norm	None	1	1
min. data in leaf	10	50	1
epochs	–	2	–
n_{steps}	8	128	–
normalize advantage	true	true	true
policy lr	0.006627	0.003184	$9.10 \cdot 10^{-4}$
reward mode	–	–	gae
shared tree struct	false	false	true
target kl	–	0.01	–
train freq.	–	–	1000
value lr	0.03955	0.01669	0.08722
value coef.	0.5	–	–
$w_{\max}$	–	–	100

Table 4: Selected hyperparameters for the TERL agents on the charging problem.

Parameter	TERL-PPO	TERL-A2C
buffer B	20000	50000
T_{AWR}	4000	10000
critic lr	0.03676	0.0142
entropy coef.	$1.21 \cdot 10^{-8}$	$1.04 \cdot 10^{-8}$
λ_{GAE}	0.98	0.9
γ	0.99	0.99
max depth	4	3
K	200	200
epochs	2	2
n_{steps}	256	128
ν	$1/\sqrt{K}$	$1/\sqrt{K}$
leaf-step lr η	0.07924	0.0187
boosting η	0.03626	0.02361