RENEWABLE-AWARE NEURAL NETWORK TRAINING WITH PHASE-DEPENDENT LYAPUNOV CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

In neural network training on GPUs, a larger batch shortens an epoch but draws more power, and its effect on the loss changes as training progresses. When renewable supply varies, the best batch size therefore depends on the available power and the training phase. Since the batch size changes how long each training segment takes, we formulate renewable-aware training as a renewal control problem that minimizes the renewable shortfall per unit time subject to a duration-weighted error constraint. We enforce the error budget with a virtual queue and design a drift-plus-penalty ratio controller that penalizes its backlog with phase-dependent Lyapunov weights. We prove an $O(1/V)$ bound on the shortfall-rate gap with an $O(V)$ average virtual backlog and extend it to finite Markov states and recurrent training classes. To run the controller in GPU training, we score each candidate action with a predictor calibrated from shared-checkpoint measurements and update the queue and the phase with the measured loss. We evaluate the controller in 75 training runs with independent supply and a measured photovoltaic (PV) day. Under independent supply, the controller reaches a common quality target in 59.7% less time than batch 16 at a similar shortfall rate, and relative to batch 128 it reduces the shortfall rate by 72.5% with a 13.4% longer target time. With a noon start on the PV day, its shortfall rate is 32.7% lower than that of batch 128 at the same target time, and as the PV capacity grows, its DWAE falls from 0.040 to 0.026 while fixed schedules keep their error. Emphasizing the backlog during rapid descent shortens the mean target time by 7.0% relative to equal weights.

1 INTRODUCTION

The energy consumed by training a neural network depends on both the power drawn and the elapsed time, and the configuration of the run determines each of them. Large batches and parallel execution shorten training (Goyal et al., 2017; Narayanan et al., 2019; Shoeybi et al., 2019; You et al., 2020) but also draw more power, and because the energy required to reach convergence depends on both (Faiz et al., 2024; Wang et al., 2019; Liu et al., 2012; Luccioni et al., 2023), a shorter run does not necessarily consume less energy. When part of the supply is renewable, its availability varies during a run, so the share of a given energy demand that the grid must cover depends on when that demand occurs. The growing data-center energy demand of AI workloads makes this timing a practical concern (Chen, 2025; O'Donnell & Crownhart, 2025).

Among the configuration choices available during a run, batch size affects power, elapsed time, and learning progress at the same time. In a pilot on an NVIDIA A800, we train a BERT model with batch size 64 for three epochs and with batch size 128 for the next seven. The mean epoch duration falls from 16.80 to 14.29 seconds, the sampled training power rises from 242.9 to 261.6 watts, and the loss decreases rapidly at first and more slowly afterward (Appendix B). A larger batch therefore reduces elapsed time at the cost of higher power, and its effect on progress depends on the stage that the run has reached, since early descent and later refinement respond differently to the same configuration (Cohen et al., 2021; Sagun et al., 2017).

A scheduler can follow a changing supply by moving computation to a different time or by changing its intensity (Iftikhar & Davy, 2024; Radovanović et al., 2023; Xu et al., 2025), and batch size provides such an intensity control for training. Two difficulties arise, however, when the workload is the training process itself. Candidate actions require different amounts of time, so comparing

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.

them per decision step does not reflect their cost per unit time. The progress that an action will produce from the current training state is also not directly observable, and scaling laws describe the resources needed to reach a final performance level rather than the effect of one more step from an intermediate state (Kaplan et al., 2020). We ask:

> *How can an online training schedule balance renewable energy use, elapsed time, and learning progress when supply and the rate of loss reduction both change?*

To answer this question, we treat training segments as renewal frames whose durations depend on the selected action, so that both the renewable shortfall and the training error are averaged over elapsed time. We cap the time-averaged error by a budget, track its violations with a virtual queue, and compare actions through a drift-plus-penalty ratio that combines this backlog with a shortfall penalty. Since training responds differently to the same action during rapid descent and near a plateau, we let the backlog weight depend on the training phase. Because progress cannot be observed in advance, a calibrated predictor scores the candidates, and the measured loss updates the queue and the phase.

We make three contributions:

- We formulate renewable-aware training as a renewal control problem whose frames have action-dependent durations, and we place renewable shortfall and training error on that common time scale (Section 2).
- We develop a phase-aware Lyapunov controller and analyze it for independent states, for finite Markov states, and for training dynamics that enter a recurrent low-loss class (Sections 3–5). We prove an $O(1/V)$ gap in the shortfall rate together with an $O(V)$ average virtual backlog, where V is the trade-off parameter. The phase weights change the backlog penalty, while the queue update and the decision rule remain the same.
- We evaluate the controller in closed-loop GPU training (Section 6), with comparisons that cover fixed and changing batch schedules, phase weights, supply timing, and supply capacity, under both a full training budget and a common quality target.

Related work. Sustainable machine learning studies the energy cost of frameworks, training configurations, and model serving (Ding & Shi, 2024; Geißler et al., 2024; Georgiou et al., 2022; Stojkovic et al., 2024), whereas work on efficient training reduces energy consumption as a consequence of reaching a target sooner (Huang et al., 2024; Wang et al., 2019; Shen et al., 2025). At the system level, GPU power management, energy-aware serving, and renewable or carbon-aware cluster scheduling shift demand toward favorable supply (Liu et al., 2012; Patel et al., 2024; Qiu et al., 2024; Radovanović et al., 2023; Stojkovic et al., 2025; Xu et al., 2025; You et al., 2023), as does work on solar-powered edge and energy-harvesting devices (Luo et al., 2025; Xu et al., 2024; Zhang et al., 2024), but these controls treat the workload as given. Adaptive batch schedules instead exploit the dependence of learning progress on batch size (Devarakonda et al., 2017; McCandlish et al., 2018; Qiao et al., 2021; Shallue et al., 2019), but optimize throughput or accuracy rather than the timing of energy use. Our scheduling state combines the two lines of work because it contains the renewable supply, the current training error, and its phase-dependent response to an action.

Our control method builds on stochastic network optimization and renewal scheduling (Neely, 2009; 2010; 2021; Li & Neely, 2013), where virtual queues and drift-plus-penalty decisions handle energy harvesting, resource allocation, and service constraints under i.i.d. or Markov environments (Chen et al., 2025; Huang & Neely, 2009; 2010; Li et al., 2025; Qiu et al., 2018; Urgaonkar & Neely, 2009). Three features of training distinguish the present setting. Frame duration is itself determined by the action, so the objective and constraint are ratios over elapsed time rather than per-slot averages. The backlog penalty depends on an observed training phase, which changes how the same violation is priced. The training component of the state does not form an irreducible chain, because a discretized loss process can leave high-loss states and remain in a closed low-loss class. The guarantee is therefore stated for the class that the trajectory reaches (Fonseca & Tweedie, 2002; Meyn & Tweedie, 1993; Puterman, 2014). Appendix A gives the extended discussion.

2 Training with a renewable-energy budget

We view training as a sequence of variable-length renewal frames $r = 0, 1, \ldots$, where each frame holds its configuration fixed over a given number of optimizer updates (20 in our experiments) or

over a complete epoch (Devarakonda et al., 2017; Qiao et al., 2021; Shallue et al., 2019). At the start of frame r, the controller observes a training and renewable-supply state $S(r) \in \mathcal{X}$ and selects, from a finite feasible set $\Pi_{S(r)} \subseteq \Pi$, an action $\pi(r)$ that can set the batch size, the learning rate, or the degree of parallelism. Since these actions take different amounts of time to execute, we compare them per unit time rather than per frame.

For a frame quantity y, the notation $\widehat{y}(S(r), \pi(r))$ makes its dependence on the state and candidate action explicit, and these arguments are omitted when clear from context. Expectations are taken over the randomness in the state, the policy, and training. Appendix C collects the remaining notation.

Time, energy, and error. Let M_{r-1} and M_r be the model parameters before and after frame r, respectively, with initialization M_{-1}. For an objective J such as cross-entropy (Bishop, 2006; Cover & Thomas, 2006), let M^* denote a minimizer. The expected or measured duration of frame r under the selected state and action is $T(r) \coloneqq \widehat{T}\big(S(r), \pi(r)\big)$.

Definition 2.1 (Renewable shortfall per unit time)**.** For frame demand $E^{\mathrm{dem}}(r)$ and renewable generation $E^{\mathrm{ren}}(r)$, define

$$E_s(r) \coloneqq \widehat{E}_s\big(S(r), \pi(r)\big) = \max\{0, E^{\mathrm{dem}}(r) - E^{\mathrm{ren}}(r)\}, \tag{1}$$

$$\overline{P}_R \coloneqq \frac{\sum_{r=0}^{R-1} E_s(r)}{\sum_{r=0}^{R-1} T(r)}. \tag{2}$$

We call $\overline{P}_R$ the time-averaged renewable shortfall (TARS), which has units of power.

This accounting balances energy within each frame and does not resolve instantaneous grid draw. Matching supply and demand within each frame is not the same as matching them in total, because renewable energy can be available in frames with little training demand and absent when computation is intensive. Since excess generation cannot be carried across frames, the aggregate ratio of renewable generation to demand can overstate the renewable energy that training uses, even when total generation is close to total demand (Appendix C relates TARS to aggregate demand and generation). Duration normalization addresses a related issue, since weighting frames equally would give a short and a long training segment the same contribution to the average.

Definition 2.2 (Error averaged over training time)**.** The endpoint error, duration-weighted error, and duration-weighted average error (DWAE) are

$$e_r \coloneqq J(M_r) - J(M^*) \geq 0, \qquad \varepsilon(r) \coloneqq T(r) e_r, \qquad \overline{e}_R \coloneqq \frac{\sum_{r=0}^{R-1} \varepsilon(r)}{\sum_{r=0}^{R-1} T(r)}. \tag{3}$$

DWAE weights the endpoint loss gap of a frame by the duration of that frame, so the same endpoint error contributes more after a long frame than after a short one. It complements update counts and time to a target loss (Dean et al., 2012; Shallue et al., 2019), because it accounts for error over the whole elapsed training time, whereas a target-crossing metric only asks when a particular loss level is first reached. Appendix F relates DWAE to the other metrics.

In the experiments, the unobservable gap e_r is replaced by a normalized validation cross-entropy. If x_r denotes the validation loss at the start of frame r divided by the value at initialization, the endpoint proxy is $e_r = x_{r+1}$. The controller predicts this endpoint for each candidate action before execution and observes its realized value afterward. The analysis requires the following conditions on these frame quantities.

Assumption 2.1. Given $S(r), \pi(r)$, the laws of $T(r)$, $E_s(r)$, $\varepsilon(r)$ do not depend on the past, and for every state and action, $0 < T_{\min} \leq T(r) \leq T_{\max}$, $0 \leq E_{\min} \leq E_s(r) \leq E_{\max}$, and $e_r^2 \leq e_{\max}^2$.

Objective. Writing $\overline{y}(R) \coloneqq \frac{1}{R} \sum_{r=0}^{R-1} \mathbb{E}[y(r)]$ for the expectation-based average of $y \in \{T, E_s, \varepsilon\}$, the controller solves the time-averaged renewable shortfall minimization (TARSM):

$$\underset{\{\pi(r)\}_{r \geq 0}}{\text{minimize}} \quad \limsup_{R \to \infty} \frac{\overline{E}_s(R)}{\overline{T}(R)}, \tag{4}$$

$$\text{subject to} \quad \limsup_{R \to \infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \leq \sigma, \qquad \pi(r) \in \Pi \quad \text{for all } r \geq 0. \tag{5}$$

Algorithm 1 Renewal Lyapunov control for training (LYA)

Require: Feasible policies Π, target σ, trade-off parameter $V > 0$, function ψ
1: Set $Q(0) = 0$
2: **for** $r = 0, 1, 2, \ldots$ **do**
3: Observe the frame state $S(r)$ and backlog $Q(r)$
4: Choose $\pi(r) \in \Pi_{S(r)}$ minimizing the ratio in Equation (7)
5: Execute the frame and observe $T(r)$, $E_s(r)$, and endpoint cost $\varepsilon(r)$
6: $Q(r+1) \leftarrow \max\{Q(r) + \varepsilon(r) - \sigma T(r), 0\}$
7: **end for**

Here σ is the largest admissible DWAE. Although the analysis uses ratios of expected totals, Appendix C gives the corresponding statement for empirical averages, and Section 4 specifies the state and feasibility assumptions behind the guarantees.

3 A RENEWAL-BASED TRAINING CONTROLLER

To couple renewable supply with learning progress on a single time scale, we combine renewal optimization with Lyapunov drift analysis (Neely, 2009; 2010), and we adapt both to a setting in which the length of a decision interval depends on the chosen action.

Encoding the constraint as a queue. We introduce a nonnegative virtual queue that records the accumulated violation of the error budget,

$$Q(r+1) = \max\{Q(r) + \varepsilon(r) - \sigma T(r), 0\}. \tag{6}$$

Endpoint error above σ increases the queue, and error below σ can reduce it. Because the increment is $T(r)(e_r - \sigma)$, a longer frame contributes more to the accumulated violation at the same endpoint error. The backlog therefore records error relative to a budget that grows with elapsed training time. Stability of the queue implies the long-run constraint in Equation (5).

Backlog penalty. To keep the queue stable, we take $\psi : \mathbb{R}_+ \to \mathbb{R}_+$ to be a radially unbounded penalty on the backlog, with the standard quadratic choice $\psi(Q) = Q^2/2$ (Huang & Neely, 2009). Adding the weighted renewable shortfall to the expected drift of ψ and normalizing by the expected frame duration gives the action objective (Li & Neely, 2013)

$$\frac{\mathbb{E}[V E_s(r) + \psi'(Q(r))\varepsilon(r) \mid Q(r), S(r)]}{\mathbb{E}[T(r) \mid Q(r), S(r)]}, \tag{7}$$

where $V > 0$ sets the weight on renewable shortfall relative to the backlog. The normalization makes configurations with different execution times comparable. Because the error term depends on the candidate action, the controller evaluates the expected endpoint cost of each feasible configuration from the current state before execution, and it updates the queue with the observed duration and endpoint error afterward.

Choosing an action per unit time. For a fixed state and backlog, let $\theta^*(r)$ be the minimum attainable ratio in Equation (7). Fractional programming gives an equivalent action choice (Dinkelbach, 1967), namely to minimize

$$\mathbb{E}[V E_s(r) + \psi'(Q(r))\varepsilon(r) - \theta^*(r) T(r) \mid S(r), Q(r)]\,. \tag{8}$$

At $\theta^*(r)$, the minimum of this additive objective is zero, so the ratio value can be found by a search that compares the feasible actions through an additive cost at each step (Appendix H gives the bisection procedure).

Algorithm 1 applies this rule at each renewal boundary, and Appendix C gives the drift expansion and the fractional-programming characterization.

4 GUARANTEES UNDER RENEWABLE-STATE DYNAMICS

Throughout this section we take the backlog penalty of Section 3 to be quadratic, $L(Q) = \psi(Q) = Q^2/2$, and we use the bounded frame quantities of Assumption 2.1. The guarantees compare the

controller with the best policy in a class that differs between independent and correlated states, so we treat the two cases in turn. Because the per-frame ratio is minimized numerically, the guarantees are stated for a C-additive solver, which returns a conditional ratio at most C above its infimum in every frame. Appendix D gives the one-frame drift bounds, and Appendices K–L give the proofs.

4.1 Independent states across frames

We first assume that the full state $S(r)$, including its training component, is i.i.d. across frames. Current actions then do not change the distribution of later states, so the comparison uses single-frame expectations conditioned on $Q(r)$.

The expected backlog drift is $\Delta(Q(r)) = \mathbb{E}[L(Q(r+1)) - L(Q(r)) \mid Q(r)]$. Squaring the queue update gives

$$\Delta(Q(r)) \le \mathcal{U} + Q(r)\mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r)], \qquad \mathcal{U} \ge \tfrac{1}{2}\mathbb{E}\big[(\varepsilon(r) - \sigma T(r))^2 \mid Q(r)\big].$$

Assumption 2.1 makes $\mathcal{U}$ finite, and the second term weights the expected constraint violation by the current backlog. Adding the shortfall penalty and normalizing by the expected frame duration gives the ratio objective in Equation (7).

Theorem 4.1 (Energy and error guarantees for i.i.d. states)**.** *Assume that an i.i.d. policy $\pi^*(r)$ exists with $\mathbb{E}[\varepsilon(r)]/\mathbb{E}[T(r)] \le \sigma$, and define the comparison ratio*

$$\zeta^* := \inf_{\text{i.i.d. policies}} \frac{\mathbb{E}[E_s(r)]}{\mathbb{E}[T(r)]}. \tag{9}$$

With $Q(0) = 0$ and a C-additive solution in every frame, the controller of Algorithm 1 satisfies

$$\limsup_{R\to\infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \le \sigma, \qquad \overline{E}_s(R) - \zeta^*\overline{T}(R) \le \frac{\mathcal{U} + CT_{\max}}{V}. \tag{10}$$

If an i.i.d. policy also satisfies the strict feasibility condition $\mathbb{E}[\widehat{\varepsilon}(\pi^*(r))] \le \mathbb{E}[\widehat{T}(\pi^*(r))](\sigma - \gamma)$ for some $\gamma > 0$, then, for every $R > 0$,

$$\frac{1}{R}\sum_{r=0}^{R-1} \mathbb{E}[Q(r)] \le \frac{VF}{\gamma T_{\min}}, \tag{11}$$

where F is a finite constant.

The constants $\mathcal{U}$ and F are given in Equations (29) and (80). Increasing V decreases the shortfall bound while allowing a larger average backlog. The backlog bound scales with $1/\gamma$ because the strict feasibility margin determines how strongly the comparison policy can compensate for past error-budget violations.

The i.i.d. proof averages over the current state before the comparison ratio is defined, whereas action selection uses the state observed at the frame boundary, and Appendix J shows that the state-conditioned choice attains the same ratio.

4.2 Correlated renewable states

We next allow renewable generation to change gradually across decision boundaries, as it does for temporally correlated solar irradiance (Miao et al., 2018). Since the chosen action also changes the training component of the state, we represent the joint dynamics by a finite-state, time-homogeneous Markov model with policy-dependent kernel P_π, so the comparison must account for the state distribution induced by each policy. To carry out this stationary comparison, beyond Assumption 2.1, we need the following additional conditions.

Assumption 4.1 (Uniform mixing and bounded costs)**.** Under every fixed policy $\pi \in \Pi$, the process $S(r) \in \mathcal{X}$ is a finite-state, irreducible, aperiodic Markov chain whose transition kernel P_π has a unique stationary distribution η^π, and:

(i) there are constants $B > 0$ and $\rho \in (0, 1)$, uniform over policies, such that

$$\sup_{S(0)} \|\Pr\{S(r) \in \cdot \mid S(0)\} - \eta^\pi(\cdot)\|_{\mathrm{TV}} \le B\rho^r \quad \text{for every } \pi \in \Pi; \tag{12}$$

(ii) Assumption 2.1 holds for every feasible policy and state.

Costs are now conditioned jointly on $S(r)$ and $Q(r)$, and their stationary averages are taken under η^π, because a policy affects both the current costs and the distribution of future states. The backlog bound additionally uses a stationary Markov Slater condition (Assumption L.1).

Theorem 4.2 (Energy and error guarantees for Markov states)**.** *Assume there is a stationary Markov policy π^* depending only on the current state for which $\mathbb{E}_{\eta^{\pi^*}}[\varepsilon(r)]/\mathbb{E}_{\eta^{\pi^*}}[T(r)] \le \sigma$, and define*

$$\zeta^* := \inf_{\text{stationary Markov policies } \pi} \frac{\mathbb{E}_{\eta^\pi}[E_s(r)]}{\mathbb{E}_{\eta^\pi}[T(r)]}. \tag{13}$$

If Assumption 4.1 holds and the controller uses a C-additive approximation to the per-frame drift-plus-penalty ratio, then the steady-state guarantees are

$$\limsup_{R\to\infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \le \sigma, \qquad \overline{E}_s(R) - \zeta^* \overline{T}(R) \le \frac{\mathcal{U} + CT_{\max}}{V} + o(1). \tag{14}$$

The term $o(1)$ tends to zero as $R \to \infty$ under geometric ergodicity.

The shortfall bound keeps the dependence on V from the independent-state case, and the additional $o(1)$ term accounts for the approach to stationarity, which the total variation bound in Assumption 4.1 controls uniformly over fixed policies. Under the Slater condition, the same argument bounds the average backlog by $VF/(\gamma T_{\min})$ as in Theorem 4.1. Appendix L gives that bound and the stationary comparison argument.

5 Adapting the backlog penalty to training phases

Training loss can fall rapidly at first and then approach a plateau, and Appendix D.3 illustrates both regimes with measured CNN trajectories. Since training responds differently to the same action in these two regimes, we let the backlog price, which the controller of Section 3 keeps fixed, depend on the regime that training has reached.

Phase as part of the state. To let the controller respond differently to the same backlog at different stages of training, we add a phase indicator $\phi(r) \in \{1, 2\}$ to the state, $\widetilde{S}(r) := \langle S(r), \phi(r)\rangle$, where phase 1 denotes rapid error reduction and phase 2 denotes slow reduction or a plateau.

Detecting the phase. The phase used in frame r is fixed before that frame begins, so an action never depends on information that is not yet available. After the frame we compute $\Delta_e(r) := (e_r - e_{r-1})/e_{r-1}$ for $e_{r-1} > 0$, where e_{-1} is the initial error, and set $\phi(r+1) = 1$ if $\Delta_e(r) \le -\tau$ and $\phi(r+1) = 2$ otherwise. We initialize $\phi(0)$ before training, and the experiments use validation loss, $\phi(0) = 1$, and $\tau = 0.1$.

Phase-weighted penalty. We then weight the backlog by the phase,

$$L(Q(r), \phi(r)) := \frac{1}{2} w_{\phi(r)} Q(r)^2, \qquad w_{\phi(r)} > 0. \tag{15}$$

A choice $w_1 \ge w_2$ places more weight on the backlog during rapid descent, so the same accumulated violation can influence action selection differently across training phases. The error metric and queue update retain their definitions, while the coefficient multiplying the error cost becomes $w_{\phi(r)}Q(r)$. Holding the current phase fixed in the one-step drift yields the bound in Appendix D.4, and minimizing the resulting ratio gives the phase-aware rule. The controller keeps the queue update of Algorithm 1 and selects

$$\pi(r) \in \arg\min_{\pi \in \Pi_{S(r)}} \mathbb{E}\big[VE_s(r) + w_{\phi(r)}Q(r)\varepsilon(r) - \theta^*(r)T(r) \mid Q(r), \widetilde{S}(r)\big], \tag{16}$$

with

$$\theta^*(r) := \inf_{\pi \in \Pi_{S(r)}} \frac{\mathbb{E}[VE_s(r) + w_{\phi(r)}Q(r)\varepsilon(r) \mid Q(r), \widetilde{S}(r)]}{\mathbb{E}[T(r) \mid Q(r), \widetilde{S}(r)]}. \tag{17}$$

Assumption 4.1 required an irreducible chain, which training dynamics do not satisfy in general. If a discretized loss process enters a low-loss region from which it cannot return to higher-loss states,

those higher-loss states are transient, and closed low-loss regions form recurrent classes. Once training enters a class it remains there, so the long-run comparison must be made within the class that training reaches.

Theorem 5.1 (Performance conditional on a recurrent training class)**.** *Consider the finite-state, time-homogeneous, aperiodic extended process $\widetilde{S}(r) = (S(r), \phi(r))$ under the phase-aware controller. The process may be reducible, so let $\mathcal{C}$ be the recurrent class entered in finite time almost surely. Suppose the following hold within $\mathcal{C}$.*

1. *Every stationary phase-aware Markov policy π induces an irreducible, aperiodic subchain with a unique stationary distribution $\eta_{\mathcal{C}}^{\pi}$.*
2. *For some $\gamma > 0$, a stationary phase-aware policy $\pi_{\mathcal{C}}^{\gamma}$ supported on $\mathcal{C}$ satisfies*

$$\mathbb{E}_{\eta_{\mathcal{C}}^{\pi_{\mathcal{C}}^{\gamma}}}[\varepsilon(\widetilde{S}, \pi_{\mathcal{C}}^{\gamma}(\widetilde{S}))] \le \mathbb{E}_{\eta_{\mathcal{C}}^{\pi_{\mathcal{C}}^{\gamma}}}[T(\widetilde{S}, \pi_{\mathcal{C}}^{\gamma}(\widetilde{S}))](\sigma - \gamma). \tag{18}$$

3. *Each frame uses a C-additive approximation to the ratio minimization restricted to $\mathcal{C}$.*

Let the class-specific comparison ratio be

$$\zeta_{\mathcal{C}}^{*} := \inf_{\substack{\text{stationary phase-aware Markov} \\ \text{policies } \pi \text{ on } \mathcal{C}}} \frac{\mathbb{E}_{\eta_{\mathcal{C}}^{\pi}}[E_s(\widetilde{S}, \pi(\widetilde{S}))]}{\mathbb{E}_{\eta_{\mathcal{C}}^{\pi}}[T(\widetilde{S}, \pi(\widetilde{S}))]}. \tag{19}$$

Conditional on entering $\mathcal{C}$, the steady-state bounds are

$$\limsup_{R\to\infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \le \sigma, \qquad \overline{E}_s(R) \le \frac{\mathcal{U}_{\max} + CT_{\max}}{V} + \zeta_{\mathcal{C}}^{*}\overline{T}(R) + o(1). \tag{20}$$

Here $\mathcal{U}_{\max}$ is the uniform drift bound, and $o(1) \to 0$ as $R \to \infty$ through mixing within the class. If every initial state reaches the same recurrent class almost surely, the bounds also hold without conditioning on that event.

The bound does not identify which low-loss class training reaches. Appendix M gives the proof, and Appendix D.5 explains the class interpretation.

6 Experimental evaluation

We evaluate the controller on a CNN training workload, first under an independent renewable supply and then on a measured photovoltaic day, reporting both full-budget metrics and the time to a common quality target.

Protocol. We train a CNN on 55,000 MNIST examples with 5,000 validation examples, using an NVIDIA A800. Each run executes 100 frames of 20 Adam updates, retaining model and optimizer states across batch changes. We set $\sigma = 0.05$ and $(w_1, w_2) = (100, 1)$, and LYA chooses from batches $\{16, 32, 64, 128, 256, 512\}$. Baselines use fixed batches (B16, B128, B512), switch from 16 to 512 halfway (STA), or traverse the candidate set upward (INC) or downward (DEC). Three initialization seeds are crossed with eight independent supply seeds. Because the batch choices of the baselines do not depend on the supply, their training trajectories are reused across supply inputs, while each adaptive run executes its own GPU updates. Reported training time excludes validation and controller overhead, which are logged separately. Appendices N–P give the calibration and full protocols.

Estimating progress and cost. The controller needs an estimate of the effect of each candidate action before executing it. A steps-to-target curve gives the number of updates that a fixed batch size requires, but not the change in loss produced by one further block at an intermediate state, so we measure that transition directly. From nine checkpoints on each of nine reference trajectories, each of the six candidate batch sizes runs 20 updates from the same model, optimizer, and data state, and the resulting 486 paired transitions are used to fit a predictor $F_B(x_r)$ of the endpoint error under batch size B (Figure 1 and Appendix N.1). On 3,000 held-out blocks, one-block prediction has RMSE 0.0161 in normalized loss units, while applying the same predictor recursively without new observations gives 0.0331. The controller therefore uses F_B only to score candidates and takes the measured endpoint x_{r+1} as feedback, so the reported learning curves and target times are obtained from GPU training rather than from the predictor.

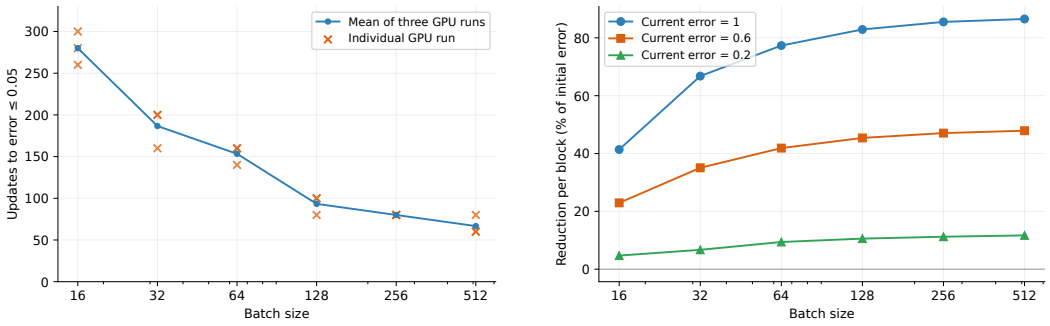


Figure 1: Batch size and training progress. Left: measured updates to normalized validation error 0.05 under fixed batch sizes, for three initialization seeds. Right: the reduction predicted over one 20-update block at current errors 1.0, 0.6, and 0.2.

Table 1: Independent supply, $\bar{V} = 1$. Means over 24 paired initialization–supply combinations. TARS and DWAE cover all 100 frames. Target rows include the first frame ending at error ≤ 0.05.

	LYA	B16	B128	B512	STA	INC	DEC
TARS (W)	2.284	2.212	8.307	70.762	47.753	27.325	26.180
DWAE	0.02542	0.04451	0.02110	0.01774	0.03114	0.03057	0.02338
Target time (s)	0.501	1.243	0.442	0.479	1.215	1.201	0.456
Target shortfall (J)	3.496	2.985	4.008	31.469	2.941	2.924	29.939
Target examples (10^3)	17.307	4.693	11.093	30.720	4.693	4.800	30.720

The controller also needs the duration and the power of each candidate batch size. We measure both in a two-epoch pilot on the same device, where a larger batch shortens an epoch but draws more power, and fit them by $Y(B) = a_Y B^{-k_Y} + c_Y$ for $Y \in \{T, P\}$ (Appendix N). Writing $T_{\mathrm{ref}} = 0.09693\,\mathrm{s}$ and $P_{\mathrm{ref}} = 94.483\,\mathrm{W}$ for the calibrated block duration and power at batch 128, and $q = Q/T_{\mathrm{ref}}$ for the normalized backlog, Equation (16) becomes the finite-action score

$$\bar{V}\,\frac{\max\{P(B) - P_{\mathrm{ren}}(r), 0\}}{P_{\mathrm{ref}}} + w_{\phi(r)}\,q(r)\,F_B(x_r), \qquad \bar{V} = V\,P_{\mathrm{ref}}/T_{\mathrm{ref}}. \tag{21}$$

This is the ratio objective after a change of units, since dividing both frame costs by the frame duration removes that duration from the score, and P_{ref} and T_{ref} only rescale the two terms. After execution the queue receives the measured increment $[T(r)/T_{\mathrm{ref}}](x_{r+1} - \sigma)$.

Independent supply. In this scenario, renewable power is drawn independently in each frame from a uniform distribution on $[47.24, 188.97]\,\mathrm{W}$ and held through the frame, and all policies share the same frame-indexed draws. Table 1 separates full-budget metrics from time to a common target. LYA and B16 have similar shortfall rates, with a paired difference of 0.072 W and a 95% bootstrap interval of $[-0.128, 0.331]\,\mathrm{W}$ (the bootstrap treats initialization and supply seeds as separate factors and preserves policy pairs). Relative to B16, LYA reduces DWAE by 42.9% (paired difference -0.01909, 95% interval $[-0.02026, -0.01791]$) and target time by 59.7%, but it reaches the target with more examples and more shortfall energy, because its larger batches process more data per update and draw more power. Because all runs continue after reaching the target, faster target attainment does not imply a shorter complete run, and the full run of LYA takes 9.738 seconds against 8.563 for B16. Compared with B128, LYA reduces TARS by 72.5% with a 13.4% longer target time, while B512 and DEC reach the target sooner than LYA but incur larger shortfall rates. These differences arise because the controller selects a more demanding batch when the current supply covers it, and error feedback changes the choices made early in training.

Phase weights. Under one shared supply seed and three initializations, weights $(100, 1)$, $(1, 1)$, and $(1, 100)$ give mean target times of 0.498, 0.536, and 0.531 seconds, so early emphasis improves the mean target time by 7.0% relative to equal weights, although the per-seed ranges overlap. Full-budget DWAE stays between 0.02531 and 0.02612, so the weight choice mainly changes transient batch decisions and target crossings (Appendix O).

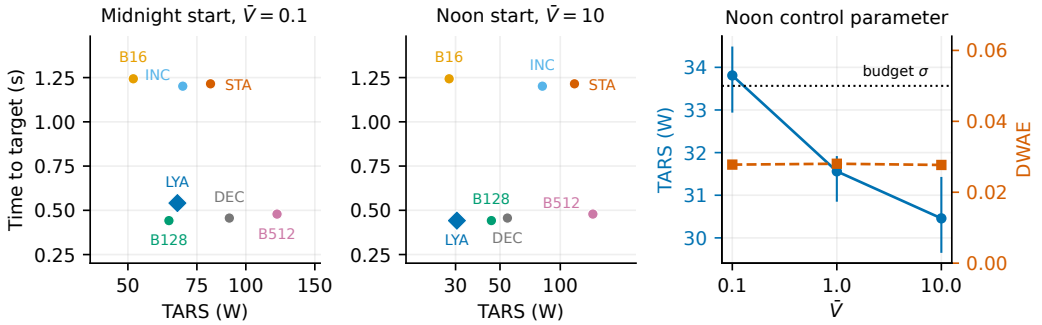


Figure 2: Measured PV scenarios, as means over three initialization seeds. Left and center: shortfall rate against time to the common target, where lower and further left is better, with LYA marked by a diamond. Right: the control parameter at noon, with bars spanning the per-seed range.

Daily photovoltaic supply. We next replay a complete residential PV day from Open Power System Data (Open Power System Data, 2020), scaled so that its peak equals $2P_{\text{ref}} = 188.97$ W, with 100 reference blocks mapped to 12 calendar hours (Appendix P.5). The controller observes only the current power when it chooses an action, while the executed frame integrates the two-minute trace over its measured duration.

Figure 2 places each policy in the plane of shortfall rate against target time for a noon start with $\bar{V} = 10$ and a midnight start with $\bar{V} = 0.1$. At noon, LYA attains 30.46 W against 45.28 W for B128, a reduction of 32.7% at the same target time of 0.442 seconds. It also reaches the target 64.5% sooner than B16, at a shortfall rate 9.3% higher. The midnight start differs because its mapped interval begins with several hours of near-zero generation, and there B128 attains both a lower shortfall rate (63.63 against 66.89 W) and a lower DWAE (0.02110 against 0.03255) than LYA. When supply is absent for much of a run, the timing of demand has little to exploit, and the adaptive policy pays for its early large-batch choices without an energy benefit. At the same $\bar{V} = 0.1$, a noon start lowers the mean shortfall rate of LYA from 66.89 to 33.81 W, so the gap between the two scenarios comes mainly from supply timing rather than from $\bar{V}$.

The right panel varies the control parameter at noon. The mean TARS falls from 33.81 to 30.46 W as $\bar{V}$ increases from 0.1 to 10, a reduction of 9.9% in the direction that the $O(1/V)$ bound predicts, and the per-seed ranges at the two endpoints do not overlap. The DWAE varies by 1.2% of its mean over the same sweep and remains well below the budget, so this range does not exercise the error side of the tradeoff. Scaling the PV peak from 0.5 to 4 times P_{ref} at noon with $\bar{V} = 10$ lowers the DWAE of LYA from 0.03999 to 0.02594, whereas the fixed schedules keep their error because their batch choices do not depend on supply (Appendix P.5 also reports the shortfall rates). Decision traces and the full policy comparison for both scenarios are in Appendix O.

7 Conclusion

We treated renewable-aware training as a renewal control problem, in which the duration, the energy cost, and the learning effect of an action are accounted for together. We enforced a duration-weighted error budget with a virtual queue, let phase weights adjust the influence of that budget on scheduling, and bounded the tradeoff between shortfall and backlog. In closed-loop GPU training, the controller reaches a common quality target in 59.7% less time than the lowest-power fixed batch at a comparable shortfall rate, and on a photovoltaic day with a noon start it attains a 32.7% lower shortfall rate than batch 128 at the same target time.

Three limitations qualify these results. The workload is a small convolutional model, so the calibrated predictor and the cost curves would have to be re-measured for a larger one. The daily supply cycle is replayed through a fixed time mapping rather than executed over twelve hours, which preserves the shape of the input but not its absolute time scale. The error budget is slack in every run, so the sweep over the control parameter shows the predicted decrease in shortfall but not the backlog cost that a tighter budget would incur, which makes a binding error budget the most direct next test.

Reproducibility statement

The appendices provide the proofs, model configuration, and measurement protocols. The experimental protocol uses disjoint sets of three seeds for calibration, predictor validation, and policy evaluation, and specifies the candidate transition models, scheduling rules, and uncertainty estimates. The accompanying code records training losses, elapsed times, random seeds, and software versions, and generates figures from the saved observations. For the public photovoltaic input, we document its source, selection rule, time zone, conversion from cumulative energy to interval-average power, and scaling to the GPU power and training-time ranges.

AI use statement

Generative AI assisted with editing the prose, preparing the LaTeX source, implementing experimental and plotting code, and checking consistency and formatting. The research problem, algorithms, and theoretical results are the work of the authors.

References

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Sophia Chen. How much energy will AI really consume? The good, the bad and the unknown. *Nature*, 639(8053):22–24, 2025.

Ying Chen, Yaozong Yang, Yuan Wu, Jiwei Huang, and Lian Zhao. Joint Trajectory Optimization and Resource Allocation in UAV-MEC Systems: A Lyapunov-Assisted DRL Approach. *IEEE Transactions on Services Computing*, 18(2):854–867, 2025. doi: 10.1109/TSC.2025.3544124.

Jeremy M. Cohen, Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jh-rTtvkGeM.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

Aditya Devarakonda, Maxim Naumov, and Michael Garland. AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv preprint arXiv:1712.02029*, 2017.

Yi Ding and Tianyao Shi. Sustainable LLM Serving: Environmental Implications, Challenges, and Opportunities. In *IEEE International Green and Sustainable Computing Conference*, pp. 37–38, 2024.

Werner Dinkelbach. On Nonlinear Fractional Programming. *Management Science*, 13(7):492–498, 1967.

Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Chukwunyere Osi, Prateek Sharma, Fan Chen, and Lei Jiang. LLMCarbon: Modeling the End-to-End Carbon Footprint of Large Language Models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aIok3ZD9to.

Giovanni Fonseca and R. L. Tweedie. Stationary Measures for Non-Irreducible Non-Continuous Markov Chains with Time Series Applications. *Statistica Sinica*, 12(2):651–660, 2002.

Daniel Geißler, Bo Zhou, Mengxi Liu, Sungho Suh, and Paul Lukowicz. The Power of Training: How Different Neural Network Setups Influence the Energy Demand. In *International Conference on Architecture of Computing Systems*, pp. 33–47. Springer, 2024.

Stefanos Georgiou, Maria Kechagia, Tushar Sharma, Federica Sarro, and Ying Zou. Green AI: Do Deep Learning Frameworks Have Different Costs? In *International Conference on Software Engineering*, pp. 1082–1094, 2022. doi: 10.1145/3510003.3510221.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677*, 2017.

Longbo Huang and Michael J. Neely. Delay Reduction via Lagrange Multipliers in Stochastic Network Optimization. In *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pp. 1–10, 2009. doi: 10.1109/WIOPT.2009.5291609.

Longbo Huang and Michael J. Neely. Max-Weight Achieves the Exact [O(1/V), O(V)] Utility-Delay Tradeoff Under Markov Dynamics. *arXiv preprint arXiv:1008.0200*, 2010. URL https://arxiv.org/abs/1008.0200.

Wenxuan Huang, Yunhang Shen, Jiao Xie, Baochang Zhang, Gaoqi He, Ke Li, Xing Sun, and Shaohui Lin. A General and Efficient Training for Transformer via Token Expansion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15783–15792, 2024.

Sunbal Iftikhar and Steven Davy. Reducing Carbon Footprint in AI: A Framework for Sustainable Training of Large Language Models. In *Future Technologies Conference*, volume 1, pp. 325–336. Springer, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*, 2020.

Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The MNIST Database of Handwritten Digits, 2010. URL http://web.archive.org/web/20241204064952/https://yann.lecun.com/exdb/mnist/index.html.

Chih-ping Li and Michael J. Neely. Network Utility Maximization over Partially Observable Markovian Channels. *Performance Evaluation*, 70(7–8):528–548, 2013. doi: 10.1016/j.peva.2012.10.003.

Ran Li, Chuan Huang, Xiaoqi Qin, Zhanhong Fu, Lei Yang, and Dong Yang. AoI-Delay Tradeoff in Mobile Edge Caching: A Lyapunov Optimization-Based Method. *IEEE Internet of Things Journal*, 12(11):16266–16281, 2025. doi: 10.1109/JIOT.2025.3531415.

Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and Cooling Aware Workload Management for Sustainable Data Centers. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, pp. 175–186, 2012. doi: 10.1145/2254756.2254779.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model. *Journal of Machine Learning Research*, 24(253):1–15, 2023.

Yu Luo, Lina Pu, and Chun-Hung Liu. Computing Power and Battery Charging Management for Solar Energy Powered Edge Computing. *IEEE Transactions on Mobile Computing*, 24(3):1913–1927, 2025. doi: 10.1109/TMC.2024.3489028.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An Empirical Model of Large-Batch Training. *arXiv preprint arXiv:1812.06162*, 2018.

Sean P. Meyn and Richard L. Tweedie. Stability of Markovian Processes II: Continuous-Time Processes and Sampled Chains. *Advances in Applied Probability*, 25(3):487–517, 1993.

Shuwei Miao, Guangtao Ning, Yingzhong Gu, Jiahao Yan, and Botao Ma. Markov Chain Model for Solar Farm Generation and Its Application to Generation Performance Evaluation. *Journal of Cleaner Production*, 186:905–917, 2018. doi: 10.1016/j.jclepro.2018.03.173.

D. Moreno, J. Friedrich, M. Wächter, J. Schwarte, and J. Peinke. Periods of Constant Wind Speed: How Long Do They Last in the Turbulent Atmospheric Boundary Layer? *Wind Energy Science*, 10(2):347–360, 2025. doi: 10.5194/wes-10-347-2025.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Michael J. Neely. Stochastic Optimization for Markov Modulated Networks with Application to Delay Constrained Wireless Scheduling. In *IEEE Conference on Decision and Control*, pp. 4826–4833, 2009.

Michael J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

Michael J. Neely. Fast Learning for Renewal Optimization in Online Task Scheduling. *Journal of Machine Learning Research*, 22(279):1–44, 2021.

James O'Donnell and Casey Crownhart. We Did the Math on AI's Energy Footprint. Here's the Story You Haven't Heard. MIT Technology Review, May 2025. URL https://www.technologyreview.com/2025/05/20/1116327/ai-energy-usage-climate-footprint-big-tech/.

Open Power System Data. Data Package Household Data, 2020. URL https://data.open-power-system-data.org/household_data/2020-04-15/. Version 2020-04-15. Primary measurements from CoSSMic, ISC Konstanz. CC BY 4.0.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. Characterizing Power Management Opportunities for LLMs in the Cloud. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, volume 3, pp. 207–222, 2024.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2014.

Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *USENIX Symposium on Operating Systems Design and Implementation*, pp. 1–18, 2021. URL https://www.usenix.org/conference/osdi21/presentation/qiao.

Chengrun Qiu, Yang Hu, Yan Chen, and Bing Zeng. Lyapunov Optimization for Energy Harvesting Wireless Sensor Communications. *IEEE Internet of Things Journal*, 5(3):1947–1956, 2018. doi: 10.1109/JIOT.2018.2817590.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Power-aware Deep Learning Model Serving with μ-Serve. In *USENIX Annual Technical Conference*, pp. 75–93, 2024. URL https://www.usenix.org/conference/atc24/presentation/qiu.

Ana Radovanović, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, et al. Carbon-Aware Computing for Datacenters. *IEEE Transactions on Power Systems*, 38(2):1270–1280, 2023. doi: 10.1109/TPWRS.2022.3173250.

Levent Sagun, Utku Evci, V. Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical Analysis of the Hessian of Over-Parametrized Neural Networks. *arXiv preprint arXiv:1706.04454*, 2017.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the Effects of Data Parallelism on Neural Network Training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

Ao Shen, Zhiquan Lai, Tao Sun, Shengwei Li, Keshi Ge, Weijie Liu, and Dongsheng Li. Efficient Deep Neural Network Training via Decreasing Precision with Layer Capacity. *Frontiers of Computer Science*, 19(10):1910355, 2025.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Inigo Goiri, and Josep Torrellas. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. *arXiv preprint arXiv:2403.20306*, 2024.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. In *IEEE International Symposium on High Performance Computer Architecture*, pp. 1348–1362, 2025. doi: 10.1109/HPCA61900.2025.00102.

Rahul Urgaonkar and Michael J. Neely. Opportunistic Scheduling with Reliability Guarantees in Cognitive Radio Networks. *IEEE Transactions on Mobile Computing*, 8(6):766–777, 2009. doi: 10.1109/TMC.2009.38.

Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-Train: Training State-of-the-art CNNs with Over 80% Less Energy. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://papers.nips.cc/paper_files/paper/2019/file/663772ea088360f95bac3dc7ffb841be-Paper.pdf.

Jiajie Xu, Kaixin Li, Ying Chen, and Jiwei Huang. Optimal Task Scheduling and Resource Allocation for Self-Powered Sensors in Internet of Things: An Energy Efficient Approach. *IEEE Transactions on Network and Service Management*, 21(4):4410–4420, 2024. doi: 10.1109/TNSM.2024.3420254.

Kaiqiang Xu, Decang Sun, Han Tian, Junxue Zhang, and Kai Chen. GREEN: Carbon-efficient Resource Scheduling for Machine Learning Clusters. In *USENIX Symposium on Networked Systems Design and Implementation*, pp. 999–1014, 2025. URL https://www.usenix.org/conference/nsdi25/presentation/xu-kaiqiang.

Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. In *USENIX Symposium on Networked Systems Design and Implementation*, pp. 119–139, 2023. URL https://www.usenix.org/conference/nsdi23/presentation/you.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Syx4wnEtvH.

Wen Zhang, Chen Pan, Tao Liu, Jeff (Jun) Zhang, Mehdi Sookhak, and Mimi Xie. Intelligent Networking for Energy Harvesting Powered IoT Systems. *ACM Transactions on Sensor Networks*, 20(2):45:1–45:31, 2024. doi: 10.1145/3638765.

A Related work

Energy and efficient learning. Studies of sustainable machine learning examine the energy cost of frameworks, training configurations, and model serving (Ding & Shi, 2024; Geißler et al., 2024; Georgiou et al., 2022; Stojkovic et al., 2024). Changes to training computation, transformer tokens, or numerical precision can also save energy by making training faster or less expensive (Huang et al., 2024; Wang et al., 2019; Shen et al., 2025). Beyond the computational cost of reaching an accuracy target, our objective includes the timing of renewable supply.

Resource management under changing power supply. GPU power management, energy-efficient serving, and training configuration search provide complementary controls at the system level (Patel et al., 2024; Qiu et al., 2024; Stojkovic et al., 2025; You et al., 2023). Renewable-aware data-center scheduling and carbon-aware workload management consider how demand can follow the available energy mix (Liu et al., 2012; Radovanović et al., 2023; Xu et al., 2025). Related questions arise in solar-powered edge computing and energy-harvesting sensors (Luo et al., 2025; Xu et al., 2024; Zhang et al., 2024). Here, the scheduling state also includes the current training error and its phase-dependent response to an action.

Adaptive batch size and training progress. Adaptive batch schedules and cluster scheduling use the dependence of learning progress on batch size to improve training efficiency (Devarakonda et al., 2017; McCandlish et al., 2018; Qiao et al., 2021; Shallue et al., 2019). We use this dependence as an input to online energy control, where a state-dependent transition predictor compares the next-block loss under different batch sizes. The controller combines this prediction with runtime and power estimates and then updates its virtual queue with the observed loss.

Lyapunov and renewal control. Our method draws on stochastic network optimization and renewal scheduling (Neely, 2010; 2021). Virtual queues and drift-plus-penalty decisions have been used for energy harvesting, resource allocation, and service constraints (Chen et al., 2025; Huang & Neely, 2009; Li et al., 2025; Qiu et al., 2018). Markov-state extensions study related utility and delay tradeoffs under dependent environmental states (Huang & Neely, 2010; Neely, 2009; Li & Neely, 2013; Urgaonkar & Neely, 2009). Training introduces an additional modeling issue, because a discretized loss process may leave high-loss states and remain in a closed low-loss class. The phase-aware formulation handles this structure with class-specific stationary quantities (Fonseca & Tweedie, 2002; Meyn & Tweedie, 1993; Puterman, 2014).

B Motivating measurements

Changing the batch size during a run illustrates the coupling between power, epoch duration, and loss. Figure 3 shows a BERT masked-language-model pilot on an NVIDIA A800, with batch size 64 for three epochs and 128 for the next seven. Mean epoch duration decreases from 16.80 to 14.29 seconds, while mean power sampled within training intervals increases from 242.9 to 261.6 watts. Loss falls rapidly early in the run and more slowly later. These measurements motivate accounting for power demand, frame duration, and the training state together.

C Model details and controller derivation

C.1 Notation and frame interpretation

The first R renewal frames have indices $0, \ldots, R-1$. The analysis uses frame-level quantities and does not require a particular number of updates per frame. The state can include model size, batch size, current loss, and available renewable generation.

For any frame quantity, the notation $y(r) = \widehat{y}(S(r), \pi(r))$ makes its dependence on the state and policy explicit. We omit these arguments when the selected state–policy pair is clear. Scalars use ordinary italic letters, and calligraphic letters denote sets. For an integer n, the convention $[n] = \{0, 1, \ldots, n\}$ includes both endpoints. The pair $\langle A, B \rangle$ specifies its first and second components in order. The maximum of a finite set uses its usual ordering. Expectations and probabilities are taken over the randomness in the state, policy, and exogenous inputs. For distributions μ and ν, the total

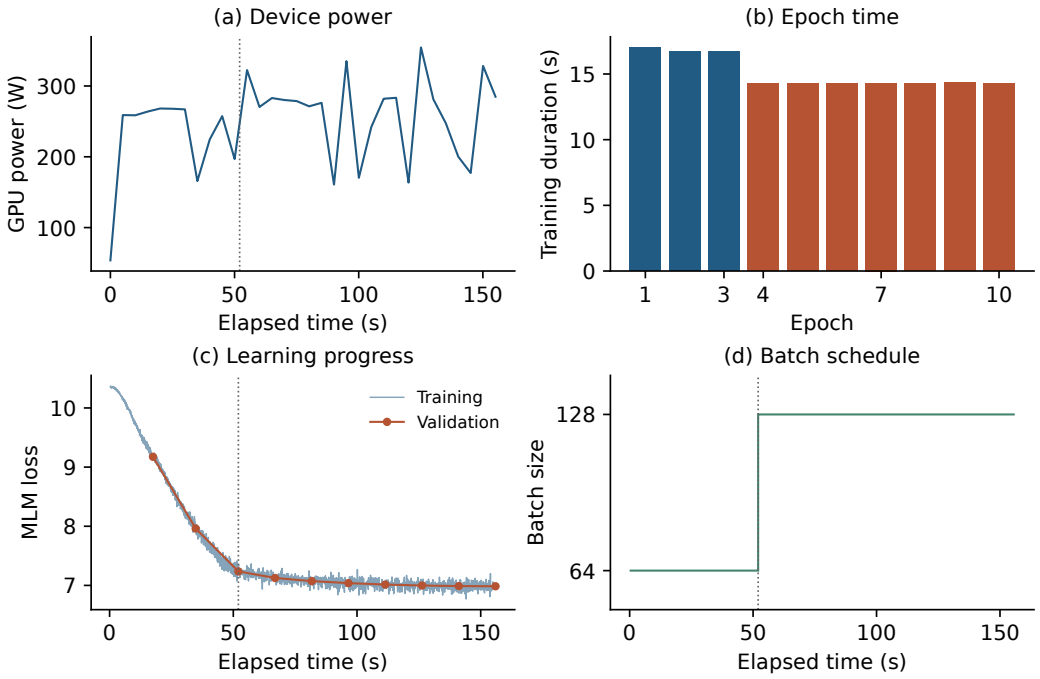


Figure 3: BERT training on WikiText-2 using an NVIDIA A800. Power is sampled through NVML every five seconds, and the dotted line marks the switch from batch size 64 to 128. Appendix P.1 gives the protocol.

variation distance is

$$\|\mu - \nu\|_{\mathrm{TV}} = \sup_{A} |\mu(A) - \nu(A)|,$$

where A ranges over measurable events. Appendix E collects the notation.

C.2 Frame-level and aggregate energy accounting

TARS measures residual demand under frame-level energy accounting. The same model can include a time-varying electricity price by placing the price in $S(r)$ and scaling the energy penalty accordingly. Duration normalization matters because weighting frames equally would give short and long segments the same importance. Dividing accumulated shortfall by elapsed time instead gives the corresponding rate.

Total renewable production alone does not capture whether supply is available when training needs it. To see this, define

$$E_{\mathrm{tot}} = \sum_{r=0}^{R-1} E^{\mathrm{dem}}(r), \qquad E_{\mathrm{ren,tot}} = \sum_{r=0}^{R-1} E^{\mathrm{ren}}(r), \qquad T_{\mathrm{tot}} = \sum_{r=0}^{R-1} T(r).$$

Then

$$\overline{P}_R = \frac{\sum_{r=0}^{R-1} \max\{0, E^{\mathrm{dem}}(r) - E^{\mathrm{ren}}(r)\}}{T_{\mathrm{tot}}} \geq \frac{\max\{0, E_{\mathrm{tot}} - E_{\mathrm{ren,tot}}\}}{T_{\mathrm{tot}}}. \tag{22}$$

If no frame has excess renewable supply, the numerator reduces to $E_{\mathrm{tot}} - E_{\mathrm{ren,tot}}$, so equality holds. If excess generation in some frames cannot be carried into other frames, grid energy may still be needed even when total generation is close to total demand. Thus, the aggregate ratio $E_{\mathrm{ren,tot}}/E_{\mathrm{tot}}$ can overstate the renewable energy that training uses.

C.3 Empirical and expectation-based averages

The uniform bounds in Assumption 2.1 describe operation with finite resources, bounded energy quantities, and bounded training errors. They also give bounded second moments for the queue increments. The infinite-horizon formulation describes sustained operation and reduces the effect of finite-horizon fluctuations in state and frame duration. The drift analysis uses ratios of expected totals, which the next proposition relates to empirical averages.

Proposition C.1 (Agreement of empirical and expected averages)**.** *Under Assumption 2.1, define*

$$\widehat{T}(R) \coloneqq \frac{1}{R}\sum_{r=0}^{R-1} T(r).$$

As $R \to \infty$, $\widehat{T}(R) - \overline{T}(R) \to 0$ almost surely. The same conclusion applies to the empirical and expectation-based averages of $E_s(r)$ and $\varepsilon(r)$. Consequently, the empirical and expectation-based versions of the optimization problem have the same asymptotic optimal value, and sequences of empirical optimizers converge to the same solution set.

Appendix G proves Proposition C.1, and Section 4 separately specifies the state models required by the control analysis.

C.4 General backlog penalty and conditional drift

To measure backlog growth, let

$$L(Q(r)) \coloneqq \psi(Q(r)), \tag{23}$$

where $\psi : \mathbb{R}_+ \to \mathbb{R}_+$ is radially unbounded. The standard quadratic choice is $\psi(x) = x^2/2$ (Huang & Neely, 2009). Other choices can change the sensitivity to backlog size, for example through piecewise penalties that weight violations differently.

At a frame boundary, the conditional drift is

$$\Delta(Q(r)) \coloneqq \mathbb{E}[L(Q(r+1)) - L(Q(r)) \mid Q(r), S(r)]\,. \tag{24}$$

It describes the expected change in the backlog penalty over one frame. For smooth ψ, a first-order expansion gives the drift bound used by the framework,

$$\Delta(Q(r)) \le \mathcal{U}_\psi + \psi'(Q(r))\mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r), S(r)], \tag{25}$$

for a finite remainder bound $\mathcal{U}_\psi$. Appendix D provides the bounds for the quadratic and phase-weighted choices analyzed in this work.

C.5 An additive form of the ratio objective

For a given state and queue, denote the infimum of Equation (7) by $\theta^*(r)$. The policy can then be selected by minimizing the additive objective in Equation (8). The characterization of fractional programs in Dinkelbach (1967) gives

$$\inf_{\pi \in \Pi_{S(r)}} \mathbb{E}[V E_s(r) + \psi'(Q(r))\varepsilon(r) - \theta^*(r) T(r) \mid S(r), Q(r)] = 0. \tag{26}$$

Thus, the optimal ratio identifies an equivalent policy-selection problem with an additive objective. Appendix H gives a bisection procedure for finding this ratio.

The controller determines $\theta^*(r)$, chooses a feasible minimizer of Equation (8), executes that frame, and updates the queue using Equation (6).

D One-frame drift calculations and phase-model details

This appendix covers the one-frame drift bounds, ratio conditioning, and phase-model details used in Sections 4 and 5. The performance proofs are in Appendices K–M.

D.1 Independent-state drift and ratio conditioning

The i.i.d. model idealizes an energy process whose frame-level observations have little temporal dependence, for example under approximately steady wind conditions (Moreno et al., 2025). The analysis, however, assumes independence for the full state process.

For this model, we average over the current state and condition the drift only on the queue,

$$\Delta(Q(r)) \coloneqq \mathbb{E}[L(Q(r+1)) - L(Q(r)) \mid Q(r)]. \tag{27}$$

The queue update gives the following one-frame bound.

Theorem D.1 (Quadratic drift with independent states)**.** *For every frame and backlog,*

$$\Delta(Q(r)) \le \mathcal{U} + Q(r)\mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r)], \tag{28}$$

where the finite constant $\mathcal{U}$ satisfies

$$\mathcal{U} \ge \frac{1}{2}\mathbb{E}[(\varepsilon(r) - \sigma T(r))^2 \mid Q(r)] \quad \textit{for all } r \textit{ and } Q(r). \tag{29}$$

The proof appears in Appendix I. Adding the expected shortfall penalty to Equation (28) yields

$$\begin{aligned}\Delta(Q(r)) + V\mathbb{E}[E_s(r) \mid Q(r)] \le \mathcal{U} + V\mathbb{E}[E_s(r) \mid Q(r)] \\ + Q(r)\mathbb{E}[\varepsilon(r) \mid Q(r)] - \sigma Q(r)\mathbb{E}[T(r) \mid Q(r)].\end{aligned} \tag{30}$$

Algorithm 1 minimizes the corresponding single-frame expression after observing $S(r)$ and $Q(r)$. In the i.i.d. analysis, its ratio parameter is defined before conditioning on the current state,

$$\theta^*(r) \coloneqq \inf_{\pi \in \Pi} \frac{\mathbb{E}[VE_s(r) + Q(r)\varepsilon(r) \mid Q(r)]}{\mathbb{E}[T(r) \mid Q(r)]}. \tag{31}$$

The action-selection objective conditions on both $S(r)$ and $Q(r)$, whereas Equation (31) conditions only on $Q(r)$. Appendix J explains the relationship between these two levels of conditioning.

We allow the ratio subproblem to be solved approximately. A C-additive solution has a conditional ratio no more than C above the infimum in each frame. Appendix K states this condition explicitly and defines the i.i.d. comparison policies.

D.2 Markov-state drift and ratio conditioning

Renewable generation often changes gradually across decision boundaries. Solar irradiance, for example, can retain dependence between successive frames (Miao et al., 2018). The Markov model in Section 4.2 keeps this state dependence in the conditional drift.

The queue update remains Equation (6), but since the current state and queue need not be independent, we use the joint conditional drift

$$\Delta(Q(r), S(r)) \coloneqq \mathbb{E}[L(Q(r+1)) - L(Q(r)) \mid Q(r), S(r)]. \tag{32}$$

Theorem D.2 (Quadratic drift with Markov states)**.** *The drift obeys*

$$\Delta(Q(r), S(r)) \le \mathcal{U} + Q(r)\mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r), S(r)], \tag{33}$$

where

$$\mathcal{U} \ge \frac{1}{2}\sup_{\pi \in \Pi,\ s \in \mathcal{X}} \mathbb{E}[(\varepsilon(r) - \sigma T(r))^2 \mid \pi, S(r) = s] < \infty. \tag{34}$$

Proof. Squaring Equation (6) and conditioning on $(Q(r), S(r))$ gives the expansion used for Theorem D.1. Assumption 4.1(ii) bounds the second-moment term uniformly. □

The drift bound has the same algebraic form as in the independent-state setting, but its expectations retain the observed state. This difference matters for the comparison argument because a policy influences both the current costs and, through P_π, the distribution of subsequent states. The analysis therefore considers the state together with the backlog and uses the stationary distributions of the induced Markov chains.

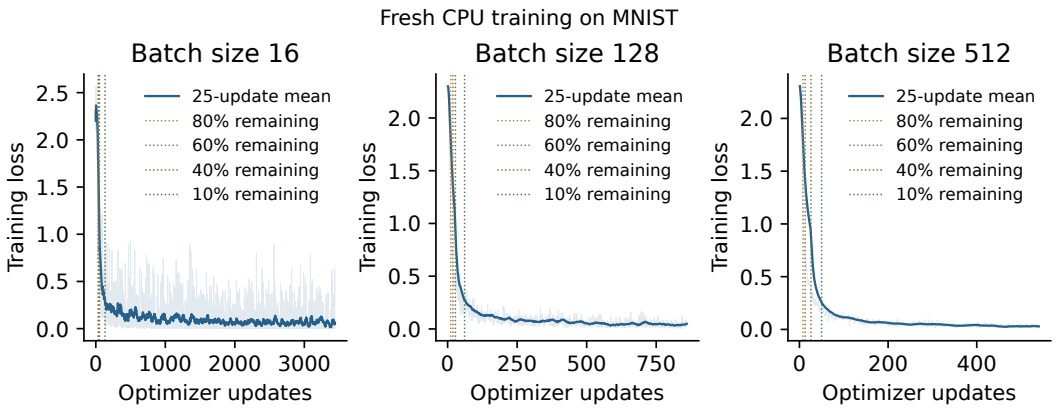


Figure 4: CNN training on MNIST with three batch sizes. Curves show raw loss and its trailing 25-update mean. Markers use the smoothed loss to locate fractions of the observed initial-to-final reduction that remain.

Adding the shortfall penalty gives

$$\begin{aligned}&\Delta(Q(r),S(r))+V\mathbb{E}[E_s(r)\mid Q(r),S(r)]\\&\quad\le \mathcal{U}+V\mathbb{E}[E_s(r)\mid Q(r),S(r)]+Q(r)\mathbb{E}[\varepsilon(r)\mid Q(r),S(r)]\\&\qquad-\sigma Q(r)\mathbb{E}[T(r)\mid Q(r),S(r)].\end{aligned}\tag{35}$$

For the observed pair $(S(r),Q(r))$, define

$$\theta^*(r)\coloneqq\inf_{\pi\in\Pi_{S(r)}}\frac{\mathbb{E}[VE_s(r)+Q(r)\varepsilon(r)\mid S(r),Q(r)]}{\mathbb{E}[T(r)\mid S(r),Q(r)]}.\tag{36}$$

The controller minimizes the corresponding additive expression in Equation (8), with $\psi'(Q)=Q$, and then updates the queue. Both the ratio and the action choice now condition on the current state.

D.3 Measured motivation for the phase state

The CNN–MNIST example in Figure 4 uses batch sizes 16, 128, and 512 and marks the points at which 80%, 60%, 40%, and 10% of the observed initial-to-final loss reduction remain. These measured trajectories illustrate an early period of rapid reduction followed by slower progress, which motivates the phase component of the state. Appendix P.3 defines the normalized loss used to locate these levels and gives the training configuration.

With the phase added, the base state retains model characteristics and renewable availability, and the extended transition model also describes training randomness and the phase-update rule under the chosen action.

D.4 Phase-weighted drift expansion

The phase-weighted penalty in Equation (15) permits different backlog weights in descent and plateau regimes. For example, $w_1\ge w_2$ reflects a model in which training moves from higher-loss states toward a low-loss plateau.

With the current phase held fixed in the one-step drift, squaring Equation (6) gives

$$\begin{aligned}&\Delta(Q(r),\widetilde{S}(r))\\&\quad\coloneqq\mathbb{E}\Big[L(Q(r+1),\phi(r))-L(Q(r),\phi(r))\mid Q(r),\widetilde{S}(r)\Big]\\&\quad\le\frac{w_{\phi(r)}}{2}\mathbb{E}[(\varepsilon(r)-\sigma T(r))^2\mid Q(r),\widetilde{S}(r)]\\&\qquad+w_{\phi(r)}Q(r)\mathbb{E}[\varepsilon(r)-\sigma T(r)\mid Q(r),\widetilde{S}(r)].\end{aligned}\tag{37}$$

A common bound for both phases is

$$\mathcal{U}_{\max} \coloneqq \frac{1}{2} \max_{k\in\{1,2\}} \sup_{\pi\in\Pi,\ s\in\mathcal{X}} w_k \mathbb{E}[(\varepsilon(r) - \sigma T(r))^2 \mid \pi, S = s] < \infty. \tag{38}$$

Hence

$$\Delta(Q(r), \widetilde{S}(r)) \le \mathcal{U}_{\max} + w_{\phi(r)} Q(r) \mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r), \widetilde{S}(r)]. \tag{39}$$

After adding the shortfall penalty, the drift-plus-penalty bound becomes

$$\begin{aligned} &\Delta(Q(r), \widetilde{S}(r)) + V\mathbb{E}[E_s(r) \mid Q(r), \widetilde{S}(r)] \\ &\quad \le \mathcal{U}_{\max} + V\mathbb{E}[E_s(r) \mid Q(r), \widetilde{S}(r)] \\ &\qquad + w_{\phi(r)} Q(r) \mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r), \widetilde{S}(r)]. \end{aligned} \tag{40}$$

The resulting action and ratio are Equations (16) and (17). The controller follows the steps of Algorithm 1, with coefficient $w_{\phi(r)}Q(r)$.

D.5 Interpreting recurrent training classes

Assume that, under a fixed phase-aware policy, the extended process is finite-state, time-homogeneous, and aperiodic. We do not require irreducibility over its entire state space, so there can be transient states and multiple closed communicating classes. From any initial state, a finite-state trajectory enters one such recurrent class $\mathcal{C}$ in finite time almost surely and then remains there. Within a class, the induced irreducible, aperiodic chain has a stationary distribution $\eta^{\pi}_{\mathcal{C}}$. Conditional on entering that class, the Markov law of large numbers gives the long-run average of a bounded quantity $f(\widetilde{S}, \pi(\widetilde{S}))$ as its expectation under $\eta^{\pi}_{\mathcal{C}}$. Theorem 5.1 uses this class-conditional interpretation, and Appendix M proves it.

The class restriction reflects the training dynamics represented by the model. If a discretized loss process can enter a low-loss region but cannot return to higher-loss regions, the high-loss states are transient and the low-loss states form closed classes. Such a process is not irreducible over the full state space. When several recurrent classes are possible, its long-run behavior must be described using the class that is reached and the stationary distribution of that class. Once the trajectory enters $\mathcal{C}$, it stays there, so the class-specific expectations determine the asymptotic metrics. A common recurrent class for all starting states removes the need for a conditional statement.

E Notation

Table 2 collects the quantities used in the formulation and analysis. A frame is one decision segment, which may contain an epoch or a fixed number of optimizer updates.

Table 2: Notation used in the paper.

Symbol	Meaning
TARS	Time-averaged renewable shortfall.
DWAE	Duration-weighted average error.
TARSM	Time-averaged renewable shortfall minimization.
$\mathcal{E} = \{0, 1, \ldots\}$	The sequence of training frames.
R	Number of completed frames.
r	Frame index, starting from zero.
$S(r) \in \mathcal{X}$	State observed at the beginning of frame r, including the model and its configuration.
Π	Finite collection of admissible scheduling policies.
$\pi(r) \in \Pi$	Policy applied in frame r.
M	Model parameters.
$J(M)$	Training objective evaluated at M.
$J(M^*)$	Minimum objective value, attained by M^*.

Symbol	Meaning
$e_r = J(M_r) - J(M^*)$	Objective gap at the end of frame r.
$T(r)$	Elapsed time for frame r.
$\widehat{T}(S(r), \pi(r))$	Model of frame duration for a given state and policy.
$E^{\mathrm{dem}}(r)$	Energy required by training in frame r.
$E^{\mathrm{ren}}(r)$	Renewable energy produced during frame r.
$E_s(r)$	Renewable shortfall, $\max\{0, E^{\mathrm{dem}}(r) - E^{\mathrm{ren}}(r)\}$.
$\widehat{E}_s(S(r), \pi(r))$	Shortfall model indexed by state and policy.
$\overline{P}_R$	TARS, $\sum_{r=0}^{R-1} E_s(r) / \sum_{r=0}^{R-1} T(r)$.
$\varepsilon(r)$	Endpoint error weighted by frame duration, $T(r)e_r$.
$\overline{e}_R$	DWAE, $\sum_{r=0}^{R-1} \varepsilon(r) / \sum_{r=0}^{R-1} T(r)$.
$T_{\min}, T_{\max}$	Lower and upper limits on the duration of a frame.
$E_{\min}, E_{\max}$	Lower and upper limits on the shortfall in a frame.
$e^2_{\max}$	Bound on the squared objective gap e_r^2.
E_{tot}	Total training demand, $\sum_{r=0}^{R-1} E^{\mathrm{dem}}(r)$.
$E_{\mathrm{ren,tot}}$	Total renewable production, $\sum_{r=0}^{R-1} E^{\mathrm{ren}}(r)$.
T_{tot}	Total elapsed time, $\sum_{r=0}^{R-1} T(r)$.
$\overline{T}(R)$	Expected duration averaged across frames, $R^{-1} \sum_{r=0}^{R-1} \mathbb{E}[T(r)]$.
$\overline{E}_s(R)$	Expected shortfall averaged across frames, $R^{-1} \sum_{r=0}^{R-1} \mathbb{E}[E_s(r)]$.
$\overline{\varepsilon}(R)$	Expected duration-weighted error averaged across frames, $R^{-1} \sum_{r=0}^{R-1} \mathbb{E}[\varepsilon(r)]$.
σ	Prescribed upper limit on DWAE.
$Q(r)$	Virtual backlog that tracks the error budget.
$L(Q(r))$	Lyapunov function, also denoted by $\psi(Q(r))$.
$\Delta(Q(r))$	Expected one-frame change in the Lyapunov function, conditional on the current backlog.

F How duration-weighted error relates to convergence

DWAE weights each endpoint objective gap by the duration of its frame. This section relates it to frame count, elapsed time, and gradient-step count. We write $\epsilon > 0$ for the target accuracy, reserving $\varepsilon(r)$ for the duration-weighted error. Recall that

$$\overline{e}_R = \frac{\sum_{r=0}^{R-1} \varepsilon(r)}{\sum_{r=0}^{R-1} T(r)} = \frac{\sum_{r=0}^{R-1} T(r) e_r}{\sum_{r=0}^{R-1} T(r)}, \qquad e_r = J(M_r) - J(M^*). \tag{41}$$

F.1 Frame count

Define the number of frames needed to first reach the accuracy target by

$$R_\epsilon = \min\{R \in \mathbb{N} : \ e_r < \epsilon \text{ for some } r < R\}. \tag{42}$$

This count describes progress in the objective but assigns the same cost to every frame, whereas DWAE also accounts for duration. DWAE can decrease when the objective gap falls more quickly or when frames with a large gap take less time, so minimizing it favors early progress and discourages long frames at high error.

F.2 Elapsed time

Let $r^* = \min\{r : e_r < \epsilon\}$ be the first frame that reaches the target. The time to convergence is

$$T_{\mathrm{conv}}(\epsilon) = \sum_{r=0}^{r^*} T(r). \tag{43}$$

For $R > r^*$, the numerator of DWAE satisfies

$$\sum_{r=0}^{R-1} T(r) e_r \geq \sum_{r=0}^{r^*} T(r) e_r > \epsilon \sum_{r=0}^{r^*} T(r) = \epsilon T_{\mathrm{conv}}(\epsilon). \tag{44}$$

Convergence measure	Connection to DWAE
Frames to reach a target	DWAE favors a rapid decrease in error and short frames while error remains high.
Time to reach a target	The comparison in Equation (45) bounds convergence time relative to the total duration.
Steps to reach a target	Weighting by step duration favors early steps that reduce error at a low time cost.

Table 3: Interpretation of duration-weighted error in terms of common convergence measures.

The comparison uses the pre-convergence lower bound $e_r \geq \epsilon$. Dividing by the total training duration gives

$$\overline{e}_R > \frac{\epsilon T_{\mathrm{conv}}(\epsilon)}{\sum_{r=0}^{R-1} T(r)}, \qquad T_{\mathrm{conv}}(\epsilon) < \frac{\overline{e}_R}{\epsilon} \sum_{r=0}^{R-1} T(r). \tag{45}$$

Thus, a low DWAE indicates a short time to reach the target relative to the full training horizon. If $\overline{e}_R = o(1)$ as $R \to \infty$, Equation (45) shows that the convergence time is negligible relative to the total elapsed time.

F.3 Gradient-step count

Write I_r for the number of gradient steps in frame r, and let $I_R = \sum_{r=0}^{R-1} I_r$ be the cumulative count. Suppose the duration is proportional to $I_r \tau_r$, where the mean time per step τ_r depends on both batch size and hardware. Then

$$\overline{e}_R = \frac{\sum_{r=0}^{R-1} T(r) e_r}{\sum_{r=0}^{R-1} T(r)} = \frac{\sum_{r=0}^{R-1} I_r \tau_r e_r}{\sum_{r=0}^{R-1} I_r \tau_r}. \tag{46}$$

The contribution of a frame now depends on both its step count and the cost of each step, so slow steps at a high objective gap receive more weight. The resulting metric favors useful early updates while accounting for their computational cost.

These relationships show that DWAE jointly measures optimization progress and runtime. In training with limited energy or compute, it can therefore be used to compare policies whose frame and step durations differ. Table 3 summarizes the interpretations above.

G Proof of Proposition C.1

Proof. It suffices to consider frame duration, because the same argument applies to shortfall and duration-weighted error. For a feasible policy, write

$$\widehat{T}(R) = \frac{1}{R} \sum_{r=0}^{R-1} T(r), \qquad \overline{T}(R) = \frac{1}{R} \sum_{r=0}^{R-1} \mathbb{E}[T(r)]. \tag{47}$$

Assumption 2.1 gives $T_{\min} \leq T(r) \leq T_{\max}$ almost surely for every r. The argument uses a finite steady-state limit $\widehat{T}$ such that

$$\widehat{T}(R) \longrightarrow \widehat{T} \quad \text{almost surely.} \tag{48}$$

For any $\epsilon > 0$, split the expectation according to whether $\widehat{T}(R)$ exceeds $\widehat{T} + \epsilon$:

$$\mathbb{E}[\widehat{T}(R)] \leq (\widehat{T} + \epsilon) \Pr\{\widehat{T}(R) \leq \widehat{T} + \epsilon\} + T_{\max} \Pr\{\widehat{T}(R) > \widehat{T} + \epsilon\}. \tag{49}$$

Almost-sure convergence implies

$$\Pr\{\widehat{T}(R) > \widehat{T} + \epsilon\} \longrightarrow 0. \tag{50}$$

As a result,

$$\limsup_{R \to \infty} \overline{T}(R) \leq \widehat{T} + \epsilon. \tag{51}$$

Algorithm 2 Bisection estimate of $\theta^*(r)$

Require: Initial bounds $\theta_{\min}, \theta_{\max}$, iteration budget $K \geq 1$, function $\mathrm{val}(\cdot)$, positive expansion steps $\Delta_{\min}, \Delta_{\max}$
Ensure: Estimate θ

while $\mathrm{val}(\theta_{\min}) < 0$ **do**
  $\theta_{\min} \leftarrow \theta_{\min} - \Delta_{\min}$
end while
while $\mathrm{val}(\theta_{\max}) > 0$ **do**
  $\theta_{\max} \leftarrow \theta_{\max} + \Delta_{\max}$
end while
for $k = 1, \ldots, K$ **do**
  $\theta \leftarrow (\theta_{\min} + \theta_{\max})/2$
  if $\mathrm{val}(\theta) = 0$ **then**
    return θ
  else if $\mathrm{val}(\theta) > 0$ **then**
    $\theta_{\min} \leftarrow \theta$
  else
    $\theta_{\max} \leftarrow \theta$
  end if
end for
return θ

Taking $\epsilon \downarrow 0$ proves $\limsup_R \overline{T}(R) \leq \widehat{T}$. Splitting at $\widehat{T} - \epsilon$ instead and applying the lower bound on duration proves the reverse inequality,

$$\liminf_{R\to\infty} \overline{T}(R) \geq \widehat{T}. \tag{52}$$

Consequently, $\overline{T}(R) \to \widehat{T}$, and together with Equation (48) this yields

$$\overline{T}(R) - \widehat{T}(R) \longrightarrow 0 \quad \text{almost surely.} \tag{53}$$

The expected and empirical averages therefore have the same asymptotic value. Applying the same reasoning to $\widehat{E}_s(R)$ and $\widehat{\varepsilon}(R)$ establishes their equivalence to $\overline{E}_s(R)$ and $\overline{\varepsilon}(R)$. Thus, replacing these expected averages with empirical averages in the limiting optimization problem preserves its objective value and optimal policies. □

H Bisection for the ratio parameter

The ratio parameter is the zero of a scalar function, which for W stochastic samples is

$$\mathrm{val}(\theta) = \frac{1}{W} \sum_{i=1}^{W} \min_{\pi(r)\in\Pi} \{VE_s(r) + Q(r)\varepsilon(r) - \theta T(r)\}. \tag{54}$$

The expression inside the sum is evaluated for sample i. Here $E_s(r)$, $\varepsilon(r)$, and $T(r)$ are the sampled frame quantities, $Q(r)$ is the current backlog, and $V > 0$ sets the energy weight.

A positive $\mathrm{val}(\theta)$ indicates that θ lies below the attainable ratio, while a negative value indicates that θ lies above it, so the zero of val corresponds to $\theta^*(r)$. Algorithm 2 first extends the initial interval by the positive increments $\Delta_{\min}$ and $\Delta_{\max}$ until val has opposite signs at its endpoints. It then halves this interval, stopping after K steps or when it finds an exact zero. Under the usual continuity condition on val, this procedure converges to the ratio parameter.

I Proof of Theorem D.1

Proof. Squaring the update in Equation (6) and using $\max\{x, 0\}^2 \leq x^2$ gives

$$Q(r+1)^2 \leq \big(Q(r) + \varepsilon(r) - \sigma T(r)\big)^2 \tag{55}$$

$$= Q(r)^2 + \big(\varepsilon(r) - \sigma T(r)\big)^2 + 2Q(r)\big(\varepsilon(r) - \sigma T(r)\big). \tag{56}$$

Subtract $Q(r)^2$ and divide by two to obtain

$$\frac{Q(r+1)^2 - Q(r)^2}{2} \leq \frac{\big(\varepsilon(r) - \sigma T(r)\big)^2}{2} + Q(r)\big(\varepsilon(r) - \sigma T(r)\big). \tag{57}$$

Conditioning on $Q(r)$ proves the drift bound. Assumption 2.1 makes its quadratic remainder $\mathcal{U}$ finite. □

J THE RATIO PROBLEM AFTER OBSERVING THE STATE

Theorem J.1 (Conditioned ratio minimization). *Let $\pi^* \in \Pi_{S(r)}$ minimize*

$$\mathbb{E}[VE_s(r) + Q(r)\varepsilon(r) - \theta^*(r)T(r) \mid S(r), Q(r)]\,. \tag{58}$$

Then the resulting policy satisfies

$$\mathbb{E}[VE_s(r) + Q(r)\varepsilon(r) - \theta^*(r)T(r) \mid Q(r)] = 0 \tag{59}$$

and attains the minimum drift-plus-penalty ratio $\theta^*(r)$.

Proof. The fractional-programming characterization of Dinkelbach (Dinkelbach, 1967) states that the optimal conditional ratio satisfies

$$\inf_{\pi\in\Pi} \mathbb{E}[VE_s(r) + Q(r)\varepsilon(r) - \theta^*(r)T(r) \mid Q(r)] = 0. \tag{60}$$

For compactness, denote the random expression inside this expectation by Z_π. Iterated conditioning gives

$$\inf_{\pi\in\Pi} \mathbb{E}[Z_\pi \mid Q(r)] = \inf_{\pi\in\Pi} \mathbb{E}[\mathbb{E}[Z_\pi \mid S(r), Q(r)] \mid Q(r)]\,. \tag{61}$$

After observing $S(r)$, a policy chooses a state-specific decision $\pi' \in \Pi_{S(r)}$. The minimization can therefore be carried out for each observed state:

$$\inf_{\pi\in\Pi} \mathbb{E}[Z_\pi \mid Q(r)] = \mathbb{E}\left[\inf_{\pi'\in\Pi_{S(r)}} \mathbb{E}[Z_{\pi'} \mid S(r), Q(r)] \mid Q(r)\right]. \tag{62}$$

A minimizer of the inner conditional expectation achieves the zero on the left, which establishes the result. □

K PROOF OF THEOREM 4.1

We make the policy dependence explicit by writing $\widehat{T}(\pi(r))$, $\widehat{E}_s(\pi(r))$, and $\widehat{\varepsilon}(\pi(r))$. The comparison below uses feasible i.i.d. policies and assumes that an optimal solution is available in this setting.

Definition K.1 (i.i.d. policy). An i.i.d. policy draws a control decision from the same fixed distribution in every frame. Its limiting averages equal the corresponding one-frame expectations. Feasibility requires

$$\frac{\mathbb{E}[\widehat{\varepsilon}(\pi^*(r))]}{\mathbb{E}[\widehat{T}(\pi^*(r))]} \leq \sigma. \tag{63}$$

Write ζ^* for the infimum of $\mathbb{E}[\widehat{E}_s(\pi^*(r))]/\mathbb{E}[\widehat{T}(\pi^*(r))]$ over feasible i.i.d. policies.

Lemma K.1 (Feasible comparison policy). *If an i.i.d. feasible policy exists, then for each $\delta > 0$ there is an i.i.d. feasible policy $\pi^*(r)$ such that*

$$\mathbb{E}[\widehat{E}_s(\pi^*(r))] \leq \mathbb{E}[\widehat{T}(\pi^*(r))](\zeta^* + \delta), \tag{64}$$

$$\mathbb{E}[\widehat{\varepsilon}(\pi^*(r))] \leq \sigma\mathbb{E}[\widehat{T}(\pi^*(r))]. \tag{65}$$

The backlog bound uses strict feasibility in addition to this comparison.

Assumption K.1 (Slater condition for i.i.d. policies). There are $\gamma > 0$ and an i.i.d. policy $\pi^\gamma(r)$ for which

$$\mathbb{E}[\widehat{\varepsilon}(\pi^\gamma(r))] \leq (\sigma - \gamma)\mathbb{E}[\widehat{T}(\pi^\gamma(r))]. \tag{66}$$

Definition K.2 (C-additive ratio solver)**.** A policy $\pi(r)$ gives a C-additive approximation if, for every backlog $Q(r)$,

$$\frac{\mathbb{E}[V\widehat{E}_s(\pi(r)) + Q(r)\widehat{\varepsilon}(\pi(r)) \mid Q(r)]}{\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)]} \leq C + \inf_{\pi\in\Pi} \frac{\mathbb{E}[V\widehat{E}_s(\pi) + Q(r)\widehat{\varepsilon}(\pi) \mid Q(r)]}{\mathbb{E}[\widehat{T}(\pi) \mid Q(r)]}. \tag{67}$$

Proof of Theorem 4.1. Multiply Equation (67) by its denominator and compare with an i.i.d. policy $\pi^*(r)$. Since $T(r) \leq T_{\max}$, this gives

$$\mathbb{E}[V\widehat{E}_s(\pi(r)) + Q(r)\widehat{\varepsilon}(\pi(r)) \mid Q(r)] \leq CT_{\max} + \mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)] \frac{\mathbb{E}[V\widehat{E}_s(\pi^*(r)) + Q(r)\widehat{\varepsilon}(\pi^*(r))]}{\mathbb{E}[\widehat{T}(\pi^*(r))]}. \tag{68}$$

In the comparison expectation, $Q(r)$ is held at its current value. Combining this inequality with Theorem D.1 yields

$$\begin{aligned}\Delta(Q(r)) + V\mathbb{E}[E_s(r) \mid Q(r)] \leq \mathcal{U} + CT_{\max} &+ \frac{\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)]}{\mathbb{E}[\widehat{T}(\pi^*(r))]} V\mathbb{E}[\widehat{E}_s(\pi^*(r))] \\ &+ \frac{\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)]}{\mathbb{E}[\widehat{T}(\pi^*(r))]} Q(r)\mathbb{E}[\widehat{\varepsilon}(\pi^*(r))] \\ &- \sigma Q(r)\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)].\end{aligned} \tag{69}$$

For any $\delta > 0$, substituting both bounds of Lemma K.1 gives

$$\Delta(Q(r)) + V\mathbb{E}[E_s(r) \mid Q(r)] \leq \mathcal{U} + CT_{\max} + V\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)](\zeta^* + \delta). \tag{70}$$

Letting $\delta \downarrow 0$ gives the central estimate

$$\Delta(Q(r)) + V\mathbb{E}[E_s(r) \mid Q(r)] \leq \mathcal{U} + CT_{\max} + V\zeta^*\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)]. \tag{71}$$

Assumption 2.1 then implies

$$\Delta(Q(r)) \leq \mathcal{U} + CT_{\max} + V\max\{\zeta^*T_{\max}, \zeta^*T_{\min}\} - VE_{\min}. \tag{72}$$

Summing the drift over frames establishes mean-rate stability, $\mathbb{E}[Q(R)]/R \to 0$. Applying Theorem 2.5(b) in Neely (2010) to the virtual queue then gives

$$\limsup_{R\to\infty} \big(\overline{\varepsilon}(R) - \sigma\overline{T}(R)\big) \leq 0. \tag{73}$$

To convert this into a ratio bound, note that

$$\begin{aligned}\frac{\overline{\varepsilon}(R)}{\overline{T}(R)} - \sigma &\leq \frac{\overline{T}(R)}{T_{\min}} \max\left\{0, \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} - \sigma\right\} \\ &= \frac{1}{T_{\min}} \max\{0, \overline{\varepsilon}(R) - \sigma\overline{T}(R)\}.\end{aligned} \tag{74}$$

Equation (73) therefore proves

$$\limsup_{R\to\infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \leq \sigma. \tag{75}$$

Taking an unconditional expectation in Equation (71) gives

$$\mathbb{E}[L(Q(r+1))] - \mathbb{E}[L(Q(r))] + V\mathbb{E}[E_s(r)] \leq \mathcal{U} + CT_{\max} + V\zeta^*\mathbb{E}[T(r)]. \tag{76}$$

Sum over $r = 0, \ldots, R-1$ and divide by RV:

$$\frac{\mathbb{E}[L(Q(R))] - \mathbb{E}[L(Q(0))]}{RV} + \frac{1}{R}\sum_{r=0}^{R-1} \mathbb{E}[E_s(r)] \leq \frac{\mathcal{U} + CT_{\max}}{V} + \frac{\zeta^*}{R}\sum_{r=0}^{R-1} \mathbb{E}[T(r)]. \tag{77}$$

Because $Q(0) = 0$ and $L \geq 0$,

$$\overline{E}_s(R) \leq \frac{\mathcal{U} + CT_{\max}}{V} + \zeta^* \overline{T}(R). \tag{78}$$

For the backlog bound, substituting the strictly feasible policy from Assumption K.1 into Equation (69) gives

$$\Delta(Q(r)) + V\mathbb{E}[E_s(r) \mid Q(r)] \leq \mathcal{U} + CT_{\max} + V\frac{\mathbb{E}[\widehat{T}(\pi(r)) \mid Q(r)]}{\mathbb{E}[\widehat{T}(\pi^\gamma(r))]} E_{\max} - \gamma T_{\min} Q(r). \tag{79}$$

Define

$$F = \frac{\mathcal{U} + CT_{\max}}{V} + \max\left\{\frac{T_{\max}}{T_{\min}} E_{\max}, \frac{T_{\min}}{T_{\max}} E_{\max}\right\} - E_{\min}. \tag{80}$$

By Assumption 2.1, the drift then satisfies

$$\Delta(Q(r)) \leq VF - \gamma T_{\min} Q(r). \tag{81}$$

Summing over the first R frames and using $L(Q(0)) = 0$ proves

$$\frac{1}{R}\sum_{r=0}^{R-1} \mathbb{E}[Q(r)] \leq \frac{VF}{\gamma T_{\min}}. \tag{82}$$

This completes the cost, feasibility, and backlog estimates. □

L Proof of Theorem 4.2

We begin with the time-average equivalence used in Proposition C.1. Fix a stationary Markov policy π, for which Assumption 4.1 gives a unique stationary distribution η^π. The strong law for finite irreducible Markov chains yields

$$\frac{1}{R}\sum_{r=0}^{R-1} T(r) \xrightarrow{\text{a.s.}} \mathbb{E}_{\eta^\pi}[T(S, \pi(S))], \tag{83}$$

$$\frac{1}{R}\sum_{r=0}^{R-1} E_s(r) \xrightarrow{\text{a.s.}} \mathbb{E}_{\eta^\pi}[E_s(S, \pi(S))], \tag{84}$$

$$\frac{1}{R}\sum_{r=0}^{R-1} \varepsilon(r) \xrightarrow{\text{a.s.}} \mathbb{E}_{\eta^\pi}[\varepsilon(S, \pi(S))]. \tag{85}$$

Thus, empirical averages converge to their stationary expectations, and the ratios have the limits used in Proposition C.1. The performance argument follows the same comparison structure as the i.i.d. analysis.

Definition L.1 (Stationary Markov policy)**.** A stationary Markov policy assigns a distribution over Π to each state $s \in \mathcal{X}$. Under this policy, $\{S(r)\}$ is a finite, time-homogeneous Markov chain with stationary distribution η^π.

Assumption L.1 (Slater condition for Markov policies)**.** There exist $\gamma > 0$ and a stationary Markov policy π^γ satisfying

$$\mathbb{E}_{\eta^{\pi^\gamma}}[\varepsilon(S, \pi^\gamma(S))] \leq (\sigma - \gamma)\mathbb{E}_{\eta^{\pi^\gamma}}[T(S, \pi^\gamma(S))]. \tag{86}$$

Definition L.2 (C-additive solver conditioned on state and backlog)**.** The selected decision is a C-additive approximation when

$$\frac{\mathbb{E}[VE_s(r) + Q(r)\varepsilon(r) \mid S(r), Q(r)]}{\mathbb{E}[T(r) \mid S(r), Q(r)]} \leq C + \inf_{\pi \in \Pi_{S(r)}} \frac{\mathbb{E}_\pi[VE_s(r) + Q(r)\varepsilon(r) \mid S(r), Q(r)]}{\mathbb{E}_\pi[T(r) \mid S(r), Q(r)]}. \tag{87}$$

The subscript π on a conditional expectation indicates the candidate decision used for that comparison.

Assuming a stationary comparison policy exists (Puterman, 2014), let

$$\zeta^* = \inf_{\text{stationary Markov policies } \pi} \frac{\mathbb{E}_{\eta^\pi}[E_s(S, \pi(S))]}{\mathbb{E}_{\eta^\pi}[T(S, \pi(S))]}. \tag{88}$$

Proof of Theorem 4.2. With the queue initialized at zero, conditioning the bounded-drift comparison on $(S(r), Q(r))$ gives

$$\begin{aligned}
&\Delta(Q(r), S(r)) + V\mathbb{E}[E_s(r) \mid S(r), Q(r)] \\
&\quad \le \mathcal{U} + CT_{\max} + \frac{\mathbb{E}[T(r) \mid S(r), Q(r)]}{\mathbb{E}_{\eta^{\pi^*}}[T(r)]} V\mathbb{E}_{\eta^{\pi^*}}[E_s(r)] \\
&\qquad + \frac{\mathbb{E}[T(r) \mid S(r), Q(r)]}{\mathbb{E}_{\eta^{\pi^*}}[T(r)]} Q(r)\mathbb{E}_{\eta^{\pi^*}}[\varepsilon(r)] - \sigma Q(r)\mathbb{E}[T(r) \mid S(r), Q(r)].
\end{aligned} \tag{89}$$

For any $\delta > 0$, take a feasible comparison policy with

$$\mathbb{E}_{\eta^{\pi^*}}[E_s(r)] \le (\zeta^* + \delta)\mathbb{E}_{\eta^{\pi^*}}[T(r)], \tag{90}$$

$$\mathbb{E}_{\eta^{\pi^*}}[\varepsilon(r)] \le \sigma\mathbb{E}_{\eta^{\pi^*}}[T(r)]. \tag{91}$$

Substituting these inequalities and taking $\delta \downarrow 0$ yields

$$\begin{aligned}
&\Delta(Q(r), S(r)) + V\mathbb{E}[E_s(r) \mid S(r), Q(r)] \\
&\quad \le \mathcal{U} + CT_{\max} + V\zeta^*\mathbb{E}[T(r) \mid S(r), Q(r)].
\end{aligned} \tag{92}$$

Taking unconditional expectations and summing over $r = 0, \dots, R-1$ gives

$$\begin{aligned}
&\frac{\mathbb{E}[L(Q(R))] - \mathbb{E}[L(Q(0))]}{R} + \frac{V}{R}\sum_{r=0}^{R-1}\mathbb{E}[E_s(r)] \\
&\quad \le \mathcal{U} + CT_{\max} + \frac{V\zeta^*}{R}\sum_{r=0}^{R-1}\mathbb{E}[T(r)].
\end{aligned} \tag{93}$$

With the steady-state interpretation under Assumption 4.1, the resulting bound is

$$\overline{E}_s(R) \le \frac{\mathcal{U} + CT_{\max}}{V} + \zeta^*\overline{T}(R) + o(1). \tag{94}$$

For the finite-horizon expected averages, the displayed telescoping inequality already gives the bound without the $o(1)$ term, which records the vanishing difference introduced when these averages are replaced by their steady-state expectations under mixing.

To bound the backlog, repeating the comparison with the policy π^γ from Assumption L.1 gives

$$\Delta(Q(r), S(r)) \le VF - \gamma T_{\min} Q(r), \tag{95}$$

where F is finite, as in Equation (80). Summing the drift gives

$$\frac{1}{R}\sum_{r=0}^{R-1}\mathbb{E}[Q(r)] \le \frac{VF}{\gamma T_{\min}}. \tag{96}$$

Mean-rate stability and the queue update then imply

$$\limsup_{R\to\infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \le \sigma, \tag{97}$$

which proves the stated constraint and cost guarantees. □

M Proof of Theorem 5.1

The comparison used in Appendices K and L can also be expressed within a recurrent class of the extended state process.

Definition M.1 (Stationary policy using the training phase)**.** A stationary phase-aware Markov policy maps each extended state $\langle s, k\rangle \in \mathcal{X} \times \{1, 2\}$ to a distribution over Π. The resulting chain $\{\widetilde{S}(r)\}$ is finite and time-homogeneous, but it need not be irreducible. Its states consist of transient states and one or more closed communicating classes. Starting from a specified initial state, it reaches a recurrent class $\mathcal{C}$ in finite time with probability one and then remains in that class. Within $\mathcal{C}$, the chain is irreducible and aperiodic and has a unique stationary distribution $\eta^{\pi}_{\mathcal{C}}$.

Assumption M.1 (Slater condition within a recurrent class)**.** For the class $\mathcal{C}$, there are $\gamma > 0$ and a stationary phase-aware Markov policy $\pi^{\gamma}_{\mathcal{C}}$, supported on $\mathcal{C}$, for which

$$\mathbb{E}_{\eta^{\pi^{\gamma}_{\mathcal{C}}}_{\mathcal{C}}}[\varepsilon(\widetilde{S}, \pi^{\gamma}_{\mathcal{C}}(\widetilde{S}))] \le (\sigma - \gamma)\mathbb{E}_{\eta^{\pi^{\gamma}_{\mathcal{C}}}_{\mathcal{C}}}[T(\widetilde{S}, \pi^{\gamma}_{\mathcal{C}}(\widetilde{S}))]. \tag{98}$$

Definition M.2 (C-additive phase-aware decision within a class)**.** At extended state $\widetilde{S}(r) = (S(r), \phi(r))$ and backlog $Q(r)$, the selected decision has additive error at most C if

$$\begin{aligned}&\frac{\mathbb{E}[VE_s(r) + w_{\phi(r)}Q(r)\varepsilon(r) \mid \widetilde{S}(r), Q(r)]}{\mathbb{E}[T(r) \mid \widetilde{S}(r), Q(r)]}\\ &\quad\le C + \inf_{\pi \in \Pi_{S(r)}} \frac{\mathbb{E}_{\pi}[VE_s(r) + w_{\phi(r)}Q(r)\varepsilon(r) \mid \widetilde{S}(r), Q(r)]}{\mathbb{E}_{\pi}[T(r) \mid \widetilde{S}(r), Q(r)]}.\end{aligned} \tag{99}$$

The class-specific comparison ratio is

$$\zeta^{*}_{\mathcal{C}} = \inf_{\substack{\text{stationary phase-aware}\\ \text{Markov policies } \pi \text{ on } \mathcal{C}}} \frac{\mathbb{E}_{\eta^{\pi}_{\mathcal{C}}}[E_s(\widetilde{S}, \pi(\widetilde{S}))]}{\mathbb{E}_{\eta^{\pi}_{\mathcal{C}}}[T(\widetilde{S}, \pi(\widetilde{S}))]}. \tag{100}$$

Proof of Theorem 5.1. The phase-dependent drift-plus-penalty bound in Equation (40) is

$$\begin{aligned}&\Delta(Q(r), \widetilde{S}(r)) + V\mathbb{E}[E_s(r) \mid Q(r), \widetilde{S}(r)]\\ &\quad\le \mathcal{U}_{\max} + V\mathbb{E}[E_s(r) \mid Q(r), \widetilde{S}(r)]\\ &\qquad + w_{\phi(r)}Q(r)\mathbb{E}[\varepsilon(r) - \sigma T(r) \mid Q(r), \widetilde{S}(r)].\end{aligned} \tag{101}$$

Within $\mathcal{C}$, the C-additive comparison takes the form

$$\begin{aligned}&\mathbb{E}[VE_s(r) + w_{\phi(r)}Q(r)\varepsilon(r) \mid Q(r), \widetilde{S}(r)]\\ &\quad\le CT_{\max} + \mathbb{E}[T(r) \mid Q(r), \widetilde{S}(r)]\frac{\mathbb{E}_{\eta^{\pi^{*}_{\mathcal{C}}}_{\mathcal{C}}}[VE_s(r) + w_{\phi}Q(r)\varepsilon]}{\mathbb{E}_{\eta^{\pi^{*}_{\mathcal{C}}}_{\mathcal{C}}}[T(r)]}.\end{aligned} \tag{102}$$

Choose the comparison policy $\pi^{*}_{\mathcal{C}}$ to satisfy

$$\mathbb{E}_{\eta^{\pi^{*}_{\mathcal{C}}}_{\mathcal{C}}}[\varepsilon(r)] \le \sigma\mathbb{E}_{\eta^{\pi^{*}_{\mathcal{C}}}_{\mathcal{C}}}[T(r)]. \tag{103}$$

Applying the preceding drift comparison gives

$$\Delta(Q(r), \widetilde{S}(r)) + V\mathbb{E}[E_s(r)] \le \mathcal{U}_{\max} + CT_{\max} + V\zeta^{*}_{\mathcal{C}}\mathbb{E}[T(r)]. \tag{104}$$

Condition on the event that the trajectory enters $\mathcal{C}$. Summing the drift and using the class-specific Slater condition of Assumption M.1 establishes mean-rate stability of $Q(r)$. The unchanged virtual-queue update then yields

$$\limsup_{R\to\infty} \frac{\overline{\varepsilon}(R)}{\overline{T}(R)} \le \sigma \qquad \text{conditional on entering } \mathcal{C}. \tag{105}$$

Taking expectations in Equation (104), summing over frames, and dividing by RV gives

$$\overline{E}_s(R) \le \frac{\mathcal{U}_{\max} + CT_{\max}}{V} + \zeta^{*}_{\mathcal{C}}\overline{T}(R) + o(1). \tag{106}$$

The term $o(1)$ tends to zero as the horizon increases and the chain mixes within $\mathcal{C}$. As in the Markov case, it represents the difference between finite-horizon expected averages and their stationary counterparts.

The stated cost and constraint guarantees apply almost surely and in expectation, conditional on the recurrent class reached by the proposed policy. If all initial states reach the same class with probability one, the guarantees also hold without conditioning on a class. □

N PROGRESS AND SYSTEM-COST ESTIMATION

A training action affects both the next model state and the time and energy needed to reach it, and we estimate these effects separately. Shared-checkpoint measurements provide an action-dependent loss predictor, and a GPU pilot supplies the power estimates. During evaluation, the controller observes the measured validation loss after every decision frame.

N.1 PROGRESS ESTIMATES FROM SHARED CHECKPOINTS

A steps-to-target curve summarizes the updates required at each fixed batch size (Shallue et al., 2019). Because it does not specify how one additional block changes loss at an intermediate training state, we measure that transition directly. A frame contains $K = 20$ optimizer updates, and x_r denotes validation cross-entropy at its start, divided by the initial validation cross-entropy of the same run. The predictor $F_B(x_r)$ estimates the endpoint under batch size B. The controller scores candidates with this prediction and then updates its queue with the measured endpoint x_{r+1}.

We collect reference trajectories at batch sizes 16, 128, and 512, using three initialization seeds. At nine prescribed checkpoints along each trajectory, we copy the model, full Adam state, random-number state, and shuffled example stream. From this common state, each of six candidate batch sizes performs 20 updates. Validation before and after each branch uses the same 5,000 examples. The resulting 486 paired transitions cover early descent and later training, including observed loss increases.

We compare two positive transition families by leaving out an entire calibration seed. One fits a monotone interpolation in log-loss coordinates, and the other describes one block of motion toward a fitted level c_B,

$$F_B(x) = c_B + \operatorname{sgn}(x - c_B)\frac{|x - c_B|}{\left(1 + p_B k_B |x - c_B|^{p_B}\right)^{1/p_B}}, \tag{107}$$

with $k_B > 0$, $p_B \geq 0$, and $c_B \geq 0$. At $p_B = 0$, the fraction is $|x - c_B| \exp(-k_B)$. The model permits an increase when $x < c_B$, and the parameters minimize squared log-endpoint error without deleting branches whose loss increases. Cross-validation selects Equation (107), and Appendix P.8 gives its coefficients, the alternative fit, and independent prediction checks.

The independent check uses three further seeds and ten fixed or switching batch schedules. Across 3,000 observed blocks, one-block prediction RMSE is 0.0161 in normalized loss units. Recursively applying the predictor without new observations gives RMSE 0.0331. This distinction matters because F_B uses only the scalar loss, which can accompany different optimizer histories. Our policy evaluation executes GPU updates and observes their endpoints, so neither the reported learning curves nor target-crossing times are recursively generated by F_B.

N.2 MEASURED RUNTIME AND POWER

A short training pilot estimates the cost of each batch size by measuring the elapsed time of an epoch and sampling GPU power during training. We fit the epoch means with

$$Y(B) = a_Y B^{-k_Y} + c_Y, \qquad Y \in \{T, P\}. \tag{108}$$

The constant term captures overhead or static power. The exponent for power may be negative in this notation, allowing power to increase with batch size. Appendix P.4 gives the measurement protocol.

Predicted energy demand is $T(B)P(B)$, and renewable energy can be forecast over a frame or estimated from the power observed at its start. The latter gives

$$E_s(B) \approx T(B) \max\{P(B) - P_{\text{ren}}, 0\}. \tag{109}$$

The curves summarize the epoch pilot, whereas each frame in the scheduling experiments contains 20 updates. Its duration is measured directly, and power is taken from the observed configuration mean at the selected batch size. The fitted power curve is used for visualization, while the finite-action controller uses the measured lookup values. The pilot is specific to the workload and hardware on which it is measured.

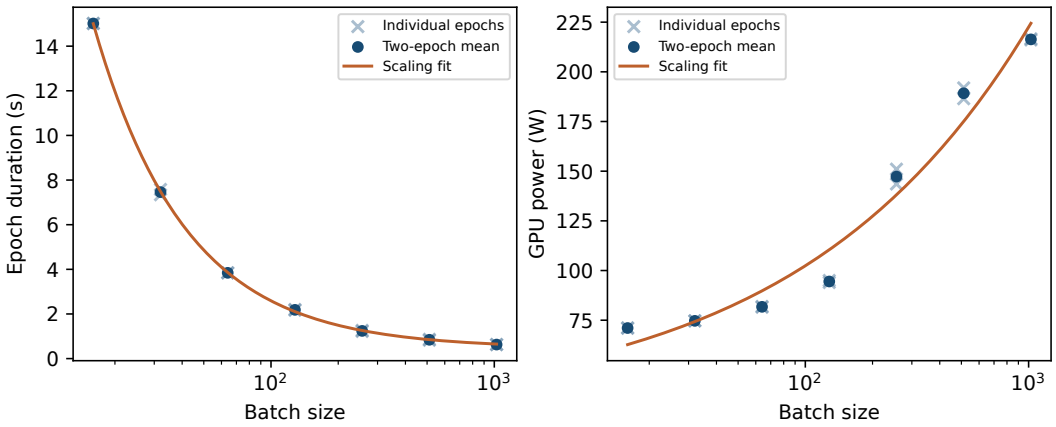


Figure 5: CNN runtime and power measured on an NVIDIA A800. Crosses show individual epochs, circles show the means of two consecutive epochs, and lines show least-squares fits. Power is the mean of NVML samples taken after each training update.

On an NVIDIA A800 with 80 GB of memory, the two-epoch pilot gives

$$T(B) = 267.3696B^{-1.0502} + 0.4672, \tag{110}$$

$$P(B) = 10.7816B^{0.4185} + 28.3052. \tag{111}$$

Time is measured in seconds and power in watts. Figure 5 shows all epoch observations, their configuration means, and the fitted curves. Across batch sizes 16 to 1,024, mean epoch time falls from 15.016 to 0.629 seconds, while sampled GPU power rises from 71.06 to 216.33 watts, so a shorter epoch can carry a higher average power cost. Over the seven configuration means, the runtime fit has root-mean-square error 0.0344 seconds and $R^2 = 0.99995$, while the power fit has error 10.28 watts and $R^2 = 0.9651$.

N.3 Estimates from transformer scaling laws

When a training pilot is too expensive, published scaling relationships provide another way to estimate system costs. For an autoregressive transformer with P_{model} trainable parameters and D_{tokens} training tokens (Kaplan et al., 2020), the approximate computation is

$$F_{\text{train}} \approx 6P_{\text{model}}D_{\text{tokens}}. \tag{112}$$

If sustained aggregate throughput is H FLOPs per second, runtime and energy can be estimated as

$$t_{\text{hours}} = \frac{F_{\text{train}}}{3600H}, \tag{113}$$

$$E_{\text{kWh}} = \frac{t_{\text{hours}}N_{\text{proc}}P_{\text{proc}}\,\text{PUE}}{1000}. \tag{114}$$

Here N_{proc} is the processor count, P_{proc} is average power per processor in watts, and PUE accounts for facility overhead. The estimates provide time and energy inputs to the controller when representative throughput and power values are available.

O Extended experimental evaluation

We study how batch scheduling balances renewable shortfall and training progress, how phase weights change time to a quality target, and how the position of a run within a PV cycle affects its energy cost. The evaluation combines CNN training on an NVIDIA A800 with renewable-supply scenarios. The BERT pilot in Figure 3 provides a separate measurement of batch-dependent system cost. Appendix P documents the workloads, inputs, and full protocols.

O.1 Workload, policies, and metrics

The CNN uses 55,000 MNIST training examples and 5,000 validation examples, with candidate batch sizes $\{16, 32, 64, 128, 256, 512\}$. Every run has 100 frames of 20 Adam updates, with no early termination. It starts in the rapid-descent phase, with normalized error one and an empty queue. We use error budget $\sigma = 0.05$, phase threshold $\tau = 0.1$, and default weights $(w_1, w_2) = (100, 1)$. After each frame, the measured relative error change determines the phase used by the next decision.

LYA chooses its batch size using the current power, validation loss, phase, and queue. B16, B128, and B512 hold batch size fixed, while STA uses batch 16 for the first 50 frames and batch 512 afterward. INC visits the candidate set in ascending order, with index $\min\{\lfloor 6r/100 \rfloor, 5\}$ for zero-based frame r, and DEC reverses that order. A shuffled training stream continues across frame boundaries, and model and Adam states are retained when batch size changes.

The calibrated reference constants are $T_{\mathrm{ref}} = 0.09693$ seconds and $P_{\mathrm{ref}} = 94.483$ watts, the block duration and power at batch 128. The normalized queue is $q = Q/T_{\mathrm{ref}}$, and the finite-action score is Equation (21) of the main text, which is the ratio objective after a change of units. The dimensionless control parameter is $\bar{V} = VP_{\mathrm{ref}}/T_{\mathrm{ref}}$, and the figures below label it $\bar{V}$. Exact score ties prefer the smaller predicted endpoint error, then the shorter calibrated duration, then batch order. After execution, the queue receives the measured increment $[T(r)/T_{\mathrm{ref}}](x_{r+1} - \sigma)$.

We report TARS, $\sum_r E_s(r)/\sum_r T(r)$, and DWAE, $\sum_r T(r)x_{r+1}/\sum_r T(r)$, together with elapsed training time and cumulative shortfall. A fixed update budget does not imply a fixed number of examples, since B16 processes 32,000 examples and B512 processes 1,024,000. We therefore also report time, shortfall, and examples through the first observed endpoint at or below 0.05. These common-target metrics include the successful frame, and runs that miss the target within the budget are marked.

Policy evaluation uses three fresh initialization seeds, each paired with eight renewable-supply seeds under independent supply. Deterministic baselines need one GPU run per initialization, and their measured trajectories are replayed against the paired supply inputs. Every adaptive LYA trajectory is trained under the specific input used to make its decisions. Prediction parameters and experimental settings are fixed before these policy runs. GPU training time excludes validation and controller instrumentation, whose times are logged separately. Scenario energy uses measured training duration and the independently profiled GPU power.

O.2 Independent renewable supply

Each frame receives an independent uniform power draw on $[0.5P_{\mathrm{ref}}, 2P_{\mathrm{ref}}]$, or approximately $[47.24, 188.97]$ watts, held through that frame. All policies share the same frame-indexed draws for a paired comparison, and $\bar{V} = 1$. This is an independent supply process at renewal boundaries, although the evolving training state remains action dependent. The comparison uses all seven policies and reports error alongside energy shortfall.

Figure 6 shows the fixed-budget results, where LYA has mean TARS 2.284 W and DWAE 0.02542. B16 has slightly lower mean TARS, 2.212 W, with DWAE 0.04451, so LYA operates near the low-power baseline while reducing duration-weighted error by 42.9%. LYA reaches the common target in 0.501 seconds on average, compared with 1.243 seconds for B16, a reduction of 59.7%. The complete 100-frame run is nevertheless longer under LYA, 9.738 versus 8.563 seconds, because the schedules continue after reaching the target.

Relative to B128, LYA reduces mean TARS by 72.5%, from 8.307 to 2.284 W, with a mean target time of 0.501 versus 0.442 seconds. B512 and DEC reach the target in 0.479 and 0.456 seconds, respectively, but incur much larger shortfall rates. These comparisons distinguish renewable matching from the use of a uniformly small batch. The controller can select a more demanding batch when the current supply covers it, while its error feedback changes the choices made during early training.

The paired LYA-minus-B16 TARS difference is 0.072 W, with a two-factor bootstrap interval of $[-0.128, 0.331]$ W that contains zero, so we interpret the energy rates as close in this experiment. The corresponding DWAE difference is -0.01909, with interval $[-0.02026, -0.01791]$. The improvement in time to target also requires more examples, 17,307 on average through the crossing for LYA versus 4,693 for B16. Table 4 and Appendix P.10 report the components of this tradeoff.

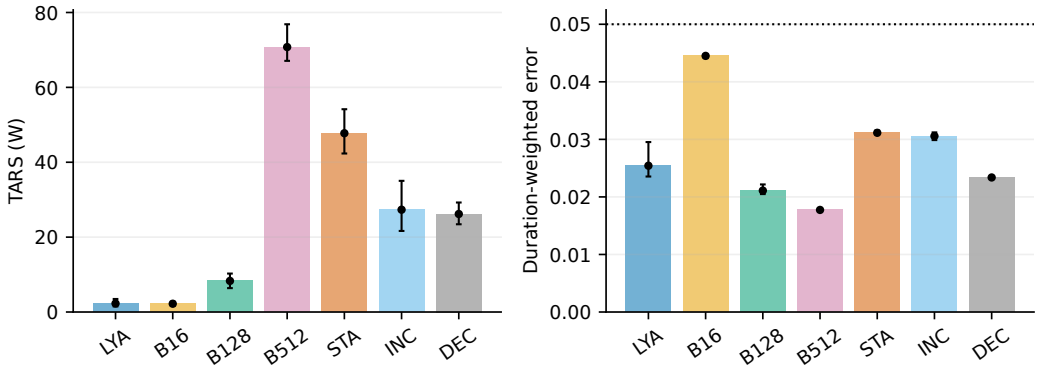


Figure 6: Independent renewable supply at $\bar{V} = 1$. Bars show mean per-run metrics over three initialization seeds crossed with eight paired supply seeds, whiskers span the observed range, and the dotted line marks the error budget.

Table 4: Independent-supply comparison. Means are taken over the 24 paired initialization–supply combinations. Target time ends at the first measured error at or below 0.05. Full time includes all 100 frames.

Policy	TARS (W)	DWAE	Target time (s)	Full time (s)
LYA	2.284	0.02542	0.501	9.738
B16	2.212	0.04451	1.243	8.563
B128	8.307	0.02110	0.442	9.529
B512	70.762	0.01774	0.479	15.701
STA	47.753	0.03114	1.215	12.202
INC	27.325	0.03057	1.201	10.340
DEC	26.180	0.02338	0.456	10.307

O.3 Effect of phase weights

We compare $(100, 1)$, $(1, 1)$, and $(1, 100)$ at $\bar{V} = 1$, using the same first renewable-supply seed and each of the three evaluation initializations. The loss trajectories use measured validation observations and cumulative measured training time. Time to 5% of the initial loss provides a common target for the weight comparison.

Figure 7 shows the three trajectories, with mean target times of 0.498 seconds for $(100, 1)$, 0.536 seconds for $(1, 1)$, and 0.531 seconds for $(1, 100)$. The early-emphasis setting has a 7.0% smaller mean than equal weights in this comparison. Across the three initializations, its target times range from 0.496 to 0.502 seconds. The other ranges are 0.471–0.662 and 0.464–0.662 seconds, so the ranges overlap and the mean ordering does not hold for every initialization.

The full-budget DWAE values are close, at 0.02558, 0.02612, and 0.02531, respectively, so in this experiment the weight settings mainly affect transient batch decisions and their target crossings. This difference between a transient objective and a full-run average motivates reporting both the loss trajectories and target times.

O.4 Daily photovoltaic supply and training start time

The daily-supply experiments start at 00:00 with $\bar{V} = 0.1$ and at 12:00 with $\bar{V} = 10$. The input is measured residential PV generation from Open Power System Data (Open Power System Data, 2020), series `DE_KN_residential1_pv` on 1 July 2015. The day and series are selected by a data-completeness rule before policy evaluation. We scale this measured shape to the GPU power range by setting the peak of the trace to $2P_{\text{ref}} = 188.97$ watts.

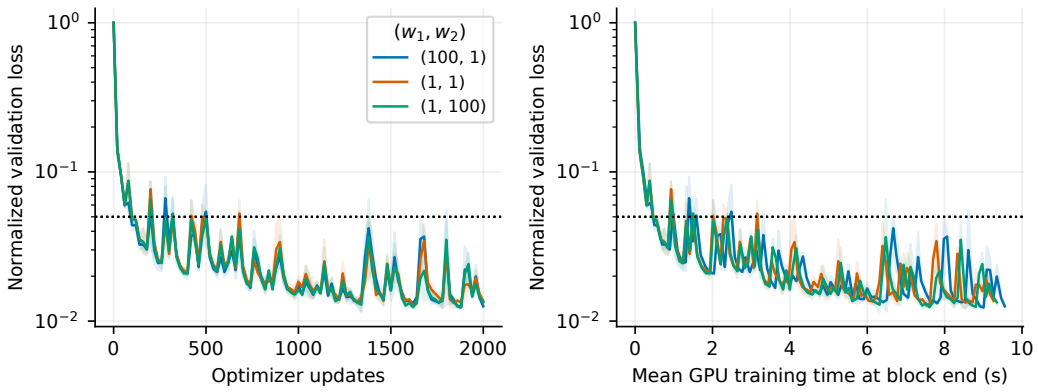


Figure 7: Effect of phase weights under a shared independent-supply input. Lines show mean observed loss over three initialization seeds, and shading spans their smallest and largest loss. The dotted line is the common target.

To expose a short training workload to a daily supply cycle, we map 100 reference blocks to 12 calendar hours, which gives the conversion $\kappa = 12/(100T_{\mathrm{ref}})$ hours per second of GPU training. All policies use this conversion, and the 24-hour input repeats if needed. Measured training seconds determine the queue, energy accounting, and reported duration, while the mapped hours identify the position in the PV trace. These are controlled supply scenarios, rather than GPU runs that physically last twelve hours.

The controller scores actions using only power observed at the current boundary. After a frame completes, its renewable energy is integrated exactly over every crossed two-minute interval. Main shortfall follows the frame-energy definition in Equation (1), with an additional integral of instantaneous positive power deficit recorded separately. Appendix P.5 gives the conversion and integration details.

Figure 8 compares the two prescribed scenarios. At midnight with $\bar{V} = 0.1$, LYA has TARS 66.89 W and DWAE 0.03255, while B16 has 51.61 W and 0.04451. LYA reaches the target in 0.541 seconds, compared with 1.243 seconds for B16, but spends 92.60 versus 88.35 joules of shortfall through that crossing. B128 reaches the target in 0.442 seconds with 41.75 joules. Early progress therefore has an energy cost when little renewable power is available.

At noon with $\bar{V} = 10$, LYA has TARS 30.46 W, 32.7% below the 45.28 W of B128, and their mean target times are nearly equal at 0.442 seconds. The DWAE of LYA is 0.02769, compared with 0.02110 for B128 and 0.04451 for B16. All three noon runs have zero frame-level net shortfall through the target crossing, although over the same prefixes the instantaneous positive deficit averages 1.67 joules. Over the full horizon, B16 again has the lowest shortfall rate, 27.86 W. These results show the role of the error trajectory when interpreting an energy-rate comparison.

The panels in Figure 8 change both start time and $\bar{V}$. To separate those factors, we also compare midnight and noon at the same $\bar{V} = 0.1$, using the existing parameter-sweep runs. The mean TARS of LYA falls from 66.89 to 33.81 W, the mean DWAE from 0.03255 to 0.02779, and the mean target time from 0.541 to 0.480 seconds. The corresponding shortfall-rate decrease is 49.5% for this PV allocation and clock mapping. Capacity, $\bar{V}$, and the full decision traces are examined separately below.

P Experimental Details

This appendix describes the language-model pilot, the CNN workload and loss phases, runtime and power calibration, the measured PV input, the control-parameter comparison, and the online scheduling traces.

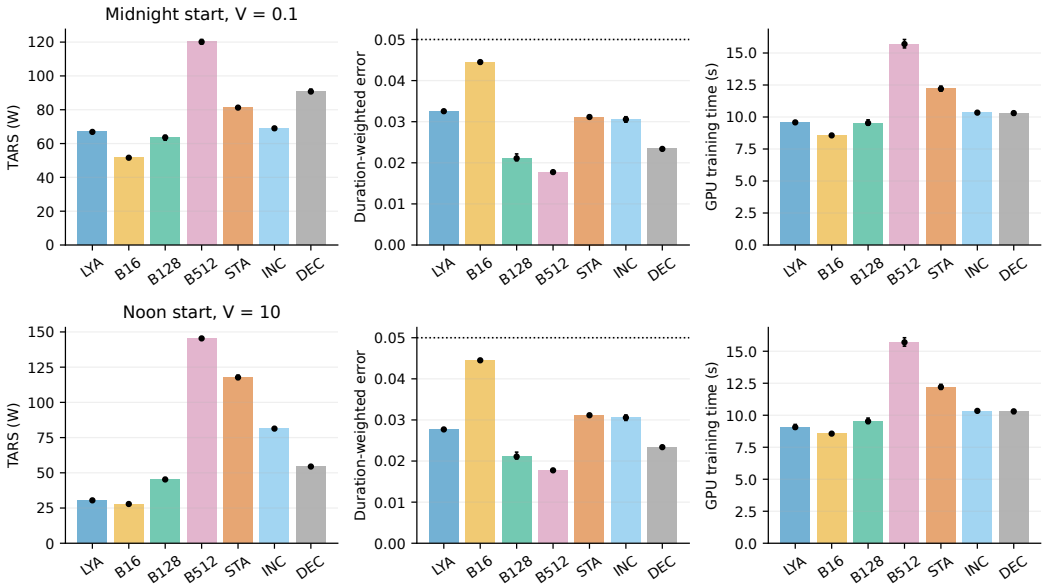


Figure 8: Policy comparison under the two daily PV scenarios, midnight at $\bar{V} = 0.1$ and noon at $\bar{V} = 10$. Bars are means over three initialization seeds, and whiskers span their observed ranges.

P.1 Motivating language-model pilot

The pilot studies how a change in batch size affects power, epoch duration, and loss during masked-language-model training. The workload is a BERT-style model with hidden width 256, four transformer layers, four attention heads, and feed-forward width 1,024, trained on WikiText-2. Token sequences have maximum length 128, and the masking probability is 15%.

The schedule uses batch size 64 for three epochs, followed by batch size 128 for seven epochs. Both stages use the same architecture and the LAMB optimizer (You et al., 2020). The peak learning rate is 5×10^{-4} and weight decay is 0.01. A single learning-rate schedule covers all optimizer updates, with 10% warmup followed by linear decay. GPU power is sampled through NVML every five seconds. Epoch duration, training loss, validation loss, and learning rate are recorded against elapsed time.

The analysis aligns the batch schedule with power, epoch boundaries, and loss. A separate view aligns loss with the recorded learning rate. These views examine whether changes in system cost occur at the batch-size transition or follow the learning-rate schedule, and the measured traces retain the initial overhead of each stage.

The experiment runs on one NVIDIA A800-SXM4-80GB using PyTorch 2.5.1, CUDA 12.4, and Transformers 4.46.3, with float32 computation and TF32 disabled. The model has 11,103,290 parameters and is initialized with seed 20270908. We use the training and validation splits of `wikitext-2-raw-v1` and the `bert-base-uncased` tokenizer, concatenate nonempty tokenized lines in order, and pack 126 content tokens with the CLS and SEP tokens to form sequences of length 128. Discarding the incomplete final block of each split leaves 18,283 training sequences and 1,894 validation sequences.

Training masks are sampled dynamically, and among the selected tokens, 80% are replaced by MASK, 10% by a random token, and 10% remain unchanged. Validation uses the same masking seed after every epoch, without changing the training random state. LAMB uses bias correction, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-6}$, with a trust ratio for each parameter tensor and weight decay on all parameters. The optimizer state and model are retained across the batch-size transition.

The run completes 1,859 updates, including 186 warmup updates, and records 32 NVML power readings over 155.98 seconds of elapsed training and validation time. Epoch losses are averaged by masked-token count, and epoch training duration excludes validation. Because the continuous

power trace includes validation, the mean powers reported with Figure 3 use the samples whose timestamps fall within training intervals, giving 10 readings at batch size 64 and 21 at batch size 128 (one initial reading precedes the first training interval).

Table 5 reports all epoch results, where the first three epochs have mean training duration 16.80 seconds, compared with 14.29 seconds for the final seven. Validation loss decreases from 9.174 after the first epoch to 6.986 after the tenth. Figure 9 places the recorded learning rate and training loss on the same elapsed-time axis. The learning-rate schedule continues through the batch-size transition.

Table 5: BERT pilot measurements on WikiText-2. Epoch time covers training, and validation loss is measured afterward.

Epoch	Batch	Training time (s)	Training loss	Validation loss
1	64	16.987	9.861	9.174
2	64	16.706	8.579	7.961
3	64	16.693	7.550	7.237
4	128	14.275	7.186	7.126
5	128	14.288	7.100	7.073
6	128	14.280	7.050	7.039
7	128	14.315	7.022	7.015
8	128	14.300	7.007	6.998
9	128	14.326	6.986	6.989
10	128	14.266	6.982	6.986

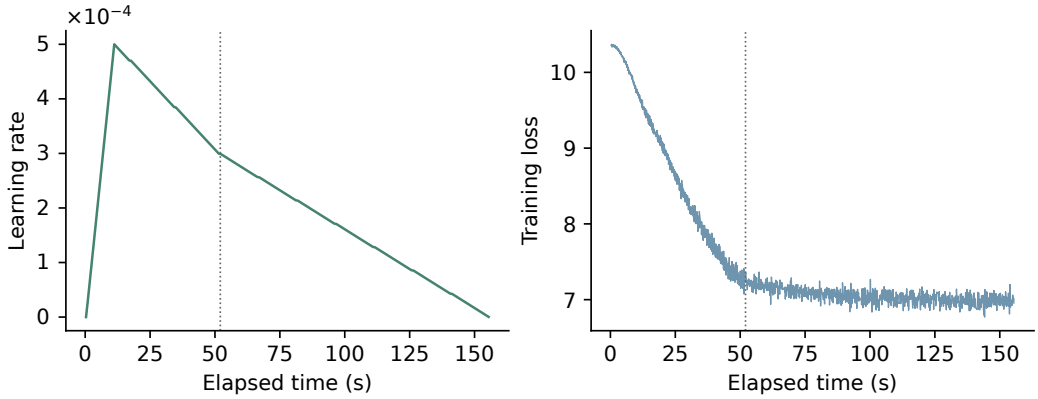


Figure 9: Recorded learning rate and training loss in the BERT pilot, both plotted against elapsed wall-clock time. The dotted line marks the batch-size transition, while a single warmup and decay schedule spans the full run.

P.2 CNN AND MNIST CONFIGURATION

The CNN task uses the 60,000 MNIST training images (LeCun et al., 2010). The first 55,000 images form the training split and the final 5,000 form the validation split. Inputs have shape $1 \times 28 \times 28$, with pixel values divided by 255. The network contains two 5×5 convolution layers with 32 and 64 output channels and padding two. Each convolution is followed by ReLU and 2×2 max pooling. A fully connected layer has 1,024 ReLU units and dropout probability 0.4, and the output has ten classes. Training uses Adam with learning rate 10^{-3} and cross-entropy loss.

The loss-phase measurements use PyTorch 2.12.1 on a local CPU, with four threads, seed 20270908, and deterministic operations enabled. Batch sizes 16, 128, and 512 each have a single loss-phase run of one, two, and five epochs, respectively, including the final partial mini-batch of each epoch. Table 6 reports the measured validation results and training times.

Table 6: CNN training measurements on MNIST. Validation loss and accuracy are evaluated after the final epoch. CPU time sums the training-epoch timers and excludes validation and data download.

Batch	Epochs	Updates	CPU time (s)	Validation loss	Accuracy (%)
16	1	3438	48.250	0.05239	98.40
128	2	860	49.658	0.04111	98.88
512	5	540	117.253	0.03359	98.96

P.3 Loss phases and their measurement

Training loss is recorded after every optimizer update. The remaining fraction of the observed loss reduction is

$$\rho_l = \frac{l - l_{\text{final}}}{l_{\text{initial}} - l_{\text{final}}}. \tag{115}$$

The denominator is the initial-to-final loss reduction within each run. The levels $[1.0, 0.8, 0.6, 0.4, 0.1, 0.0]$ define successive training stages. Figure 4 shows the raw mini-batch losses and a trailing mean over up to 25 updates. Each boundary is marked where this mean first crosses the corresponding level, and the initial and final losses are taken from the same smoothed trajectory.

This normalization is used only to describe the plotted CPU loss phases. The scheduling experiments use the independently measured, unsmoothed validation cross-entropy normalized by its initialization value. Their phase detector operates on successive 20-update endpoints.

P.4 Runtime and power calibration

The calibration protocol uses batch sizes $\{16, 32, 64, 128, 256, 512, 1024\}$ and two training epochs for each configuration. GPU power is queried through NVML after each mini-batch, and elapsed training time is recorded for every epoch. The per-epoch measurements are averaged within each batch size. Nonlinear least squares then fits the duration and power models in Equation (108). Measurement points and fitted curves are displayed together to show how well the model describes the observed batch-size dependence.

The pilot uses an NVIDIA A800-SXM4-80GB, driver 550.144.03, CUDA 12.4, PyTorch 2.5.1, and TorchVision 0.20.1. Each batch-size configuration starts from a new model with seed 20270908 and uses the CNN and optimizer described in Appendix P.2. Computation is in float32, TF32 is disabled, deterministic PyTorch operations are enabled, and the CPU thread count is four.

Each NVML reading follows a synchronized training update, and epoch power is the arithmetic mean of these samples. Epoch duration measures the training loop, including data transfer, telemetry, and logging, while excluding validation. The two epochs are consecutive observations from one run per batch size. Table 7 reports their means, and Figure 5 displays both observations separately.

Table 7: Measured CNN calibration on the A800. Each entry is the mean of two training epochs.

Batch size	Epoch time (s)	Sampled GPU power (W)
16	15.0160	71.062
32	7.4597	74.671
64	3.8430	81.730
128	2.1839	94.483
256	1.2408	147.282
512	0.8432	189.201
1024	0.6287	216.327

Within the family $aB^{-k}+c$, with $a, c \geq 0$ and $k \in [-4, 4]$, the fit minimizes squared deviations from the seven configuration means. Starting from six initial exponent values, it retains the solution with the smallest residual sum of squares, and neither fitted exponent reaches its bound. The maximum absolute deviation from a configuration mean is 0.0794 seconds for runtime and 15.96 watts for

power, and the fitted curves describe the measured batch-size range. All individual samples, epoch aggregates, coefficients, and residuals are saved with the experiment.

P.5 Measured PV input and coupled evaluation

The PV input comes from the Open Power System Data household release dated 15 April 2020 (Open Power System Data, 2020). We select `DE_KN_residential1_pv`, the first residential PV column in lexicographic order, and the earliest complete July day with positive generation, 1 July 2015. The selection uses data completeness, valid cumulative-energy counters, and the selected series' interpolation flags, but not policy outcomes.

The day is defined in CEST (UTC+02:00), from 22:00 UTC on 30 June to 22:00 UTC on 1 July. The source subset contains 1,441 one-minute cumulative-energy readings, including both midnight boundaries. For a counter C in kWh, the two-minute average power is

$$P_{\mathrm{PV}}(t) = 30000\big[C(t + 2\,\mathrm{min}) - C(t)\big]\ \mathrm{W}. \tag{116}$$

This gives 720 interval averages, each held constant over its two-minute interval. The selected subset has no missing counters, negative counter differences, or selected-series interpolation flags, and it retains the regularization applied by the publisher. Daily generation is 59.461 kWh, and peak unscaled power is 8,340 W. The primary allocation is $2P_{\mathrm{ref}}/8340$, giving the peak scenario power of 188.97 W shown in Figure 10. Source counters, timestamps, selection code, and file hashes accompany the processed trace.

Let $p(h)$ be the allocated power at calendar hour h, extended with period 24, and let $A(h) = \int_0^h p(u)\,du$. A frame starting at hour h_r with measured training duration $T(r)$ ends at $h_r + \kappa T(r)$, where $\kappa = 12/(100T_{\mathrm{ref}})$. Renewable energy on the GPU training clock is

$$E_{\mathrm{ren}}(r) = \frac{A(h_r + \kappa T(r)) - A(h_r)}{\kappa}. \tag{117}$$

The integral is exact for the piecewise-constant input, including frames that cross interval or midnight boundaries. Power is in watts and the integration variable is in hours, so division by κ converts the result to joules on the training clock. This input transformation compresses the daily supply pattern and does not represent the energy consumed by twelve hours of continuous GPU execution.

The controller uses only $p(h_r)$ when scoring candidates. The realized interval integral is computed after the chosen training block completes. The main shortfall is $\max\{P(B_r)T(r) - E_{\mathrm{ren}}(r), 0\}$, consistent with the frame-level energy accounting in Section 2. We separately record

$$E^{\mathrm{inst}}_{\mathrm{grid}}(r) = \frac{1}{\kappa}\int_{h_r}^{h_r+\kappa T(r)} \max\{P(B_r) - p(u), 0\}\,du. \tag{118}$$

This quantity is the positive instantaneous deficit without netting within a frame, so it can exceed the main shortfall when supply crosses GPU demand during that frame.

The evaluation couples a renewable-input component to a training process, as Figure 11 shows. The controller uses the current supply and calibrated candidate predictions, then receives the measured loss and duration after executing its chosen batch. The energy component integrates the input over that duration.

Figure 12 changes the allocated PV peak while retaining the same measured daily shape. At noon with $\bar{V} = 10$, the mean TARS of LYA is 52.10, 35.17, 30.46, and 33.01 W at peak-to-reference ratios 0.5, 1, 2, and 4, and its DWAE is 0.03999, 0.03406, 0.02769, and 0.02594. More available power supports lower error, but the full-run shortfall rate need not decrease monotonically because an adaptive batch schedule also changes demand and the interval of the daily trace that the run covers. For example, the mean training duration increases from 9.067 seconds at ratio 2 to 10.432 seconds at ratio 4.

We also retain two absolute supply settings as sensitivity checks. With independent power on $[40, 80]$ watts, LYA has TARS 16.76 W and DWAE 0.03672, while B16 has 13.46 W and 0.04451. With PV multiplied by 0.1, the peak is 834 W, or $8.83P_{\mathrm{ref}}$. The noon LYA run has mean TARS 31.67 W and DWAE 0.02639. Despite the high peak, its full-horizon shortfall remains positive because the mapped run extends beyond the high-supply interval. Figure 13 reports all policies under these settings.

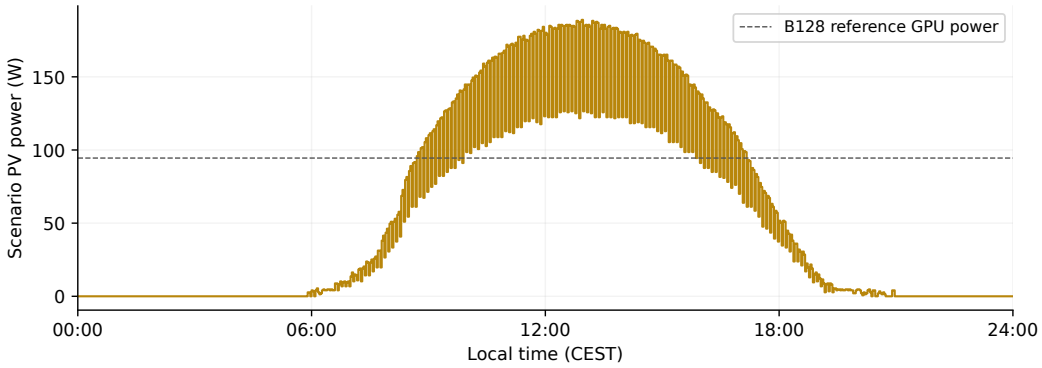


Figure 10: Measured PV shape after allocation to the evaluated GPU. Two-minute power averages are scaled to peak $2P_{\mathrm{ref}}$. The underlying series is `DE_KN_residential1_pv` on 1 July 2015.

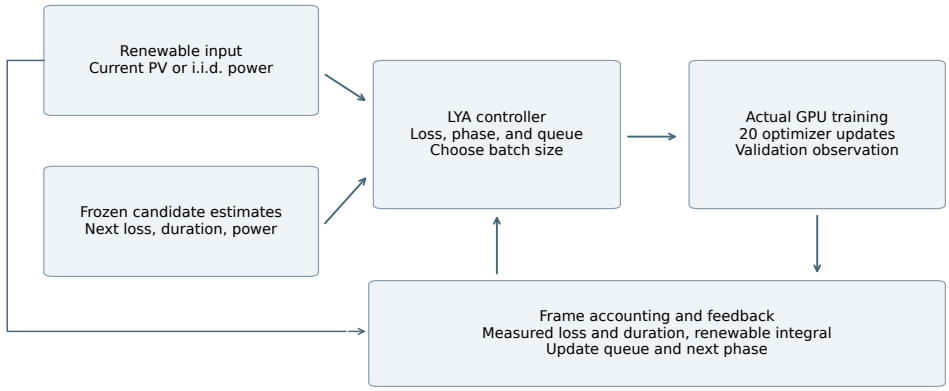


Figure 11: Coupled training and energy evaluation. Candidate estimates guide the controller, the GPU executes each chosen block and returns observed loss and duration, and the energy component accounts for supply over that interval.

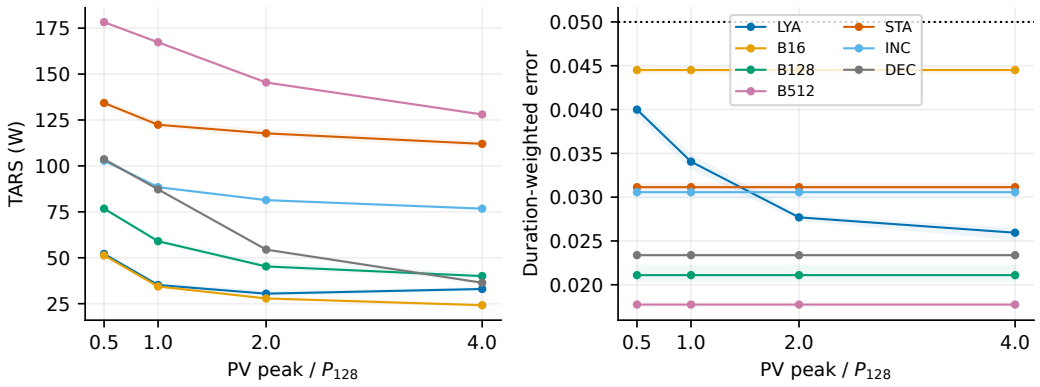


Figure 12: PV allocation sensitivity at noon and $\bar{V} = 10$. Lines join three-initialization means, and shading shows their minimum and maximum. The input shape and time mapping are unchanged. The dotted horizontal line denotes the error budget.

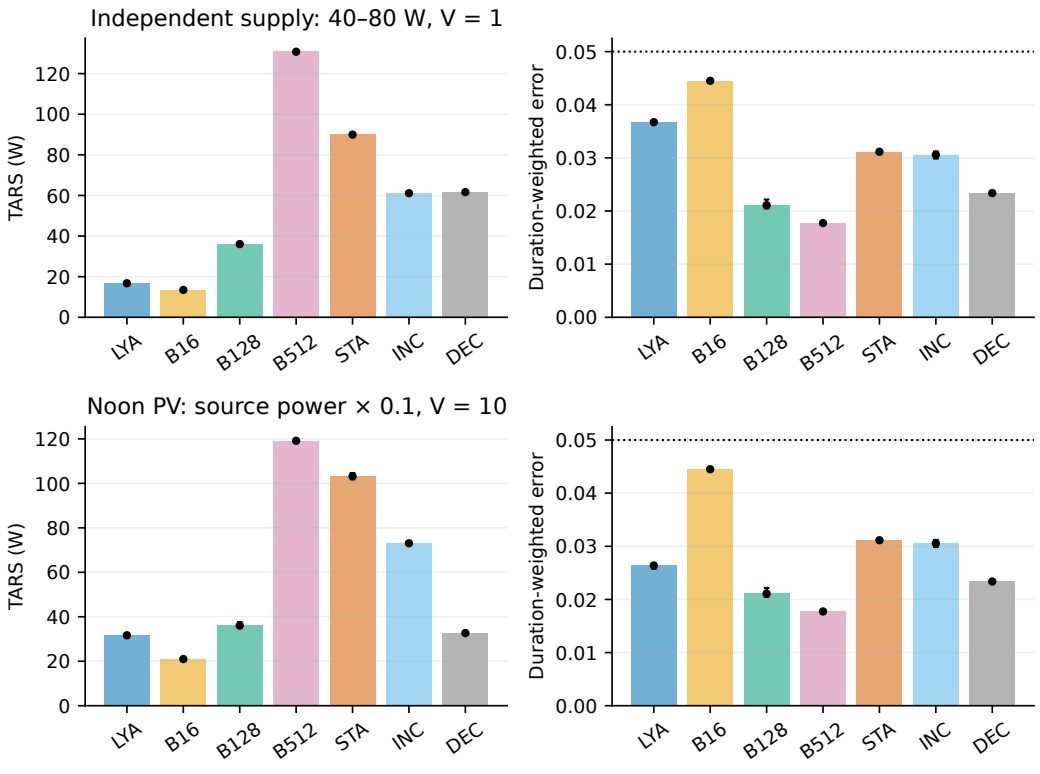


Figure 13: Sensitivity to absolute supply levels. Top: independent power on $[40, 80]$ watts, one supply seed and three initializations. Bottom: the measured PV shape multiplied by 0.1, a noon start, and $\bar{V} = 10$.

P.6 Dependence on the control parameter

The parameter comparison starts at noon and uses $\bar{V} \in \{0.1, 1, 10\}$ in the normalized units of Equation (21). Architecture, target, detector, initialization seeds, PV allocation, and clock mapping are shared. Each adaptive trajectory is executed separately because its decisions depend on the control parameter. The comparison reports TARS, DWAE, and elapsed training time.

Mean TARS decreases from 33.81 to 31.56 to 30.46 W as $\bar{V}$ increases from 0.1 to 1 to 10. The corresponding cumulative shortfalls are 314.78, 288.92, and 276.27 joules. DWAE stays between 0.02769 and 0.02804, and mean training duration changes from 9.308 to 9.153 to 9.067 seconds. Figure 14 shows the resulting differences and the range over the three initializations. This finite-horizon comparison measures how $\bar{V}$ changes the schedule under a particular daily input.

P.7 Online decisions and interpretation

The online view aligns batch size, validation error, queue, and renewable power on the PV replay clock, a fixed linear transformation of cumulative measured training time. All 100 decision frames are shown, including frames after the first target crossing. The midnight and noon trajectories show the supply intervals encountered by the controller. The measured daily peak occurs after noon, and a long mapped trajectory can also cross midnight. The plotted GPU demand is the independently profiled mean at the selected batch size.

Figures 15 and 16 show the first prescribed evaluation initialization, seed 20270941. The midnight trajectory visits all six candidate batch sizes and spans 0 to 11.99 hours on the replay clock. Its observed boundary supply ranges from zero to 181.49 W. The normalized queue reaches 0.466 and returns to zero by the last frame. In the noon trajectory, five batch sizes are selected over replay hours 12 to 23.14. Its queue peaks at 0.150 and also ends at zero. The recorded validation loss increases

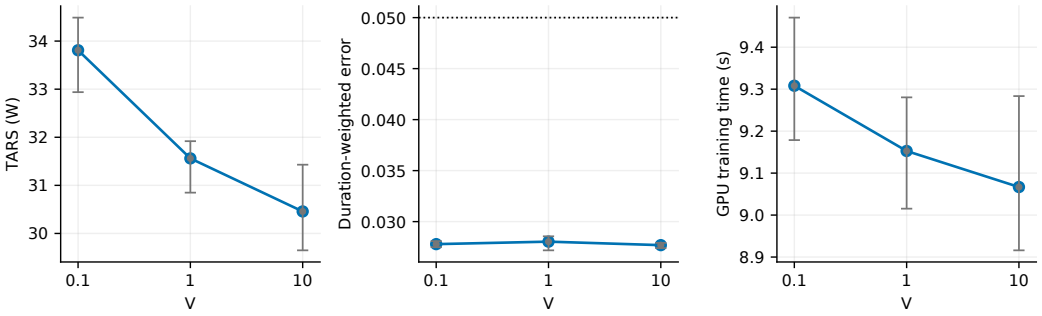


Figure 14: Noon sensitivity to $\bar{V}$ under the same PV allocation and three initialization seeds. Points are means, vertical intervals show minimum and maximum values, and the error-budget line is 0.05.

in 35 midnight frames and 44 noon frames, and these observations remain in both the plotted loss and the queue update.

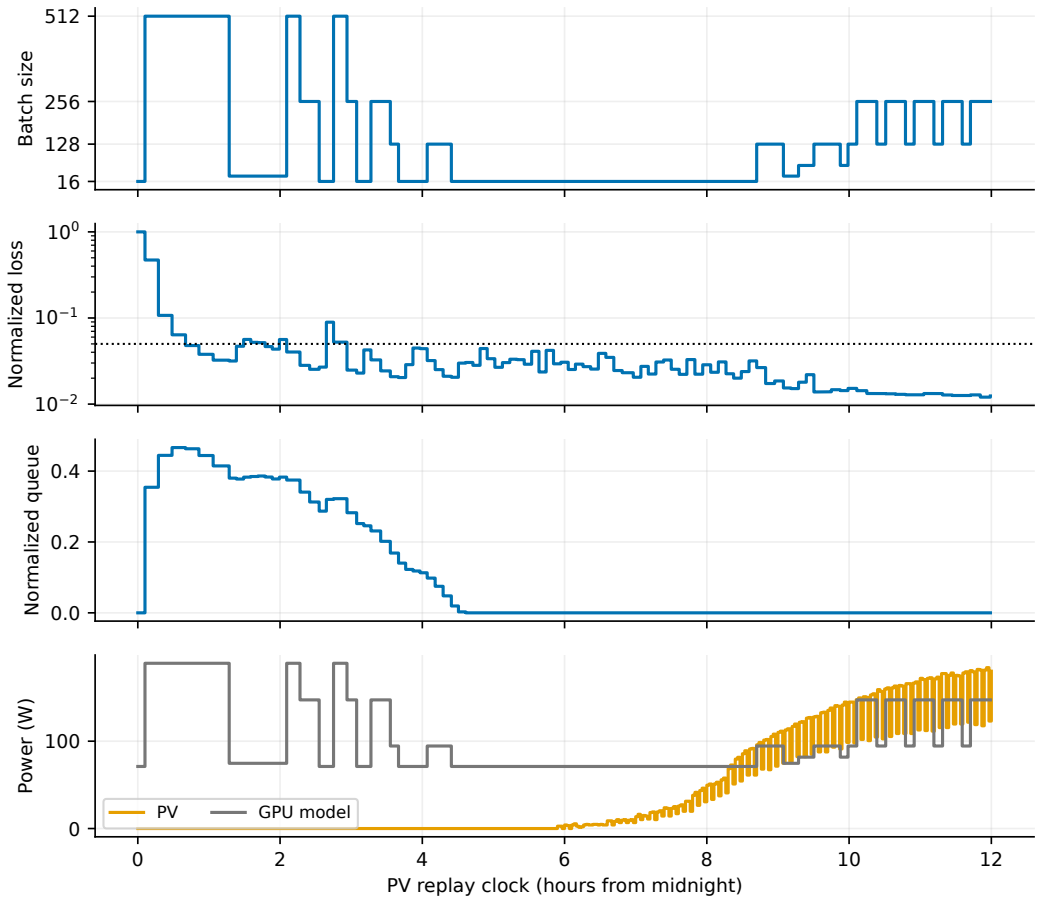


Figure 15: Midnight LYA trajectory at $\bar{V} = 0.1$, seed 20270941, over all 100 frames. The horizontal axis is the PV replay clock, obtained from measured training seconds through a fixed conversion.

P.8 Progress calibration and independent checks

Calibration, predictor validation, and policy evaluation use disjoint initialization seeds. Calibration seeds are 20270921, 20270922, and 20270923. The independent predictor checks use 20270931,

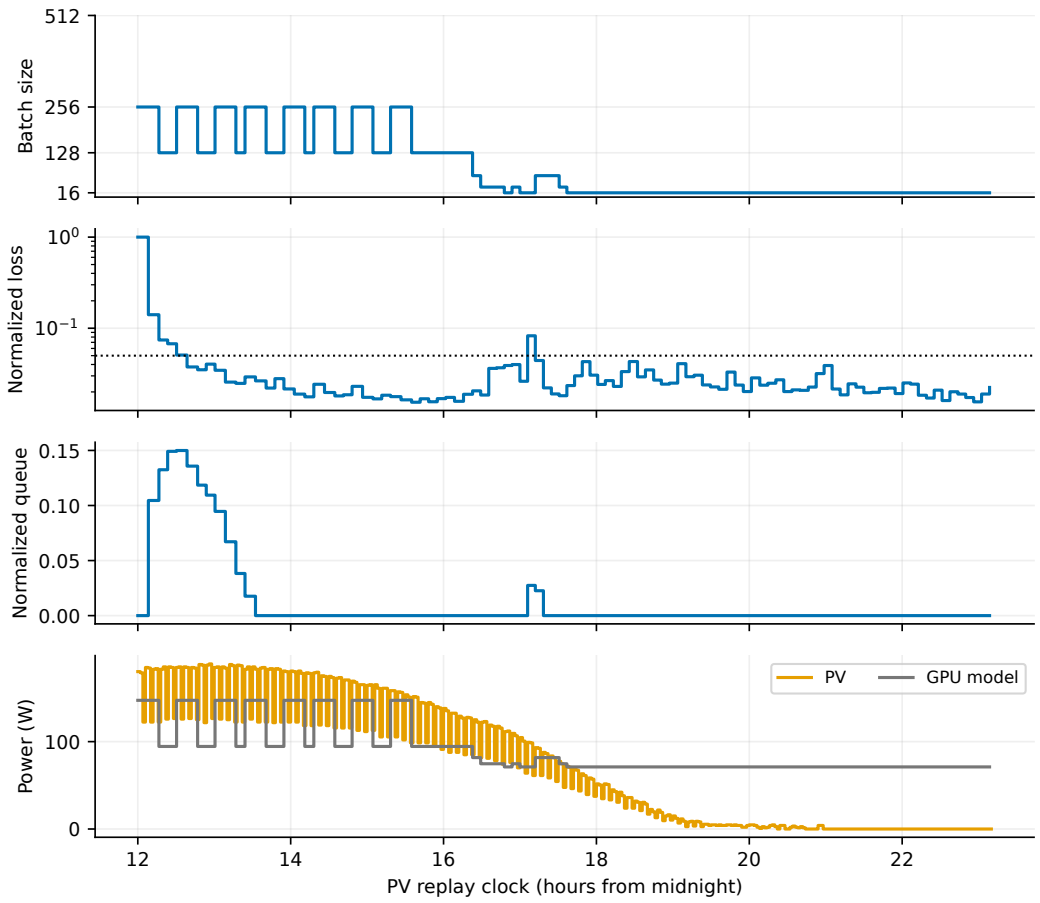


Figure 16: Noon LYA trajectory at $\bar{V} = 10$ for the same initialization seed, drawn with the conventions of Figure 15.

20270932, and 20270933. Policy evaluation uses 20270941, 20270942, and 20270943. All GPU CNN runs use the architecture, split, Adam learning rate, precision, deterministic settings, and hardware in Appendices P.2 and P.4.

For calibration, three reference paths hold batch size at 16, 128, or 512. Checkpoints occur after $\{0, 20, 40, 80, 160, 320, 640, 1280, 2000\}$ optimizer updates. Every checkpoint stores the model parameters, the complete Adam state, the CPU and CUDA random states, the shuffled example permutation and position, and the example-stream generator state. Each candidate branch restores this complete state, verified by checkpoint hashes, before running 20 updates, and the original reference state is restored before its path continues. Different batch sizes use a common shuffled prefix but consume different numbers of examples.

Validation uses the same fixed 5,000 examples, evaluation mode, and probe batch size 256. The input normalization is pixel value divided by 255. A continuous shuffled stream crosses dataset boundaries without dropping or shortening a batch. It uses a separate CPU generator seeded by the initialization seed plus 100,000. Adam has learning rate 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and zero weight decay. Loss increases in 176 of the 486 transitions, and no checkpoint branch or observed increase is removed.

Fitting the ODE family in Equation (107) estimates $\log k_B$, p_B, and c_B separately at each batch, with bounds $10^{-7} \le k_B \le 10^5$, $0 \le p_B \le 4$, and $0 \le c_B \le 1$. Eight starting points combine $p_B \in \{0, 1, 2, 4\}$ with $c_B = 0$ or the clipped 15th percentile of calibration endpoint error. The best residual sum of squares is retained. The log-interpolation alternative averages repeated log-input

locations, applies weighted isotonic regression to log-endpoints, interpolates linearly between fitted nodes, and holds endpoint values constant outside the calibration range.

Model selection leaves out an entire calibration seed, fits the other two, and evaluates mean squared log-endpoint error on the omitted seed, with ties within 10^{-12} favoring the ODE. The aggregate cross-validation errors are 0.10641 for the ODE and 0.10955 for log interpolation, so the ODE is selected and then fit to all calibration observations. Table 8 gives its coefficients and the median measured duration of a 20-update branch. Figure 17 displays all paired observations and fitted transitions.

Table 8: Calibrated one-block progress predictor and measured block duration. The fitted level c_B is a prediction parameter, not an estimate of the best achievable validation loss.

Batch	k_B	p_B	c_B	Block time (s)
16	0.61452	0.01687	0.09440	0.085075
32	1.61451	0.33724	0.06937	0.084802
64	2.43766	0.42903	0.04207	0.087048
128	3.59494	0.57865	0.02930	0.096930
256	5.00051	0.81495	0.00893	0.115645
512	5.40368	0.83630	0.00528	0.160500

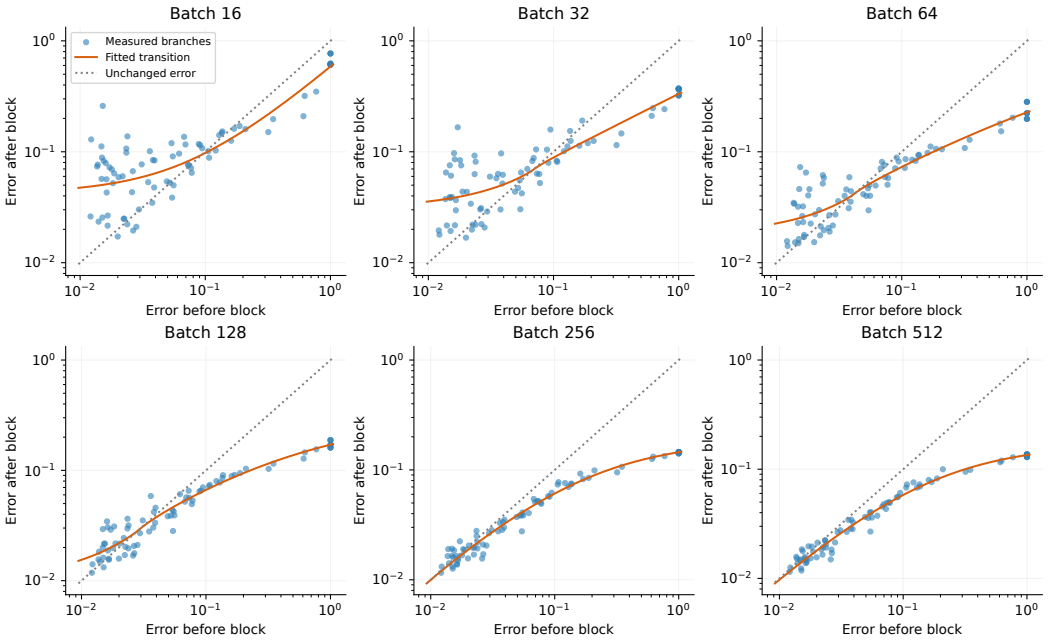


Figure 17: Observed normalized validation loss before and after each 20-update branch, with fitted candidate transitions and a diagonal indicating unchanged loss. Each batch panel contains 81 observations from three initialization seeds and three reference paths.

The independent check executes 100 frames for each of ten schedules and three seeds. Six schedules keep one of the candidate batches fixed, and the others are STA, INC, DEC, and a uniformly random batch schedule using a separate generator seeded by the initialization seed plus 200,000. We assess both one-step predictions from the observed state and recursive predictions initialized at one, without refitting the model to these observations.

Across 3,000 endpoints, one-step RMSE is 0.01606 and mean absolute error is 0.01042. Recursive rollout RMSE is 0.03309, with mean absolute error 0.02474. All 30 measured trajectories reach error 0.05, while 24 model rollouts do. For trajectories where both cross, the mean absolute crossing difference is 8.04 blocks. The mismatch is largest for the low-batch fitted levels and schedules that

start with small batches, as Figures 18 and 19 show directly. The fixed predictor therefore remains an imperfect input to the controller, while policy results use measured feedback and fresh seeds throughout.

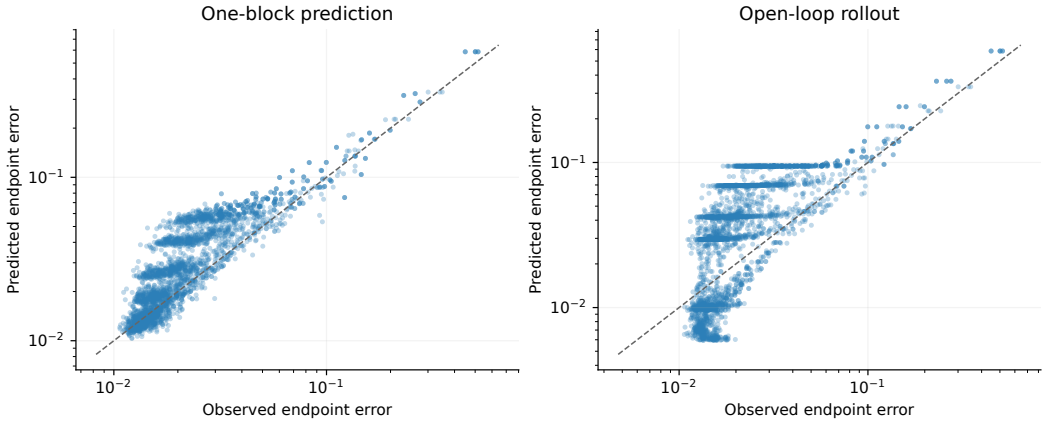


Figure 18: Independent endpoint checks. Left: one-block prediction from the observed starting loss. Right: recursively generated loss along the same recorded batch schedule. Both panels compare predictions with GPU validation observations.

The block-duration predictor is more stable across schedules, with independent RMSE 0.00548 seconds. Measured block duration is used in policy evaluation, including energy accounting, PV clock advancement, and queue increments. Power comes from the longer two-epoch configuration profile. Short-block NVML statistics are retained separately because the device sensor can have a refresh or smoothing interval comparable with these short blocks.

P.9 Execution, aggregation, and uncertainty

Before policy evaluation, the process warms up every candidate batch with ten updates and runs a validation probe using a separate initialization. It then discards this model and optimizer, so recorded runs restart from their prescribed seed. The controller and deterministic baselines use the same initialization and example-stream seed within a paired comparison. Each run preserves its model and optimizer for the full 100-frame budget.

There are 25 measured runs per evaluation seed. Six are deterministic baselines, eight run LYA under independent supply seeds 20271000 through 20271007, two use the additional phase-weight pairs under the first supply seed, and two use the primary midnight and noon PV conditions. Two more cover the remaining noon $\bar{V}$ settings, three cover the remaining PV capacities, one uses independent power on $[40, 80]$ watts, and one uses a PV multiplier of 0.1. The default phase weights, noon $\bar{V} = 10$, and primary capacity are reused in their respective comparisons. In total, evaluation contains 75 GPU runs, 7,500 frames, and 150,000 optimizer updates.

The actions of a baseline do not depend on supply, so its observed error and duration sequence can be replayed for each energy scenario. Adaptive runs are specific to their supply and parameters. Their records are checked by recomputing every selected action, phase update, queue increment, and energy integral from the saved inputs. Main figures summarize per-run ratios rather than pooling all durations into one ratio. Independent-supply results cover three initialization seeds crossed with eight supply seeds, while PV and phase-parameter comparisons use three initialization seeds. The figure captions identify the spread shown, which reflects these training seeds and inputs rather than independent measurements from multiple solar sites or days.

Validation and controller computation are experimental instrumentation, so their elapsed times are recorded separately from training time, and their energy is not added to the modeled training demand. Target time therefore measures the training component needed to reach the observed target. The complete records also retain probe time, processed-example count, final validation accuracy, and instantaneous-deficit energy for further comparisons.

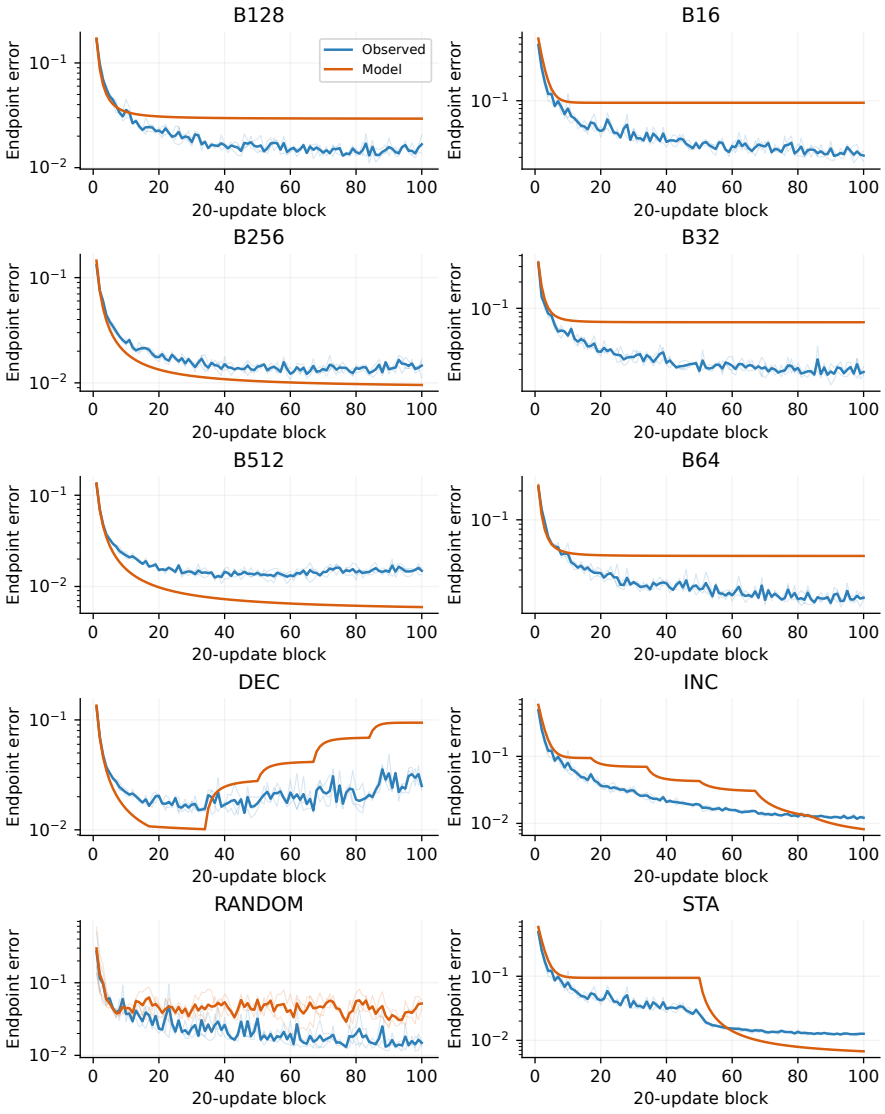


Figure 19: Independent observed and predicted loss trajectories for ten schedules. Faint lines show the three initialization seeds, and solid lines show their means. Predicted rollouts receive the recorded actions but no intermediate validation observations.

P.10 Comparison at a common quality target

Every reported evaluation reaches normalized validation loss 0.05 within 100 frames. Across all evaluated supply–policy combinations, the largest full-horizon DWAE is 0.04476. These are finite-budget measurements, so they do not establish the asymptotic stochastic assumptions of the theoretical results.

Table 9 gives the independent-supply target costs through the measured first crossing, including the successful block, and keeps time, energy, and example count separate. This separation shows that a small batch processes fewer examples before the target but can take more GPU training time, while a large fixed batch can reach the target quickly with high renewable shortfall. Adaptive scheduling uses the available supply to change this balance. Figure 20 shows these costs together with those of the noon PV scenario.

The training-component times in the main comparisons exclude instrumentation, which is measured separately. Across the 75 GPU runs, mean training time is 9.898 seconds, mean validation-probe

Table 9: Mean costs through the common quality target under independent supply. Each policy is evaluated on the same 24 initialization–supply pairs, and all reach the target.

Policy	Training time (s)	Shortfall (J)	Examples (10^3)
LYA	0.501	3.496	17.307
B16	1.243	2.985	4.693
B128	0.442	4.008	11.093
B512	0.479	31.469	30.720
STA	1.215	2.941	4.693
INC	1.201	2.924	4.800
DEC	0.456	29.939	30.720

time is 1.949 seconds, and action selection takes 0.0145 seconds per run on average. Mean recorded run wall time is 11.931 seconds, including initialization and bookkeeping.

For independent-supply paired differences, we use 10,000 bootstrap draws. Each draw independently samples three initialization levels and eight supply levels with replacement, forms their Cartesian product, and retains the LYA–baseline pairing. Only three initialization levels are available, so these intervals describe uncertainty within the measured design. The raw per-initialization values, per-supply values, and all matched differences are also provided. PV error bars instead show the observed range over three training initializations on one measured solar day.

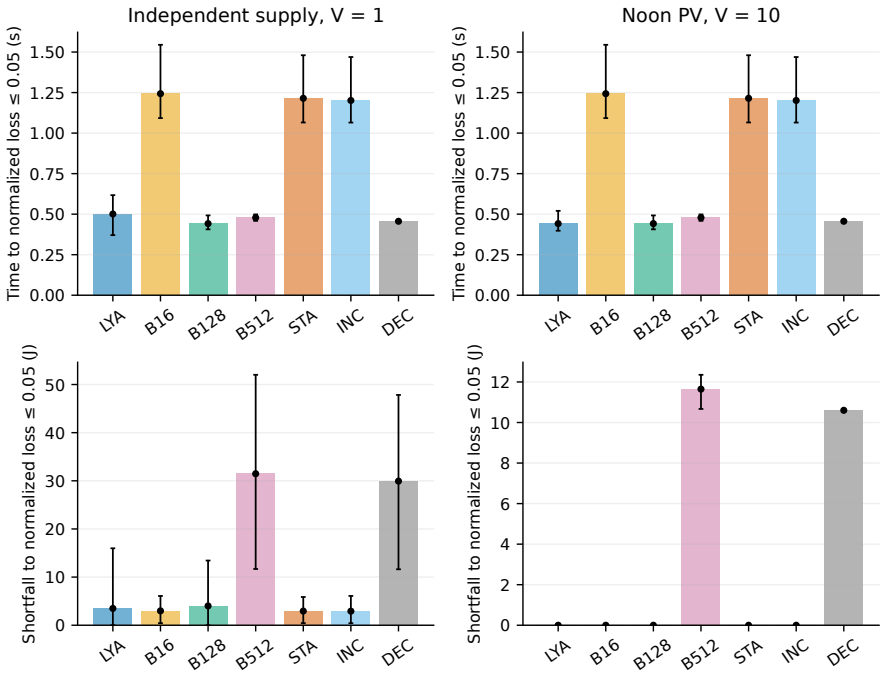


Figure 20: Training time and renewable shortfall through the first observed target crossing. Left: independent supply, 24 paired conditions. Right: noon PV at $\bar{V} = 10$, three initializations. Bars show means and whiskers span observed ranges.