GRAPHBC: GRAPH-EMBEDDED BIAS CORRECTION FOR DIFFERENTIABLE HYBRID SIMULATORS UNDER MODEL–PLANT MISMATCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Hybrid simulators combine mechanistic modules that are built on simplifying assumptions with data-driven modules learned from noisy or incomplete data. Local model–plant mismatches therefore compound through multi-stage computation, creating a dilemma: calibrating each module independently keeps intermediate states physically meaningful but leaves large terminal errors, whereas end-to-end training reduces terminal errors by letting modules compensate for one another's mismatch, at the cost of intermediate states and parameters losing their physical meaning. To address this dilemma, we propose **Graph**-embedded **B**ias **C**orrection (GraphBC), which represents a hybrid simulator as a differentiable computation graph and embeds a learnable correction at an internal node, trained jointly with the simulator. We decompose simulation errors into propagated and local components and derive upper bounds along the error propagation. We then derive a gradient-alignment condition under which terminal supervision also improves intermediate states. Finally, we establish a turning-point criterion for correction placement in scaling-compatible chains. We evaluate GraphBC on two simulators driven by real-world data. On a freeway traffic simulator, it reduces terminal error by 29–43% relative to end-to-end training while shifting simulator parameters about 37% less, and gradient alignment identifies which intermediate states improve. On an industrial-scale ride-hailing simulator, it reduces terminal MAPE by 68% while preserving the physical meaning of intermediate states.

1 INTRODUCTION

Differentiable hybrid simulators increasingly combine data-driven models with mechanistic equations, numerical solvers, and structured components to model real-world systems (Belbute-Peres et al., 2018; Nava et al., 2022; Willard et al., 2022). However, both kinds of components are imperfect: mechanistic modules rest on simplified assumptions (Ensinger et al., 2024), while data-driven modules learn from noisy, incomplete, or biased data. These local model–plant mismatches can accumulate or amplify downstream and corrupt the terminal output (Um et al., 2020; Wei et al., 2025).

This creates a dilemma. Calibrating each module independently keeps intermediate states physically meaningful, but residual errors compound into a large terminal error. Training all modules end to end reduces this error, but lets modules compensate for one another's mismatch, so that intermediate states and parameters lose their physical meaning (Brynjarsdóttir & O'Hagan, 2014). Ideally, a simulator should be accurate at its terminal output while its intermediate states remain physically meaningful.

Existing bias-correction methods aims to address this dilemma (Table 1; Appendix A). Post-hoc correction improves the terminal output but leaves internal errors unchanged (Berg et al., 2012; Zarei et al., 2021); discrepancy modeling separates model error from simulator parameters at the output, with constraints keeping parameters physically meaningful (Kennedy & O'Hagan, 2001; Plumlee, 2017); and residuals embedded in the simulator correct errors inside the dynamics (Srikishan et al., 2025) and can preserve the physical parameters of a single dynamical model (Yin et al., 2021). However, the dilemma remains open for multi-stage simulators with heterogeneous modules, which admit multiple correction positions. We therefore embed a learnable correction at an internal graph node and train it jointly with the simulator, so that it can absorb the mismatch while simulator components retain their physical roles. However, realizing this idea raises three main challenges.

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.

Table 1: Positioning of GraphBC. Prior methods correct single models at fixed locations, whereas GraphBC treats placement as a design variable and analyzes intermediate fidelity.

Method	Correction location	Simulator	Fidelity analyzed	Placement
Post-hoc correction[a]	Output	Single	–	Fixed
Discrepancy calibration[b]	Output	Single	Parameters	Fixed
APHYNITY[c]	Dynamics	Single	Parameters	Fixed
Solver-in-the-Loop[d]	Each solver step	Single	–	Fixed
NeuralSim[e]	Multiple	Single	–	Empirical
GraphBC (ours)	**Any internal node**	**Multi-module**	**States + parameters**	**Criterion**

[a] Berg et al., 2012; Zarei et al., 2021; [b] Kennedy & O'Hagan, 2001; Plumlee, 2017; [c] Yin et al., 2021; [d] Um et al., 2020; [e] Heiden et al., 2021.

First, local mismatch is entangled with propagated upstream error. The error observed at a node mixes inherited error with the node's own mismatch, and downstream sensitivities may amplify or attenuate both (Neary & Topcu, 2023; Wei et al., 2025), so the magnitude of a local error alone does not determine its terminal impact. This calls for separating the two and quantifying the downstream amplification associated with each graph position.

Second, terminal-loss reduction does not necessarily improve intermediate states. Terminal gradients propagate through multiple paths and jointly update trainable components (Belbute-Peres et al., 2018; Karkus et al., 2019), improving some internal states but degrading others; we therefore analyze gradient alignment (Du et al., 2018; Yu et al., 2020) between terminal and internal losses.

Third, correction placement is coupled with simulator adaptation. The insertion position determines the correction's downstream influence and learning signal (Heiden et al., 2021; Wei et al., 2025), while joint adaptation changes internal states and sensitivities during training (Um et al., 2020; Kochkov et al., 2024). This coupling motivates studying both the effect of correction placement on the simulator, and how that effect changes when simulator parameters adapt accordingly.

To address these challenges, we formulate hybrid simulators as differentiable computation graphs, decompose each node error into propagated error and local mismatch, and trace both to the terminal output through edge-wise sensitivities. We then characterize when terminal training improves intermediate states through gradient alignment, and derive a criterion for correction placement. We evaluate terminal accuracy and intermediate states on a traffic simulator driven by real freeway detector data and on an industrial-scale ride-hailing simulator. Our contributions are threefold:

1. We establish a position-dependent analysis framework for differentiable simulation graphs and derive upper bounds on the downstream effect of local model–plant mismatch on terminal error.
2. We provide a sufficient gradient-alignment condition for improving an intermediate trainable node under terminal supervision.
3. We derive a turning-point criterion for correction placement in scaling-compatible chains, and show on two real-world simulators that an internally placed correction improves terminal accuracy while keeping intermediate states physically meaningful.

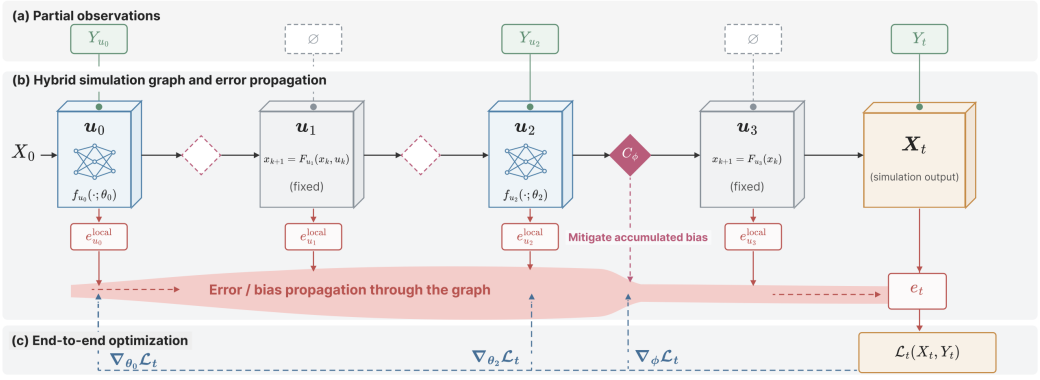


Figure 1: Overview of GraphBC

2 Simulation Graphs and Position-Dependent Error Propagation

2.1 Differentiable Computation Graph for Hybrid Simulators

We model a simulation pipeline as a directed acyclic computation graph (DAG) denoted by G:

$$G = (V, E, F) \tag{1}$$

where V denotes the set of computational nodes, and $E \subseteq V \times V$ denotes the set of directed edges, with $(v, u) \in E$ indicating that the output of node v serves as an input to node u. Moreover, $F = \{f_u : u \in V\}$ denotes the collection of node-wise computational mappings. Let $t \in V$ denote the designated terminal target node, whose state x_t is the final output of interest.

Specifically, for each non-source node u, f_u maps the states of its parent nodes to its state such that:

$$x_u = \begin{cases} f_u\left(x_{\mathrm{Pa}(u)}; \phi_u\right), & u \in V_{\mathrm{param}} \\ f_u\left(x_{\mathrm{Pa}(u)}\right), & u \in V_{\mathrm{fixed}} \end{cases} \tag{2}$$

where $\mathrm{Pa}(u) = \{v \in V : (v, u) \in E\}$ denotes the parent set of node u, $x_{\mathrm{Pa}(u)}$ collects the parent-node states, and ϕ_u denotes the trainable parameters of f_u. In this work, we assume that these mappings are (piecewise) differentiable to support end-to-end training (Belbute-Peres et al., 2018). For source nodes with $\mathrm{Pa}(u) = \varnothing$, x_u represents an exogenous input or initial condition.

To capture heterogeneous parameterization and partial observability, we characterize each node by its *parameterization* and *observability* status.

- **Parameterization.** The set V is partitioned into parameterized nodes V_{param} and fixed nodes V_{fixed}:

$$V = V_{\mathrm{param}} \cup V_{\mathrm{fixed}}, \qquad V_{\mathrm{param}} \cap V_{\mathrm{fixed}} = \varnothing \tag{3}$$

  where parameterized nodes include learned models or simulator components with trainable physical parameters, whereas fixed nodes contain no trainable parameters and only implement physical equations, deterministic rules, or other domain-specific mappings.
- **Observability.** Let y_u denote the reference state associated with node u, and define $V_{\mathrm{obs}} = \{u \in V : y_u \text{ is available}\}$ and $V_{\mathrm{unobs}} = V \setminus V_{\mathrm{obs}}$. Reference states can be obtained via empirical observations, high-fidelity simulations, or domain-validated intermediate states. It should be noted that these two attributes are independent, allowing all four combinations of parameterization and observability.

2.2 Local Model-Plant Mismatch and Error Propagation

We now analyze error propagation in G. For each node u, define its state error and magnitude as:

$$e_u = x_u - y_u, \qquad \epsilon_u = \|e_u\|_2 \tag{4}$$

where $\|\cdot\|_2$ denotes the Euclidean norm. Here, we analyze sample-wise discrepancies with fixed simulator parameters and omit ϕ_u for simplicity. Assuming that the reference process follows the same graph structure and each non-source node satisfies:

$$y_u = g_u\left(y_{\mathrm{Pa}(u)}\right) \tag{5}$$

where g_u denotes the ground-truth mapping (plants). The node-wise error e_u can be decomposed as:

$$\begin{aligned} x_u - y_u &= f_u\left(x_{\mathrm{Pa}(u)}\right) - g_u\left(y_{\mathrm{Pa}(u)}\right) \\ &= \underbrace{f_u\left(x_{\mathrm{Pa}(u)}\right) - f_u\left(y_{\mathrm{Pa}(u)}\right)}_{e_u^{\mathrm{up}}\text{: propagated upstream error}} + \underbrace{f_u\left(y_{\mathrm{Pa}(u)}\right) - g_u\left(y_{\mathrm{Pa}(u)}\right)}_{e_u^{\mathrm{local}}\text{: local model-plant mismatch}}. \end{aligned} \tag{6}$$

In Eq. (6), e_u^{up} denotes propagated upstream error, while e_u^{local} denotes the local model–plant mismatch at node u. The local term plays the role of the model-discrepancy function in calibration (Kennedy & O'Hagan, 2001), and is defined at every internal node rather than only at the terminal. These errors can accumulate through downstream computations and affect the terminal output.

Using the standard Lipschitz arguments, the following lemma bounds the accumulated error at the node, path, and DAG levels, with the proof provided in Appendix B.

Lemma 2.1 (Error propagation in a simulation graph)**.** *Let $L_{v\to u} \geq 0$ denote the edge-wise sensitivity bound associated with the directed edge $v \to u$, and let $W_{u\to t}$ denote the downstream sensitivity weight from node u to the terminal target t. Then the following bounds hold:*

$$\textbf{\textit{Node:}} \quad \epsilon_u \leq \sum_{v\in \mathrm{Pa}(u)} L_{v\to u}\epsilon_v + \epsilon_u^{\mathrm{local}}, \tag{7}$$

$$\textbf{\textit{Path:}} \quad \epsilon_{u_n} \leq \left(\prod_{j=1}^{n} L_{u_{j-1}\to u_j}\right)\epsilon_{u_0} + \sum_{k=1}^{n}\left(\prod_{j=k+1}^{n} L_{u_{j-1}\to u_j}\right)\epsilon_{u_k}^{\mathrm{local}}, \tag{8}$$

$$\textbf{\textit{DAG:}} \quad \epsilon_t \leq \sum_{s\in S} W_{s\to t}\epsilon_s + \sum_{u\in V\setminus S} W_{u\to t}\epsilon_u^{\mathrm{local}}. \tag{9}$$

where S denotes the set of source nodes. An empty product is defined as one.

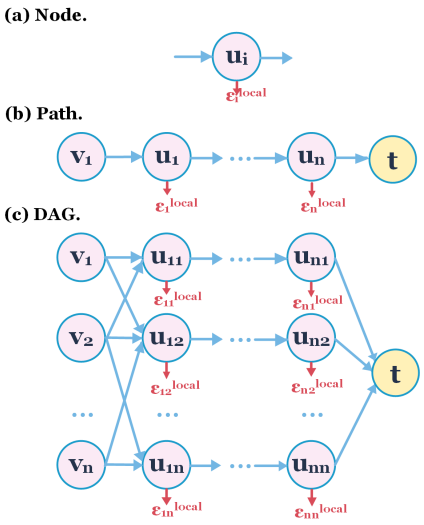


Figure 2: Illustration of node-, path-, and DAG-level error propagation.

Remark 2.1. Lemma 2.1 shows that the terminal error ϵ_t is bounded by a weighted accumulation of upstream errors and local model–plant mismatches governed by sensitivity $W_{i\to t}$. Thus, mismatches of similar magnitude can have substantially different downstream effects depending on their graph positions, motivating the analysis of component interactions and optimal correction placement.

3 Graph-Embedded Bias Correction with End-to-End Learning

Learned correction modules have been widely used to reduce discrepancies between simulator outputs and reference solutions by embedding correction functions into differentiable simulation pipelines (Um et al., 2020). Here, the correction node c_m is inserted at placement m to transform the original state x_m into the corrected state $\widetilde{x}_m$, which is then passed to downstream nodes. This yields the augmented acyclic graph G'_m as follows:

$$G'_m = (V'_m, E'_m, F'_m), \qquad V'_m = V \cup \{c_m\}, \tag{10}$$

while preserving the functional forms of the original mappings. Throughout the augmented graph G'_m, node states are denoted by $\widetilde{x}_u$. At the correction node c_m, the corrected state is generated by:

$$\widetilde{x}_m = \mathcal{P}_m\left(x_m, R_\theta(z_m)\right), \tag{11}$$

where R_θ is a learnable correction model, z_m denotes the contextual input features, and $\mathcal{P}_m$ is the insertion operator. In this work, we adopt the classic additive residual form (Yin et al., 2021):

$$\widetilde{x}_m = x_m + R_\theta(z_m). \tag{12}$$

The correction parameters θ are jointly optimized with the trainable simulator parameters Φ under end-to-end supervision, allowing correction to interact with the adaptation of trainable f_u.

For a fixed correction placement m, the sample-level terminal loss is:

$$L_t(\theta, \Phi; m) = \ell\left(\widetilde{x}_t(\theta, \Phi; m), y_t\right), \tag{13}$$

where $\widetilde{x}_t$ is the corrected terminal prediction, y_t is the corresponding reference target, and ℓ is the sample-level loss function. The training objective averages L_t over samples.

We next characterize how terminal supervision propagates through G'_m. At a differentiable operating point, we define the local edge Jacobian of each directed edge $v \to u$ and the ordered Jacobian product of each directed path $p = (u_0 = u, u_1, \ldots, u_q = t)$ from node u to the terminal t as

$$J_{uv} = \frac{\partial \widetilde{x}_u}{\partial \widetilde{x}_v}, \qquad J_p = J_{u_q u_{q-1}} \cdots J_{u_1 u_0}, \tag{14}$$

where J_{uv} is computed with the other parent states of u fixed, and J_p composes the edge Jacobians along p. Let $\mathcal{P}_{G'_m}(u,t)$ denote all directed paths from u to t. By the chain rule, as in reverse-mode differentiation on a DAG (Baydin et al., 2018), the total downstream Jacobian from u to t is:

$$A_{u\to t} = \sum_{p\in\mathcal{P}_{G'_m}(u,t)} J_p, \qquad A_{t\to t} = I, \tag{15}$$

where I denotes the identity matrix. Thus, $A_{u\to t}$ captures the total differential influence of node u on the terminal output through all directed paths. For a trainable node u with parameter block η_u, we define the local parameter Jacobian as follows:

$$B_{u,\eta_u} = \left.\frac{\partial \widetilde{x}_u}{\partial \eta_u}\right|_{\widetilde{x}_{\mathrm{Pa}(u)}}, \tag{16}$$

where the parent states are held fixed. Combining the local parameter effect with the downstream Jacobian, the following can be obtained:

$$\nabla_{\eta_u} L_t = \underbrace{B_{u,\eta_u}^\top}_{\text{local sensitivity}} \cdot \underbrace{A_{u\to t}^\top}_{\text{downstream propagation}} \cdot \underbrace{\nabla_{\widetilde{x}_t}\ell}_{\text{terminal signal}} \tag{17}$$

Here, $(\cdot)^\top$ denotes transpose, ∇ denotes the gradient operator, and $\nabla_{\widetilde{x}_t}\ell$ is the gradient of the sample-level loss with respect to the terminal state $\widetilde{x}_t$. Thus, terminal-driven updates depend on both local parameter sensitivity and downstream propagation. For the correction node, let $r_m = R_\theta(z_m)$. The generic gradient expression becomes:

$$\nabla_\theta L_t = B_{c_m,\theta}^\top A_{c_m\to t}^\top \nabla_{\widetilde{x}_t}\ell = \left(\frac{\partial R_\theta(z_m)}{\partial \theta}\right)^\top \left(\frac{\partial \mathcal{P}_m(x_m, r_m)}{\partial r_m}\right)^\top A_{c_m\to t}^\top \nabla_{\widetilde{x}_t}\ell, \tag{18}$$

Equations 15 and 17 give two necessary conditions for updating an intermediate trainable node.

Corollary 3.1 (Necessary conditions for successful terminal-driven updates)**.** *For terminal supervision to update the parameters η_u of a trainable node u, the following conditions are necessary:*

$$\textit{Condition 1: Structural reachability} \implies \mathcal{P}_{G'_m}(u,t) \neq \varnothing, \tag{19}$$

$$\textit{Condition 2: Active downstream path} \implies \exists\, p \in \mathcal{P}_{G'_m}(u,t) \textit{ such that } J_p \neq 0. \tag{20}$$

If either condition is violated, terminal supervision cannot update η_u, since

$$A_{u\to t} = 0, \qquad \nabla_{\eta_u} L_t = 0. \tag{21}$$

Corollary 3.1 shows that terminal-driven updates require structural reachability and at least one active downstream path. These conditions are necessary but not sufficient for nonzero gradients, since path cancellation, vanishing local sensitivity, or gradient orthogonality may still yield $\nabla_{\eta_u} L_t = 0$.

Open issues: Although the end-to-end training mechanism is now characterized, two issues remain unresolved. First, it remains unclear under what conditions terminal-driven updates improve or degrade the local prediction at an intermediate trainable node. Second, it is still unknown how correction placement affects the interaction between correction and differentiable simulator adaptation.

4 Theoretical Analysis of Correction Interaction and Placement

4.1 Gradient Alignment for Intermediate-State Adaptation

The effect of terminal-driven updates on intermediate predictions depends on their interaction with local errors. We therefore fix the correction parameters θ and placement m, and analyze one update of the trainable simulator parameters Φ, writing $L_t(\Phi) := L_t(\theta, \Phi; m)$.

Let $\mathcal{O}$ denote the set of observed internal nodes, excluding the terminal target. We define the aggregate internal-state loss as:

$$L_{\mathrm{int}}(\Phi) = \frac{1}{|\mathcal{O}|}\sum_{u\in\mathcal{O}} \|\widetilde{x}_u(\Phi) - y_u\|_2^2. \tag{22}$$

The following proposition establishes when a terminal signal also improves internal-state prediction, and its proof is provided in Appendix C.

Proposition 4.1 (Internal-state improvement under gradient alignment)**.** *Assume that L_t and L_{int} are differentiable with respect to Φ, and that L_{int} is κ-smooth, with $\kappa > 0$. For nonzero gradients, define the gradient alignment as*

$$\gamma = \frac{\langle \nabla_\Phi L_{\text{int}}, \nabla_\Phi L_t \rangle}{\|\nabla_\Phi L_{\text{int}}\|_2 \|\nabla_\Phi L_t\|_2}. \tag{23}$$

Consider one terminal-gradient update $\Phi' = \Phi - \alpha \nabla_\Phi L_t(\Phi)$ with step size $\alpha > 0$. If the gradients are positively aligned and the step size is sufficiently small such that:

$$\gamma > 0, \qquad 0 < \alpha < \frac{2\gamma \|\nabla_\Phi L_{\text{int}}\|_2}{\kappa \|\nabla_\Phi L_t\|_2}, \tag{24}$$

then the terminal-driven update also improves the internal-state prediction:

$$L_{\text{int}}(\Phi') < L_{\text{int}}(\Phi). \tag{25}$$

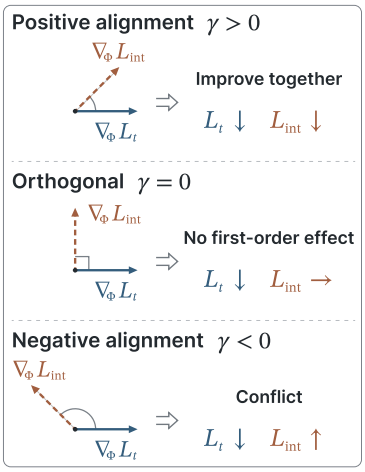


Figure 3: Gradient alignment

Thus, positive gradient alignment provides a **sufficient condition** for terminal-driven updates to improve internal-state prediction under a sufficiently small step size.

4.2 Optimal Correction Placement

Placement determines how a correction propagates downstream. For a small perturbation $\delta\widetilde{x}_m$ to the corrected state, first-order expansion of the downstream mapping gives

$$\delta\widetilde{x}_t = A_{c_m \to t} \delta\widetilde{x}_m + o\left(\|\delta\widetilde{x}_m\|_2\right), \tag{26}$$

where $o(\|\delta\widetilde{x}_m\|_2)$ collects higher-order terms. Since $A_{c_m \to t}$ varies with the correction position m, the terminal effect of a local correction is position dependent.

We therefore formulate correction placement as the discrete optimization problem as follows:

$$m^\star = \arg\min_{m \in \mathcal{M}} L_{\text{place}}(m), \tag{27}$$

where $\mathcal{M}$ denotes the set of admissible correction positions. We define the placement objective as:

$$L_{\text{place}}(m) = \eta \left\| \widetilde{x}_t^{(m)} - y_t \right\|_2^2 + \sum_{u \in \mathcal{O}} \lambda_u \left\| \widetilde{x}_u^{(m)} - y_u \right\|_2^2, \tag{28}$$

where $\eta > 0$ and $\lambda_u \geq 0$ are weighting coefficients, and $\widetilde{x}_u^{(m)}$ denotes the state at node u when the correction is placed at m. The following proposition gives an explicit solution to this placement problem for a scaling-compatible scalar chain, with its proof and analysis provided in Appendix D.

Proposition 4.2 (Placement turning point in a scaling-compatible chain)**.** *Consider a scalar chain $x_i = f_i(x_{i-1})$, $i = 1, \ldots, n$, with a fixed input, fixed simulator parameters, reference states y_i, and $x_n \neq 0$. Let $b = y_n / x_n > 0$ denote the scaling factor that matches the terminal state, and assume that $f_i(bx) = b f_i(x)$ on the relevant states. For $i = 1, \ldots, n-1$, we define marginal correction effect as:*

$$\Delta_i = \lambda_i \left[(bx_i - y_i)^2 - (x_i - y_i)^2 \right], \tag{29}$$

where Δ_i measures the change in the weighted error at node i after applying the scaling correction.
If there exists $q \in \{1, \ldots, n-1\}$ such that $\Delta_i \geq 0$ for $i < q$ and $\Delta_i < 0$ for $i \geq q$, then the optimal correction position is

$$m^\star = q. \tag{30}$$

If $\Delta_i \geq 0$ for all $i = 1, \ldots, n-1$, then $m^\star = n$.

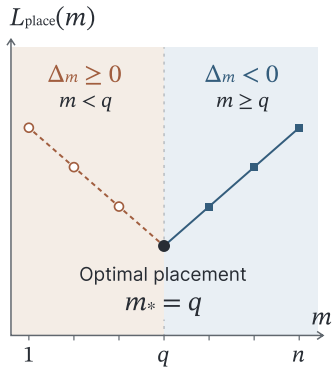


Figure 4: Illustration of the placement turning point.

Table 2: Case Study 1: Internal v.s. terminal placement under terminal-loss training.

Injected Bias Level	Method	**Internal Placement (C_9)**			**Terminal Placement (C_{13})**		
		Terminal RMSE (veh/h)	Internal RMSE (veh/h)	Parameter Shift vs. Local (%)	Terminal RMSE (veh/h)	Internal RMSE (veh/h)	Parameter Shift vs. Local (%)
25%	Local	657.29	569.07	0.00	657.29	569.07	0.00
	Correction Only	$333.09_{\pm 1.98}$	$520.37_{\pm 1.16}$	0.00	$328.18_{\pm 0.52}$	$566.89_{\pm 0.15}$	0.00
	E2E Only	393.24	**431.24**	17.05	393.24	**431.24**	17.05
	GraphBC	$\mathbf{280.13}_{\pm \mathbf{0.64}}$	$448.78_{\pm 1.30}$	$\mathbf{10.66}_{\pm \mathbf{0.32}}$	$\mathbf{282.17}_{\pm \mathbf{3.34}}$	$447.46_{\pm 1.79}$	$\mathbf{9.10}_{\pm \mathbf{0.66}}$
50%	Local	715.11	584.24	0.00	715.11	584.24	0.00
	Correction Only	$334.33_{\pm 0.79}$	$520.62_{\pm 1.27}$	0.00	$331.23_{\pm 0.44}$	$581.83_{\pm 0.13}$	0.00
	E2E Only	448.99	**440.61**	16.16	448.99	**440.61**	16.16
	GraphBC	$\mathbf{286.54}_{\pm \mathbf{0.89}}$	$449.80_{\pm 1.16}$	$\mathbf{10.16}_{\pm \mathbf{0.29}}$	$\mathbf{290.60}_{\pm \mathbf{8.59}}$	$458.66_{\pm 7.21}$	$\mathbf{9.07}_{\pm \mathbf{0.12}}$
75%	Local	774.42	601.83	0.00	774.42	601.83	0.00
	Correction Only	$338.47_{\pm 1.56}$	$520.37_{\pm 1.10}$	0.00	$335.74_{\pm 0.85}$	$599.43_{\pm 0.14}$	0.00
	E2E Only	517.93	451.72	17.09	517.93	**451.72**	17.09
	GraphBC	$\mathbf{293.91}_{\pm \mathbf{4.06}}$	$\mathbf{450.14}_{\pm \mathbf{1.19}}$	$\mathbf{10.99}_{\pm \mathbf{1.16}}$	$\mathbf{295.69}_{\pm \mathbf{4.00}}$	$472.12_{\pm 6.05}$	$\mathbf{9.43}_{\pm \mathbf{0.93}}$

Note: Best results are shown in **bold**, and second-best results are underlined. For Parameter Shift, ranking is reported among methods that update simulator parameters (E2E Only and GraphBC).

Remark 4.1. *In Proposition 4.2, the correction at position m is $\widetilde{x}_m^{(m)} = bx_m$, a special case of the additive form in Eq. 12 with $R_\theta(z_m) = (b-1)x_m$. By scaling compatibility, $\widetilde{x}_n^{(m)} = y_n$ holds for every m, so all placements match the terminal target equally well and the choice of m is determined solely by the internal-state errors.*

Proposition 4.2 shows that the optimal position occurs where the marginal effect of moving the correction downstream changes sign. Specifically,

$$L_{\text{place}}(m+1) - L_{\text{place}}(m) = -\Delta_m. \tag{31}$$

Thus, moving the correction downstream decreases the placement objective when $\Delta_m > 0$, but increases it when $\Delta_m < 0$. The turning point therefore separates positions where applying the correction is locally harmful from those where it is locally beneficial.

5 Case Study

Experiment scenarios. We evaluate GraphBC on two simulators driven by real-world data. Case Study 1 builds a cell transmission model (CTM; Daganzo, 1994) of a freeway corridor from loop-detector measurements. Because its chain-structured graph admits a correction at every cell, we use it to study correction placement, together with gradient alignment. Case Study 2 uses an industrial-scale citywide ride-hailing simulator with market data from 40 cities to study how correction affects intermediate and terminal states. Full experimental details are provided in Appendix E.

Compared methods. We compare four configurations that form a 2×2 design over how the simulator modules are calibrated and whether a correction module is added. **Local** calibrates each trainable module independently to its own reference states, without correction. **Correction Only** starts from the calibrated simulator, keeps all modules fixed, and trains only the correction module through the differentiable simulator (Um et al., 2020). **E2E Only** trains all trainable modules jointly under end-to-end supervision, with gradients propagated through the entire simulation graph, but without correction (Belbute-Peres et al., 2018). **GraphBC** inserts the correction module at the selected position and trains it jointly with all trainable modules end-to-end.

5.1 Case Study 1: Chain-Structured Freeway Traffic Simulation

GraphBC achieves the best accuracy–fidelity trade-off. In Table 2, with the correction at C_9, GraphBC attains the lowest terminal error at every bias level, 29–43% below E2E Only, while keeping its internal error within about 4% of the best method. It also shifts the fundamental-diagram parameters by only 10.2–11.0%, versus 16.2–17.1% for E2E Only, as the correction absorbs much of the mismatch. Moving the correction to C_{13} leaves the terminal error nearly unchanged and reduces the parameter shift slightly (9.1–9.4%), but leaves the deficit from C_8 uncorrected in the intermediate cells: the internal error rises to 458.66 and 472.12 veh/h at 50% and 75% bias, versus 449.80 and 450.14 veh/h at C_9. Since the internal error is measured directly against detector data, whereas parameter shift is only a proxy, C_9 is the better placement when intermediate states matter.

Internal changes corroborate gradient alignment. As shown in Fig. 5, GraphBC sharply reduces the error at the terminal cell and at three of the five intermediate cells, while the other two rise only slightly. These outcomes are consistent with the sign of the gradient cosine between each cell's

Figure 7: Simulation graph of Case Study 2. ML: trainable module; Fixed: deterministic node; BC: correction node.

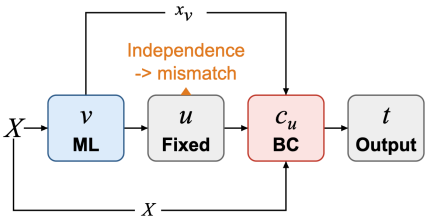


Table 3: Case Study 2 test results, averaged over three seeds. State MSE ($\times 10^{-4}$) of the uncorrected x_u and of x_t; output MAPE (%) of three outputs of v and of t.

	MSE ($\times 10^{-4}$)		MAPE (%)			
Method	x_u	x_t	v_1	v_2	v_3	t
E2E Only	154.92	83.97	17.00	12.30	43.24	13.62
GraphBC	**61.65**	**4.20**	**7.95**	**5.50**	**16.55**	**4.40**
Reduction	60.2%	95.0%	53.2%	55.3%	61.7%	67.7%

loss and the terminal loss, computed at the Local parameters before training with the correction weights fixed: every improved cell is positively aligned with the terminal loss and every other cell is negatively aligned. Although Proposition 4.1 is a one-step result, the alignment at the starting point thus anticipates which intermediate states improve after training.

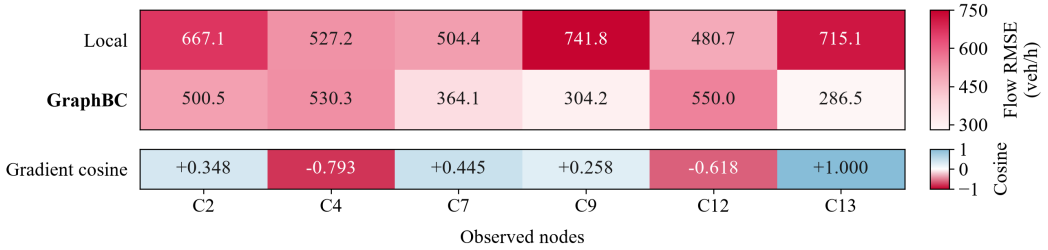


Figure 5: Flow RMSE of Local and GraphBC (top) and gradient cosine with the terminal loss at the Local parameters (bottom). Improved cells are exactly those with positive alignment.

Placement matters less when the simulator adapts. Fig. 6 sweeps the correction over all 13 cells. Correction Only has almost no effect at C_1–C_5, becomes effective from C_6 onward near the biased on-ramp at C_8, and outperforms E2E Only from C_7 onward, showing that placement is critical when the simulator cannot adapt. In contrast, GraphBC consistently stays well below E2E Only at every position, attains the lowest error throughout with a much flatter profile, and reaches its minimum near C_9, just downstream of the bias. Joint adaptation thus makes the correction robust to its placement, while the best position still lies close to the source of the mismatch, indicating that placement and simulator adaptation should be treated jointly. In practice, placing the correction just downstream of the mismatch source is therefore a reliable choice.

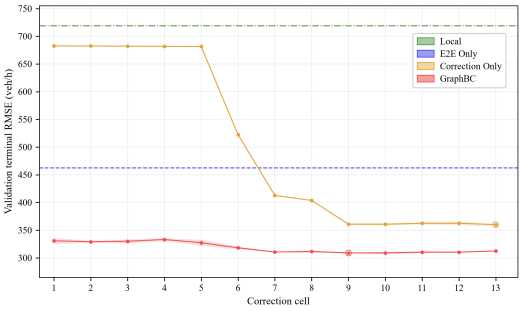


Figure 6: Terminal RMSE with correction placed at each cell (50% bias, terminal-loss training, mean ± std over three seeds). Dashed lines show Local and E2E Only, which use no correction. Circles mark the best position of each method.

5.2 Case Study 2: Hybrid Citywide Ride-Hailing Simulation

The simulator predicts city-level request outcomes from market features (Fig. 7; Subsection Appendix E.2). A trainable module v predicts the probabilities of several component outcomes, a deterministic node u combines them into probabilities of joint outcomes under a conditional-independence assumption, and the terminal node t aggregates the joint outcomes. As this assumption is the main mismatch source, we place the correction at $m = u$, with $\widetilde{x}_u = x_u + R_\theta(z_u)$ and $z_u = (X, x_v)$ for market features X. We compare GraphBC with E2E Only, the training scheme used for this simulator in practice. Since the component- and joint-level outputs also inform downstream decisions, all three levels are supervised jointly via Eq. 65. E2E Only is thus a strong baseline for intermediate-state accuracy, as it directly supervises x_v and x_u. GraphBC uses the identical loss and differs only in the added correction node.

GraphBC improves intermediate and terminal states while preserving the upstream prior. In Table 3, GraphBC reduces the terminal MSE by 95% and the terminal MAPE by 68% relative to E2E Only. The intermediate states improve as well: the MAPE of every reported output of v decreases by 53–62%, and the MSE of x_u decreases by 60% even though u has no trainable parameters and, under GraphBC, is not directly supervised. Since x_u is fully determined by x_v, this indicates that the upstream module becomes more accurate. Without the correction, v can reduce the downstream error only through u, and is thus pushed to compensate for the structural bias of the independence assumption; with the correction absorbing this bias, v is freed to fit its own targets while its structured prior, which fixes the direction of each response, is retained by construction. As in Case Study 1, the terminal output becomes accurate while the intermediate states remain physically meaningful.

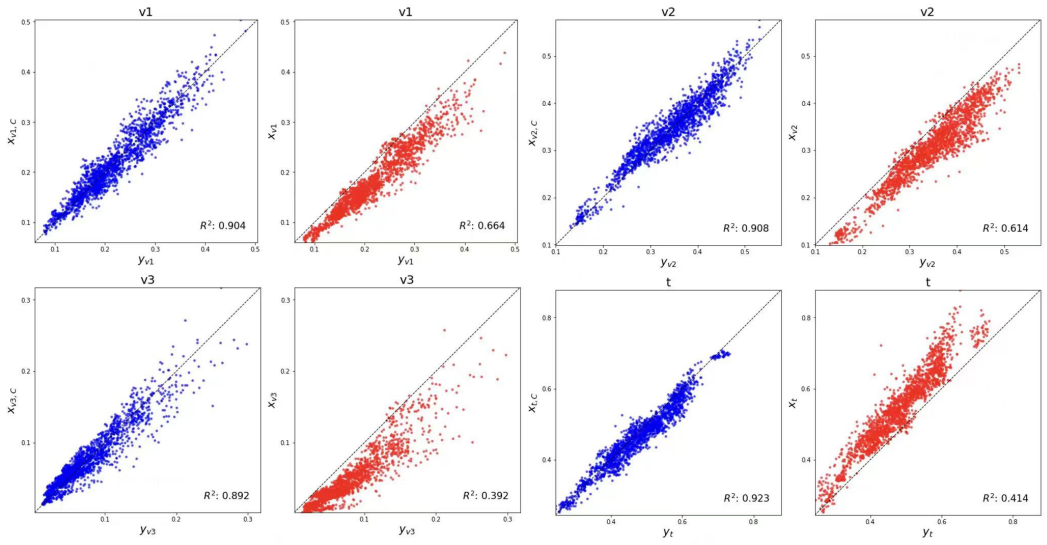


Figure 8: Predictions x versus ground truth y on the test set for three outputs of v and for terminal node t. Blue panels (subscript C): GraphBC; red panels: E2E Only. Dashed line: $y = x$.

GraphBC restores the physical meaning of the upstream module. Figure 8 plots predictions against ground truth. With the correction, both v and t lie close to the diagonal ($R^2 \geq 0.89$), so the upstream module recovers its physical meaning while the terminal output becomes accurate, as in Case Study 1. In contrast, under E2E Only the outputs of v are systematically underestimated while the terminal output remains overestimated (R^2 of 0.39–0.66 for v and 0.41 for t), consistent with v being pushed away from its targets to offset the bias of the independence assumption.

6 Conclusion

This paper presented GraphBC, a general framework for graph-embedded bias correction in differentiable simulators under model–plant mismatch. We represented hybrid simulators as differentiable computation graphs and analyzed how local errors propagate through downstream components. We further characterized the conditions under which terminal-driven optimization improves intermediate states through gradient alignment, and derived a turning-point criterion for correction placement in a scaling-compatible chain. Based on this analysis, GraphBC inserts a learnable correction node at an admissible graph location and jointly optimizes the correction module and causally connected trainable simulator components under end-to-end supervision. Experiments on a traffic-flow simulator and an industrial-scale ride-hailing transaction simulator demonstrate that placement is critical when the simulator cannot adapt, while joint adaptation makes the correction robust to placement.

Several directions remain for future work. **Placement generalization** could extend the current optimal-placement analysis from scaling-compatible scalar chains to general nonlinear and branched simulation graphs. **Dynamic placement** could adapt the correction location to different operating regimes, input states, or evolving system conditions. **Sparse observability** could be addressed through physical constraints, latent-state estimation, and uncertainty-aware objectives when intermediate observations are limited or unavailable.

REPRODUCIBILITY STATEMENT

Sections 2–4 specify the simulation-graph formulation, GraphBC, and the assumptions of our theoretical results, and Appendices B–D provide the complete proofs. Appendix E.1 documents Case Study 1 in full, including the public data source (Caltrans PeMS), detector-to-cell mapping, data preprocessing and splits, simulator implementation, local calibration, correction architecture, training settings, random seeds, and evaluation metrics. We provide anonymous supplementary material containing the code for the differentiable CTM simulator, GraphBC (Algorithm 1), and all compared configurations (Local, Correction Only, and E2E Only), together with scripts for data retrieval and preprocessing and for reproducing Table 2 and Figures 5, 6, and 10–12. Its README file lists the dependencies, data links, and commands. The code and processed data will be publicly released upon publication.

Case Study 2 is based on an industrial ride-hailing simulator and proprietary city-level market data provided by an industrial partner. Because of confidentiality agreements, we are unable to release the data or the simulator. To support transparency, Appendix E.2 describes the simulation graph, the source of model–plant mismatch, the placement and architecture of the correction node, the training objective, the data split, and the evaluation metrics. The GraphBC correction and training procedure are simulator-agnostic, and the same implementation as in the released code is used in both case studies. The released Case Study 1 therefore allows readers to verify the method itself on public data.

AI USE STATEMENT

Large language models (LLMs) were used in a limited, assistive capacity. First, they assisted in preparing several schematic illustrations of the framework (e.g., Figure 1), based on sketches and content specified by the authors; the authors checked and revised every figure for accuracy. Second, they were used to help read and summarize related literature; the authors read the original papers, verified all summaries and claims, and selected and added the references themselves. Third, they were used for proofreading, including grammar, spelling, and wording, after the authors had drafted the manuscript. LLMs were not used to formulate the research problem, develop the method, derive the theoretical results, write the code, conduct the experiments, or analyze the results. The authors take full responsibility for all content of this paper and its supplementary material.

REFERENCES

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Joshua B. Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Peter Berg, H Feldmann, and H-J Panitz. Bias correction of high resolution regional climate model data. *Journal of Hydrology*, 448:80–92, 2012.

Jenný Brynjarsdóttir and Anthony O'Hagan. Learning about physical parameters: The importance of model discrepancy. *Inverse problems*, 30(11):114007, 2014.

Chao Chen, Karl Petty, Alexander Skabardonis, Pravin Varaiya, and Zhanfeng Jia. Freeway performance measurement system: Mining loop detector data. *Transportation Research Record*, 1748(1): 96–102, 2001.

Carlos F. Daganzo. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B: Methodological*, 28(4): 269–287, 1994.

Vu Dao, Claudia Jimenez Arellano, Phu Nguyen, Fahad Almutlaq, Kuolin Hsu, and Soroosh Sorooshian. Bias correction of satellite precipitation estimation using deep neural networks

and topographic information over the western us. *Journal of Geophysical Research: Atmospheres*, 130(4):e2024JD042181, 2025.

Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Mehrdad Farajtabar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

Katharina Ensinger, Sebastian Ziesche, and Sebastian Trimpe. Learning hybrid dynamics models with simulator-informed latent states. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 11892–11900, 2024. doi: 10.1609/aaai.v38i11.29075.

L Gudmundsson, JB Bremnes, JE Haugen, and T Engen Skaugen. Downscaling rcm precipitation to the station scale using quantile mapping–a comparison of methods. *Hydrology & Earth System Sciences Discussions*, 9(5):6185–6201, 2012.

Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9474–9481. IEEE, 2021.

Peter Karkus, Xiao Ma, David Hsu, Leslie Pack Kaelbling, Wee Sun Lee, and Tomás Lozano-Pérez. Differentiable algorithm networks for composable robot learning. *arXiv preprint arXiv:1905.11602*, 2019.

Marc C Kennedy and Anthony O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

Dmitrii Kochkov, Janni Yuval, Ian Langmore, Peter Norgaard, Jamie Smith, Griffin Mooers, Milan Klöwer, James Lottes, Stephan Rasp, Peter Düben, et al. Neural general circulation models for weather and climate. *Nature*, 632(8027):1060–1066, 2024.

Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. Differentiability in unrolled training of neural physics simulators on transient dynamics. *Computer Methods in Applied Mechanics and Engineering*, 433:117441, 2025.

Douglas Maraun. Bias correcting climate change simulations-a critical review. *Current Climate Change Reports*, 2(4):211–220, 2016.

Elvis Nava, John Z. Zhang, Mike Yan Michelis, Tao Du, Pingchuan Ma, Benjamin F. Grewe, Wojciech Matusik, and Robert Kevin Katzschmann. Fast aquatic swimmer optimization with differentiable projective dynamics and neural network hydrodynamic models. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 16413–16427, 2022.

Cyrus Neary and Ufuk Topcu. Compositional learning of dynamical system models using port-hamiltonian neural networks. In *Learning for Dynamics and Control Conference*, pp. 679–691. PMLR, 2023.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013.

Matthew Plumlee. Bayesian calibration of inexact computer models. *Journal of the American Statistical Association*, 112(519):1274–1285, 2017.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning, 2020.

Sanghyun Son, Yi-Ling Qiao, Jason Sewall, and Ming C. Lin. Differentiable hybrid traffic simulation. *ACM Transactions on Graphics*, 41(6):1–10, 2022.

Bharat Srikishan, Daniel O'Malley, Mohamed Mehana, Nicholas Lubbers, and Nikhil Muralidhar. Model-agnostic knowledge guided correction for improved neural surrogate rollout. In *International Conference on Learning Representations*, 2025.

Naoya Takeishi and Alexandros Kalousis. Physics-integrated variational autoencoders for robust and interpretable generative modeling. *Advances in Neural Information Processing Systems*, 34: 14809–14821, 2021.

Rui Tuo and CF Jeff Wu. Efficient calibration for imperfect computer models. 2015.

Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Advances in Neural Information Processing Systems*, 2020.

Hao Wei, Aleksandra Franz, Björn List, and Nils Thuerey. Inc: An indirect neural corrector for auto-regressive hybrid pde solvers. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating scientific knowledge with machine learning for engineering and environmental systems. *ACM Computing Surveys*, 55(4):1–37, 2022.

Yuan Yin, Vincent Le Guen, Jérémie Dona, Emmanuel de Bézenac, Ibrahim Ayed, Nicolas Thome, and Patrick Gallinari. Augmenting physical models with deep networks for complex dynamics forecasting. In *International Conference on Learning Representations*, 2021.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in neural information processing systems*, 33: 5824–5836, 2020.

Morteza Zarei, Mohsen Najarchi, and Reza Mastouri. Bias correction of global ensemble precipitation forecasts by random forest method. *Earth Science Informatics*, 14(2):677–689, 2021.

APPENDIX A DETAILED LITERATURE REVIEW

We organize and review the prior work by where the bias correction acts: at the simulator output, inside a single model, or through a solver's rollout.

OUTPUT-LEVEL POST-HOC CORRECTION AND CALIBRATION

Statistical calibration and machine-learning methods have been widely used to reduce systematic discrepancies in simulation and forecasting outputs, including scaling, distribution mapping, random forests, and neural correction models (Berg et al., 2012; Maraun, 2016; Gudmundsson et al., 2012; Zarei et al., 2021; Dao et al., 2025). These post-hoc approaches learn a transformation from an original prediction to a calibrated output while the simulator is held fixed, so the internal trajectory and its errors remain unchanged. Computer-model calibration goes further by inferring an additive discrepancy function jointly with the simulator parameters (Kennedy & O'Hagan, 2001). This line of work also showed that joint fitting can bias the inferred physical parameters (Brynjarsdóttir & O'Hagan, 2014), and proposed loss-based definitions of the optimal parameter (Tuo & Wu, 2015) and orthogonality constraints (Plumlee, 2017) to keep them meaningful, which is the same tension that motivates our dilemma. In both cases, however, the correction is attached at the output of a single simulator.

HYBRID MODELS AND NEURAL-AUGMENTED SIMULATORS

Hybrid modeling moves the correction inside the model by combining known scientific structure with trainable components that represent unresolved dynamics (Willard et al., 2022). Universal differential equations, learned discretizations, and neural-augmented physics engines incorporate learned components into analytical and numerical simulators (Rackauckas et al., 2020; Bar-Sinai et al., 2019; Heiden et al., 2021), and end-to-end differentiable physics enables gradient-based learning through them (Belbute-Peres et al., 2018; Nava et al., 2022). NeuralSim further investigates how to discover useful augmentations within a differentiable physics engine (Heiden et al., 2021). Several works address the balance between physical and learned components: APHYNITY chooses a minimum-norm residual so that the physical model explains as much of the data as possible (Yin et al., 2021), physics-integrated variational autoencoders regularize the learned part so that the physics part is not bypassed (Takeishi & Kalousis, 2021), and simulator-informed latent states limit error accumulation (Ensinger et al., 2024). Compositional approaches learn subsystem models that are assembled into larger systems (Neary & Topcu, 2023; Karkus et al., 2019). These methods typically consider a single dynamical model or a fixed augmentation location.

SOLVER-IN-THE-LOOP CORRECTION AND DIFFERENTIABLE UNROLLING

A third line of work trains corrections through recurrent numerical solvers using rollout-level objectives. Solver-in-the-Loop shows that exposing a correction model to trajectories affected by its earlier corrections improves rollout accuracy compared with purely local training (Um et al., 2020). Subsequent work distinguishes the effects of training-distribution shift and temporal differentiation in unrolled training (List et al., 2025), corrects the governing equations instead of the state to reduce error amplification (Wei et al., 2025), and uses knowledge-guided corrections to stabilize neural surrogate rollouts (Srikishan et al., 2025). Related differentiable traffic simulators demonstrate gradient-based learning and control through coupled macroscopic and microscopic dynamics (Son et al., 2022). These works study how corrections are trained over time within a single solver.

RESEARCH GAP AND OUR POSITIONING

Taken together, prior work has made substantial progress on correcting simulator outputs, preserving physical parameters in a single dynamical model, and training corrections through differentiable solvers (Table 1). Less attention has been paid to multi-stage hybrid simulators whose modules are heterogeneous, only partially trainable, and only partially observed. In this setting, a correction can be inserted at many internal positions, and its effect depends on how errors propagate through the graph and on which other modules adapt during training. GraphBC addresses this gap in three ways: it separates local mismatch from propagated error and quantifies the downstream influence of each graph position; it characterizes when terminal supervision also improves intermediate states, extending gradient-alignment analyses from multi-objective learning (Du et al., 2018; Yu et al., 2020) to simulation graphs; and it derives a placement criterion for scaling-compatible chains and empirically examines how simulator adaptation affects placement sensitivity. Across a traffic flow

simulator and an industrial-scale ride-hailing simulator, these analyses show that an internal correction can make the terminal output accurate while keeping intermediate states and parameters physically meaningful.

APPENDIX B PROOF OF LEMMA 2.1

To characterize error propagation, we assume that each node mapping f is locally Lipschitz with respect to its parent states. Specifically, for each edge $v \to u$, let $L_{v\to u} \geq 0$ denote an edge-wise local sensitivity bound such that

$$\left\| f_u\big(x_{\mathrm{Pa}(u)}\big) - f_u\big(y_{\mathrm{Pa}(u)}\big) \right\|_2 \leq \sum_{v\in \mathrm{Pa}(u)} L_{v\to u}\epsilon_v \tag{32}$$

To capture the cumulative downstream effect of these local sensitivities, we define the downstream sensitivity weight $W_{i\to t}$ as follows:

$$W_{i\to t} = \sum_{p\in\mathcal{P}_G(i,t)} \prod_{(v\to u)\in p} L_{v\to u} \tag{33}$$

where $\mathcal{P}_G(i,t)$ denotes the set of all directed paths from node i to the terminal target t in graph G. In particular, we set $W_{t\to t} = 1$ and $W_{i\to t} = 0$ if t is unreachable from i.

Here, $L_{v\to u}$ characterizes the local one-step sensitivity along edge $v \to u$, whereas $W_{i\to t}$ aggregates these edge-wise sensitivities over all downstream paths from node i to terminal t, thereby quantifying the potential cumulative amplification of an error introduced at node i.

With the above sensitivity measures defined, the error at each node u can be decomposed as:

$$e_u = e_u^{\mathrm{up}} + e_u^{\mathrm{local}}, \qquad \epsilon_u^{\mathrm{local}} = \|e_u^{\mathrm{local}}\|_2. \tag{34}$$

By the Cauchy–Schwarz inequality, we can obtain:

$$\left\langle e_u^{\mathrm{up}}, e_u^{\mathrm{local}} \right\rangle \leq \|e_u^{\mathrm{up}}\|_2 \|e_u^{\mathrm{local}}\|_2. \tag{35}$$

Therefore,

$$\begin{aligned} \|e_u\|_2^2 &= \|e_u^{\mathrm{up}} + e_u^{\mathrm{local}}\|_2^2 \\ &= \|e_u^{\mathrm{up}}\|_2^2 + 2\left\langle e_u^{\mathrm{up}}, e_u^{\mathrm{local}} \right\rangle + \|e_u^{\mathrm{local}}\|_2^2 \\ &\leq \|e_u^{\mathrm{up}}\|_2^2 + 2\|e_u^{\mathrm{up}}\|_2\|e_u^{\mathrm{local}}\|_2 + \|e_u^{\mathrm{local}}\|_2^2 \\ &= \left(\|e_u^{\mathrm{up}}\|_2 + \|e_u^{\mathrm{local}}\|_2\right)^2. \end{aligned} \tag{36}$$

Taking nonnegative square roots on both sides gives:

$$\epsilon_u = \|e_u\|_2 \leq \|e_u^{\mathrm{up}}\|_2 + \epsilon_u^{\mathrm{local}}. \tag{37}$$

By the edge-wise local sensitivity bound, we can obtain:

$$\|e_u^{\mathrm{up}}\|_2 = \left\| f_u\big(x_{\mathrm{Pa}(u)}\big) - f_u\big(y_{\mathrm{Pa}(u)}\big) \right\|_2 \leq \sum_{v\in \mathrm{Pa}(u)} L_{v\to u}\epsilon_v. \tag{38}$$

Substituting the bound in Eq. 38 into the preceding inequality Eq. 37 yields:

$$\epsilon_u \leq \sum_{v\in \mathrm{Pa}(u)} L_{v\to u}\epsilon_v + \epsilon_u^{\mathrm{local}}, \tag{39}$$

which establishes the node-level error recurrence as presented in Eq. 7.

Recursively applying the node-level bound along a directed path yields the path-level bound in Eq. 8, while recursive substitution over the DAG in topological order yields the DAG-level bound in Eq. 9, where $W_{i\to t}$ aggregates products of edge-wise sensitivity bounds over all directed paths from i to t.

APPENDIX C PROOF OF PROPOSITION 4.1

In this appendix, we provide the proof of Proposition 4.1, together with additional analysis of its gradient-alignment condition.

Proof. By the κ-smoothness of L_{int} and the standard descent lemma (Nesterov, 2013), we have:

$$\begin{aligned} L_{\text{int}}(\Phi') &\le L_{\text{int}}(\Phi) + \langle \nabla_\Phi L_{\text{int}}, \Phi' - \Phi \rangle + \frac{\kappa}{2} \|\Phi' - \Phi\|_2^2 \\ &= L_{\text{int}}(\Phi) - \alpha \langle \nabla_\Phi L_{\text{int}}, \nabla_\Phi L_t \rangle + \frac{\kappa \alpha^2}{2} \|\nabla_\Phi L_t\|_2^2 , \end{aligned} \tag{40}$$

where the second line follows from the terminal-gradient update $\Phi' = \Phi - \alpha \nabla_\Phi L_t(\Phi)$.

By the definition of the gradient alignment γ,

$$\langle \nabla_\Phi L_{\text{int}}, \nabla_\Phi L_t \rangle = \gamma \|\nabla_\Phi L_{\text{int}}\|_2 \|\nabla_\Phi L_t\|_2 . \tag{41}$$

Substituting this relation into Eq. 40 gives

$$\begin{aligned} L_{\text{int}}(\Phi') - L_{\text{int}}(\Phi) &\le -\alpha\gamma \|\nabla_\Phi L_{\text{int}}\|_2 \|\nabla_\Phi L_t\|_2 + \frac{\kappa \alpha^2}{2} \|\nabla_\Phi L_t\|_2^2 \\ &= \alpha \|\nabla_\Phi L_t\|_2 \left(-\gamma \|\nabla_\Phi L_{\text{int}}\|_2 + \frac{\kappa \alpha}{2} \|\nabla_\Phi L_t\|_2 \right). \end{aligned} \tag{42}$$

Under the conditions

$$\gamma > 0, \qquad 0 < \alpha < \frac{2\gamma \|\nabla_\Phi L_{\text{int}}\|_2}{\kappa \|\nabla_\Phi L_t\|_2}, \tag{43}$$

the term in parentheses is strictly negative. Therefore,

$$L_{\text{int}}(\Phi') < L_{\text{int}}(\Phi), \tag{44}$$

which proves the result. □

APPENDIX D PROOF OF PROPOSITION 4.2

In this appendix, we provide the proof of Proposition 4.2, together with additional analysis of the placement turning-point condition.

Proof. For the scalar chain in Proposition 4.2, place the scaling correction at position m as $\widetilde{x}_m = bx_m$. Since the downstream mappings satisfy $f_i(bx) = bf_i(x)$, repeated application of the scaling-compatibility condition gives

$$\widetilde{x}_i^{(m)} = \begin{cases} x_i, & i < m, \\ bx_i, & i \ge m. \end{cases} \tag{45}$$

Because $b = y_n / x_n$, we have $\widetilde{x}_n^{(m)} = bx_n = y_n$ for every correction position m. Hence, the terminal term in Eq. 28 is identical across placements, and the placement objective reduces to

$$L_{\text{place}}(m) = \sum_{i=1}^{n-1} \lambda_i \left(\widetilde{x}_i^{(m)} - y_i \right)^2 . \tag{46}$$

Using Eq. 45, we have:

$$L_{\text{place}}(m) = \sum_{i=1}^{m-1} \lambda_i (x_i - y_i)^2 + \sum_{i=m}^{n-1} \lambda_i (bx_i - y_i)^2 . \tag{47}$$

Define

$$C = \sum_{i=1}^{n-1} \lambda_i (x_i - y_i)^2 . \tag{48}$$

Using the definition of Δ_i in Eq. 29,

$$L_{\text{place}}(m) = C + \sum_{i=m}^{n-1} \Delta_i. \tag{49}$$

Therefore,

$$L_{\text{place}}(m+1) - L_{\text{place}}(m) = -\Delta_m. \tag{50}$$

If $m < q$, then $\Delta_m \geq 0$, so $L_{\text{place}}(m+1) \leq L_{\text{place}}(m)$. If $m \geq q$, then $\Delta_m < 0$, so $L_{\text{place}}(m+1) > L_{\text{place}}(m)$. Thus, the placement objective is nonincreasing up to $m = q$ and strictly increasing thereafter, implying $m^\star = q$.

If $\Delta_i \geq 0$ for all $i = 1, \ldots, n-1$, then $L_{\text{place}}(m)$ is nonincreasing over the entire chain, so $m = n$ is a minimizer. Zero-valued marginal effects may yield tied minimizers. □

Threshold interpretation. Assume $x_i > 0$ and define

$$r_i = \frac{y_i}{x_i}. \tag{51}$$

Then

$$\Delta_i = \lambda_i (b-1) x_i^2 \left[(b+1) - 2r_i\right]. \tag{52}$$

For $\lambda_i > 0$ and $b \neq 1$,

$$\Delta_i < 0 \quad \Longleftrightarrow \quad \begin{cases} r_i < (1+b)/2, & 0 < b < 1, \\ r_i > (1+b)/2, & b > 1. \end{cases} \tag{53}$$

Hence, under the sign pattern in Proposition 4.2, the threshold $(1+b)/2$ identifies where the correction changes from locally harmful to locally beneficial.

General nonlinear simulators. The closed-form characterization above relies on the scalar-chain and scaling-compatibility assumptions. For a general nonlinear simulation graph, placement is treated as a discrete model-selection problem, in the similar idea as searching over augmentation locations in neural-augmented physics engines (Heiden et al., 2021). Let $\widehat{\Psi}_m = (\widehat{\theta}_m, \widehat{\Phi}_m)$ denote the parameters obtained by training the augmented simulator with correction placed at m. The practical placement is selected by

$$m^\star = \arg\min_{m \in \mathcal{M}} L_{t,\text{val}}\left(\widehat{\Psi}_m; m\right). \tag{54}$$

APPENDIX E EXPERIMENT SETTINGS

APPENDIX E.1 EXPERIMENT SETTINGS FOR CASE STUDY 1

Simulator and fundamental diagram. We use the Cell Transmission Model (CTM) (Daganzo, 1994), a first-order macroscopic traffic model that discretizes a road into cells and updates cell densities by conservation of vehicles. The implemented fundamental diagram has independently fitted free-flow speed $v_{f,i}$, capacity Q_i, and congestion-wave speed w_i:

$$f_i(\rho) = \max(0, \min\{v_{f,i}\rho, Q_i, w_i(\rho_{j,i} - \rho)\}). \tag{55}$$

Because Q_i is fitted independently of the free-flow and congested branches, the resulting diagram can contain an explicit capacity plateau. Jam density is fixed at 180 veh/mile/lane, giving $\rho_{j,i} = 540$ veh/mile for the three-lane corridor. The corresponding sending and receiving functions are

$$S_i(t) = \max(0, \min\{v_{f,i}\rho_i(t), Q_i\}), \qquad R_i(t) = \max(0, \min\{w_i(\rho_{j,i} - \rho_i(t)), Q_i\}). \tag{56}$$

The flow across the interface between cells i and $i+1$ is

$$q_i(t) = \min\{S_i(t), R_{i+1}(t)\}. \tag{57}$$

We retain this hard minimum and differentiate through its active branches. The full simulator is implemented in float64 arithmetic.

With cell length L_i in miles and net source a_i^k in veh/h, the numerical update is

$$\rho_i^{k+1} = \text{clip}_{[0,\rho_{j,i}]}\left[\rho_i^k + \frac{\Delta t}{L_i}\left(q_{i-1}^k - q_i^k + a_i^k\right)\right], \qquad \Delta t = \frac{5}{3600}\ \text{h}. \tag{58}$$

The inlet flow q_0^k is supplied by the preprocessed upstream observations, and the downstream boundary uses free outflow, $q_{13}^k = S_{13}^k$. Clipping enforces density bounds after each update; when active, it modifies the unconstrained conservation update.

Freeway corridor and observations. The simulated corridor is a segment of Interstate 15 northbound bounded by seven consecutive mainline detectors from the Caltrans Performance Measurement System (PeMS) (Chen et al., 2001), which define six road links. Dividing these links into $(2, 2, 3, 2, 3, 1)$ numerical cells yields the 13-cell chain in Fig. 9. On- and off-ramp flows enter or leave the corridor at cells C_1, C_8, and C_{13}. Detector measurements are available at $\{C_2, C_4, C_7, C_9, C_{12}, C_{13}\}$, and C_{13} is the terminal target. The remaining seven cells are unobserved.

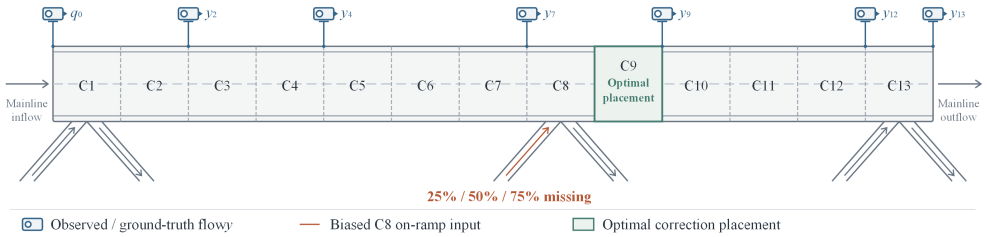


Figure 9: The 13-cell CTM graph.

The corridor lies in Caltrans District 8 and spans 5.831 miles. All six links have three lanes. Table 4 gives their lengths and detector-to-cell mapping. The shortest cell is approximately 0.4143 miles, giving a maximum speed-based CFL ratio of approximately 0.302 at the five-second step and the upper free-flow-speed bound of 90 mph.

Table 4: Detector-to-cell mapping; lengths are in miles.

Link	Upstream detector	Downstream detector	Length	Cells	Cell length
1	823418	817779	0.931	C1–C2	0.4655
2	817779	817781	0.900	C3–C4	0.4500
3	817781	826595	1.243	C5–C7	0.4143
4	826595	817785	0.957	C8–C9	0.4785
5	817785	826629	1.378	C10–C12	0.4593
6	826629	817789	0.422	C13	0.4220

The on-ramp and off-ramp station pairs are $(823420, 823419)$ at C1, $(826597, 826596)$ at C8, and $(826630, 826632)$ at C13. Initial densities are assigned from the corresponding detector anchors. C6 and C11 use the arithmetic mean of their two endpoint-detector density proxies, while C13 uses its downstream detector. FD parameter initialization follows the same anchor assignment, as detailed below.

Data and evaluation protocol. Each day provides four 120-minute trajectories. Each trajectory is initialized from detector-derived densities and then evolves freely under the observed boundary and ramp inputs with a five-second step, without further data assimilation. Simulated flows are compared with detector flows every five minutes. The four daily windows are 01:00–03:00, 03:30–05:30, 10:00–12:00, and 17:30–19:30, using the recorded local timestamps. These windows are fixed across all weeks and day types. Each trajectory contains $24 \times 60 = 1440$ numerical steps and 24 prediction targets. Its initial observation initializes the state and is excluded from the target sequence, yielding 672 targets per observed cell per week.

Five-minute vehicle counts are multiplied by 12 to obtain flow rates in veh/h. Detector density proxies are calculated as flow divided by speed in mph. Duplicate station–timestamp records retain the last entry. A trajectory is retained only when all required density, reference-flow, and ramp-flow values are available at its 25 observation timestamps.

The inflow equals the first detector's flow minus the net ramp source assigned to the first link. A link's downstream reference flow is adjusted by subtracting the net source in the immediately following link, where present; this affects the targets at C7 and C12. Other observed outputs use the corresponding detector flows directly. Reference targets are constructed before injecting the input bias and are shared across methods.

Boundary and ramp inputs are held constant over each five-minute interval. At the end of its 60 numerical substeps, outflows are recomputed from the updated densities and compared with the next reference observation. These endpoint flows define the observation operator used for training and evaluation.

We train on September 8–14, 2025, and select checkpoints and correction positions on September 15–21. The main comparison evaluates October 6–12 using validation-selected checkpoints, without further parameter updates. September 22–28 and September 29–October 5 are development weeks used for supplementary analyses. Table 5 summarizes the complete split.

Table 5: Data split; all dates are in 2025.

Split	Dates	Use
Training	Sep. 8–14	Calibration, normalization, training
Validation	Sep. 15–21	Checkpoint and placement selection
Development A	Sep. 22–28	Supplementary analyses
Development B	Sep. 29–Oct. 5	Additional development analyses
Final evaluation	Oct. 6–12	Main-table and node evaluation

Figure 10 shows one mean daily profile for each week. At each five-minute clock time, a curve averages seven days and seven mainline detectors, giving 49 observations. These descriptive profiles use raw detector flows without smoothing. Shading marks the four rollout windows.

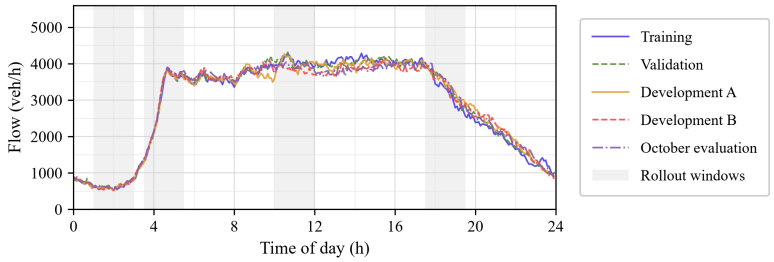


Figure 10: Weekly mean flow profiles; shading marks rollout windows.

Figure 11 separates the weekly profiles into weekdays and weekends, retaining the same week colors. Each clock-time mean averages 35 detector observations for weekdays and 14 for weekends. The comparison describes the demand patterns encountered by the fixed rollout windows.

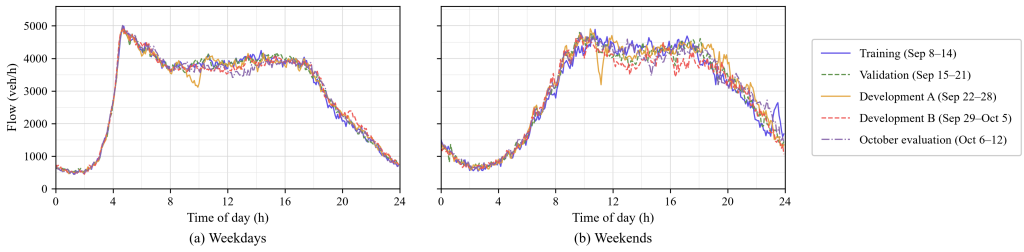


Figure 11: Weekday and weekend flow profiles across five weeks.

Injected model–plant mismatch. To create a mismatch with a known source, we remove 25%, 50%, or 75% of the measured on-ramp inflow at C_8 before it enters the simulator, while keeping all mainline reference observations unchanged. The simulator therefore receives a biased ramp input $\widetilde{r}_8(t)$, and the resulting vehicle deficit propagates downstream from C_8 to the terminal cell.

FD adaptation changes transport and storage, whereas the correction introduces an explicit signed source term. The network receives the retained ramp input and learns through the selected flow-prediction loss. The known missing inflow is not used as a correction target.

Supervision and correction placement. The prediction errors are standardized using training-derived scales. Let $\mathcal{D} = \{2, 4, 7, 9, 12, 13\}$ index the observed cells. For each $i \in \mathcal{D}$, let s_i be the population standard deviation of its 672 training targets, lower-bounded by 1 veh/h. The node loss is

$$\ell_i = \frac{1}{BN} \sum_{b=1}^{B} \sum_{n=1}^{N} \left(\frac{\widehat{q}_{i,b,n} - q^{\mathrm{ref}}_{i,b,n}}{s_i} \right)^2, \qquad B = 28, \quad N = 24. \tag{59}$$

We consider three training objectives: *Terminal*, which uses ℓ_{13}; *Mean*, which uses $\frac{1}{6}\sum_{i\in\mathcal{D}} \ell_i$; and *C9*, which uses ℓ_9. Training-derived scales remain fixed for validation and evaluation. Mean supervision includes the terminal cell and therefore differs from the internal-only loss used in the gradient diagnostic.

The correction can be placed at any of the 13 cells. We train each correction method separately at all 13 positions under 50% bias and Terminal supervision, using the same training budget and paired seeds. Each run is evaluated at its validation-selected checkpoint, and placement is selected by the mean validation C_{13} RMSE across seeds. GraphBC selects C_9, with C_{10} nearly tied, whereas Correction Only attains its minimum at C_{13}. Both correction methods use C_9 in the main comparison to control for insertion location, and all comparisons report C_{13} flow RMSE regardless of the training loss.

Correction network and training. The correction at position m is an additive source/sink term $R_\theta(z_m(t))$, produced by a residual multilayer perceptron with one hidden layer of 32 tanh units. Its input features are

$$z_m(t) = \left[\frac{\widetilde{r}_8(t)}{1000}, \frac{q_{\mathrm{up}}(t)}{5000}, \frac{\rho_m(t)}{\rho_{j,m}}, \sin(2\pi t/T), \cos(2\pi t/T) \right], \tag{60}$$

where q_{up} is the corridor inlet flow, ρ_m is the density of the correction cell, and T is the trajectory length. The temporal features use elapsed time relative to the start of the two-hour trajectory. The output is bounded as $1500\tanh(\cdot)$ veh/h. The network contains 225 parameters. Hidden weights use Xavier-uniform initialization; hidden biases and all output-layer weights and biases are zero initially, giving zero initial correction for every seed.

The network is evaluated at the start of each five-minute interval using the current simulated density. Its signed output is held constant over the next 60 substeps and added to the net source at the selected cell. Density bounds are enforced after each CTM update through Eq. 58.

Local calibration and simulator initialization. Local first fits one FD to each of the seven mainline detectors using the full training week. Accepted samples have observation coverage of at least 80%, nonnegative flow, speed between 5 and 90 mph, and density proxy between 0 and 900 veh/mile. Calibration minimizes unnormalized flow MSE using differential evolution with 100 maximum iterations, population multiplier 12, tolerance 10^{-7}, polishing enabled, and fixed seed $20250908 + \mathrm{stationID}$.

Calibration bounds are $v_f \in [20, 90]$ mph and $w \in [1, 40]$ mph. With $q_{\mathrm{ref}} = \max(q_{0.99}, 1000)$, where $q_{0.99}$ is the detector's training-flow 99th percentile, capacity is fitted within

$$[\max(1000, 0.35q_{\mathrm{ref}}), \min(6500, 1.20q_{\mathrm{ref}})]$$

veh/h.

Detector-adjacent cells inherit the corresponding fitted parameters. C6 and C11 use arithmetic means of their endpoint-detector parameters, while C13 uses its downstream detector. Table 6 reports the resulting cell initialization. Figure 12 shows the initialized cell-specific fundamental diagrams with their corresponding training samples.

End-to-end optimization. During end-to-end training, each physical FD parameter is obtained from an independent raw coordinate through

$$p_i = p_{\min} + (p_{\max} - p_{\min})\sigma(\phi_{p,i}), \qquad p \in \{v_f, Q, w\}. \tag{61}$$

The bounds are $v_f \in [20, 90]$ mph, $Q \in [1000, 6500]$ veh/h, and $w \in [1, 40]$ mph. Adjacent cells may share initial values but are updated independently. Jam densities, lane counts, cell lengths, and graph topology remain fixed.

Table 6: Local FD initialization; jam density is 540 veh/mile.

Cell	v_f (mph)	Q (veh/h)	w (mph)
C1	63.89	4242.93	9.89
C2	66.24	4701.47	10.09
C3	66.24	4701.47	10.09
C4	61.60	3508.16	7.29
C5	61.60	3508.16	7.29
C6	63.60	3827.66	8.79
C7	65.60	4147.16	10.29
C8	65.60	4147.16	10.29
C9	65.29	4495.68	9.86
C10	65.29	4495.68	9.86
C11	64.93	4468.98	10.33
C12	64.56	4442.29	10.80
C13	64.12	5136.77	12.15

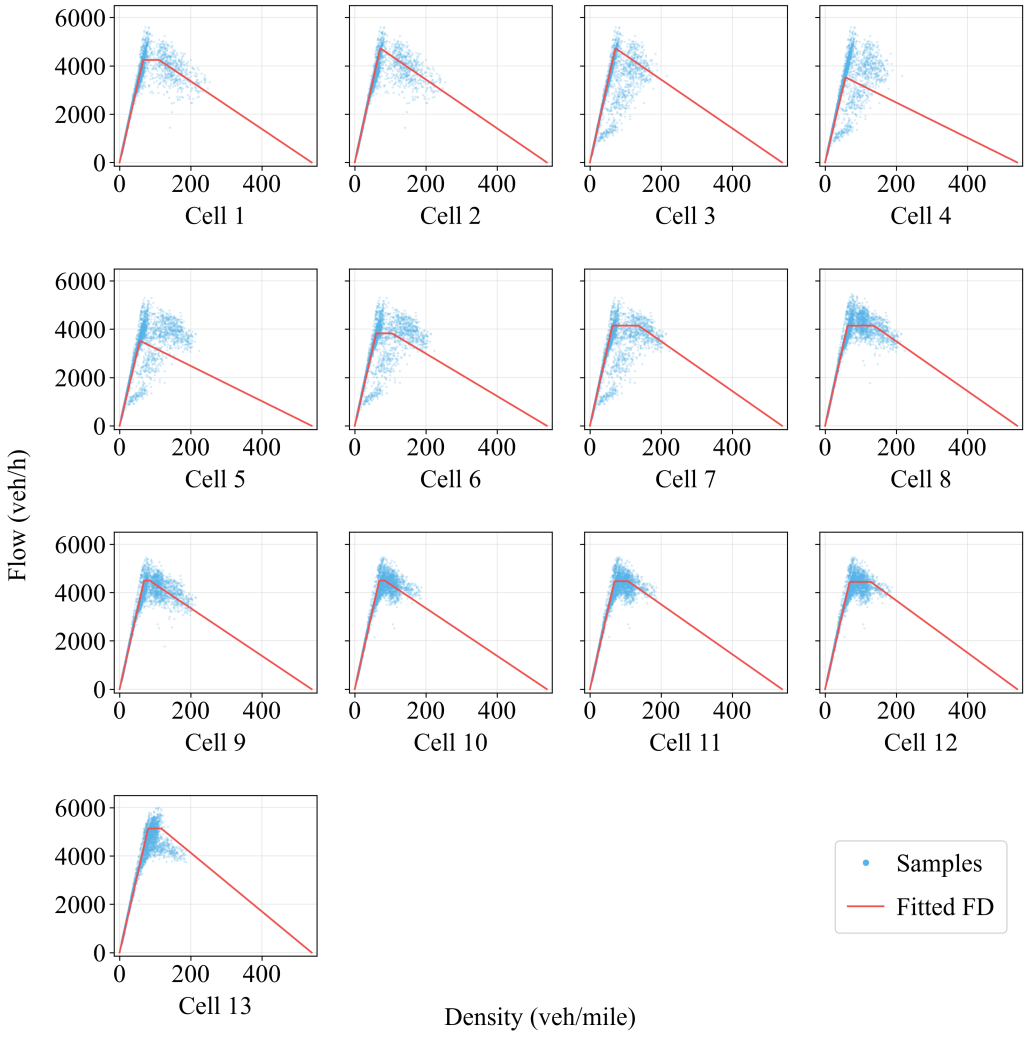


Figure 12: Initialized cell-specific fundamental diagrams and corresponding training samples.

Table 7: Training settings.

Setting	Value
Optimizer	Adam
Optimization updates	80
Batch size	28 trajectories
Trajectory duration	120 minutes
Simulation step	5 seconds
Observation interval	5 minutes
Simulator learning rate	0.025
Correction learning rate	0.005
Gradient clipping	Global norm 10
Arithmetic / device	Float64 / CPU
Random seeds	0, 1, 2
Validation updates	1, 2, 4, . . ., 80
Checkpoint criterion	Lowest matching validation loss

After Local calibration, Correction Only updates the 225 correction parameters, E2E Only updates the 39 simulator parameters, and GraphBC updates both blocks, totaling 264 parameters. Local performs no further optimization.

Validation is performed after update 1 and every even-numbered update. Each checkpoint is scored using the same supervision objective as training, with training-derived normalization. The checkpoint with the lowest matching validation loss is retained; exact ties keep the earliest checkpoint. Under Terminal supervision, this is equivalent to selecting the lowest validation C13 RMSE. All runs complete 80 updates without early stopping.

Table 7 summarizes the optimization settings. Each update uses the complete batch of 28 training trajectories. Adam uses separate learning rates for the simulator and correction parameter blocks, and the global gradient norm is clipped before each update.

Training procedure. Algorithm 1 summarizes training for one seed and one correction position. Let $\phi \in \mathbb{R}^{39}$ denote the simulator coordinates in Eq. 61, and let $\theta \in \mathbb{R}^{225}$ denote the correction-network parameters. The optimized parameter block is

$$\psi = \begin{cases} \phi, & \text{E2E Only}, \\ \theta, & \text{Correction Only}, \\ (\phi, \theta), & \text{GraphBC}. \end{cases} \tag{62}$$

The Local calibration supplies the initial simulator coordinates ϕ_0. The correction network is initialized as described above, giving zero initial correction.

The operator $\mathrm{Rollout}(\mathcal{T}; p(\phi), R_\theta, m)$ advances the trajectories in $\mathcal{T}$ from their observed initial states and returns flows at the six observed cells and 24 prediction times. During this rollout, R_θ is evaluated at the beginning of each five-minute interval and added to the net source at cell m for that interval. For E2E Only, $R_\theta \equiv 0$. The physical simulator parameters $p(\phi)$ are obtained through Eq. 61.

Each training rollout produces a $28 \times 24 \times 6$ prediction tensor. The selected standardized loss aggregates these predictions into one scalar objective, followed by one backward pass and one Adam update. Validation uses the current parameters and the same normalization scales and supervision objective.

The three seeds are 0, 1, and 2. Local calibration uses fixed detector-specific seeds, and E2E Only uses fixed initialization and deterministic full-batch updates; their results are identical across the three runs. Correction Only and GraphBC vary through hidden-weight initialization despite their common zero initial correction. Reported uncertainty is the sample standard deviation across the three runs.

Simulator parameter shift. We measure parameter shift as the root-mean-square relative change of the 39 physical fundamental-diagram parameters from their Local-calibrated values, expressed as

Algorithm 1 End-to-end training with graph-embedded correction

Require: Training trajectories $\mathcal{T}_{\mathrm{tr}}$ and validation trajectories $\mathcal{T}_{\mathrm{val}}$, with reference flows Y_{tr} and Y_{val}; Local coordinates ϕ_0; correction position m; training objective $\mathcal{L}$; trainable parameter block ψ
Ensure: Validation-selected parameters $\psi^\star$

```
Initialize φ ← φ_0 and the correction network when used
Compute training-derived normalization scales s = {s_i}_{i∈D}
Initialize Adam for ψ; set L_best ← ∞
for u = 1, ..., 80 do
    q̂_tr ← Rollout(T_tr; p(φ), R_θ, m)
    L_tr ← L(q̂_tr, Y_tr; s)
    g ← ∇_ψ L_tr
    Clip the global norm of g to 10
    ψ ← AdamStep(ψ, g)
    if u = 1 or u is even then
        q̂_val ← Rollout(T_val; p(φ), R_θ, m)
        L_val ← L(q̂_val, Y_val; s)
        if L_val < L_best then
            Save ψ* ← ψ
            L_best ← L_val
        end if
    end if
end for
return ψ*
```

a percentage:

$$S^{(s)}_{\mathrm{param}} = 100 \sqrt{\frac{1}{39} \sum_{i=1}^{13} \sum_{p \in \{v_f, Q, w\}} \left(\frac{p_i^{(s)} - p_i^{\mathrm{Local}}}{p_i^{\mathrm{Local}}} \right)^2}. \tag{63}$$

Here, p_i^{Local} is the locally calibrated value, and $p_i^{(s)}$ is the physical parameter value at the validation-selected checkpoint for seed s. We compute this metric separately for each seed and report its mean and sample standard deviation across the three seeds. Local and Correction Only retain the calibrated simulator parameters and therefore have zero parameter shift.

Controlled gradient-alignment diagnostic. We evaluate 312 saved GraphBC checkpoints spanning four mismatch configurations (25% and 75% missing input at C_8, and 50% missing input at C_1 and C_{13}), three seeds, 13 correction positions, and training updates 20 and 80. Using the validation trajectories, we define the internal objective as $L_{\mathrm{int}} = \frac{1}{5}\sum_{i\in\{2,4,7,9,12\}} \ell_i$. At each checkpoint, we fix the correction weights and placement and apply $\phi' = \phi - h g_t / \|g_t\|_2$, where $g_t = \nabla_\phi \ell_{13}$ and ϕ contains the 39 raw simulator coordinates in Eq. 61. We test displacement norms $h \in \{10^{-6}, 10^{-4}, 10^{-2}\}$, restoring the saved parameters before each probe.

At $h = 10^{-6}$, the internal objective decreases at all 198 positively aligned checkpoints and increases at all 114 negatively aligned checkpoints. Among the 198 checkpoints with aggregate improvement, 182 have an increased loss at at least one individual internal node. The alignment sign predicts the direction of internal-loss change at 312/312, 302/312, and 238/312 checkpoints for $h = 10^{-6}$, 10^{-4}, and 10^{-2}, respectively.

Controlled validation of placement analysis. We construct 384 free-flow steady-state CTM scenarios using three seeds and 32 samples from each of four reference-source configurations: a single on-ramp, a single off-ramp, two oppositely signed sources, and three signed sources. Reference flows are generated with sources at the designated cells C_1, C_8, and C_{13}. The baseline uses the same fixed fundamental-diagram parameters, constant upstream inflow q_0, and zero internal sources, yielding steady flows $x_i = q_0$. With $b = y_{13}/x_{13}$, a constant correction source $(b-1)q_0$ at position m yields $q_i = q_0$ for $i < m$ and $q_i = bq_0$ for $i \geq m$, implementing the scaling intervention at the steady-flow interface.

Analytical placement predictions are saved before recomputing the CTM at all 13 positions. We evaluate objectives using all 12 internal references, the five observed internal references, and terminal-

only supervision. The terminal weight and each included internal weight equal $1/q_{\text{safe}}^2$, where $q_{\text{safe}} = 0.65 \min_i\{Q_i, w_i\rho_{j,i}/(1 + w_i/v_{f,i})\}$. Optimal sets include all positions within 10^{-10} of the minimum normalized objective.

Analytical and recomputed optimal sets agree in all 384 scenarios under each reference setting. With all 12 internal references, 250 scenarios satisfy the beneficial-suffix condition and 117 satisfy the all-nonnegative case of Proposition 4.2; the remaining 17 validate the general suffix-sum identity in Appendix D. With the five observed internal references, 122 scenarios have a unique optimum. Under terminal-only supervision, all 13 positions attain the same objective value in every scenario.

APPENDIX E.2 EXPERIMENT SETTINGS FOR CASE STUDY 2

Simulator overview. We use an industrial-scale citywide ride-hailing simulator that reproduces how users request trips across several service categories, allowing a user to select multiple categories at once. Given category-level service attributes and market features of a city on a given day, the simulator predicts request outcomes at three levels: the probability that each service category is requested, the probability of each bundle of categories that a user may select together, and a city-level output aggregated from the bundles.

Simulation graph. The simulation graph (Fig. 7) contains three nodes. The category-level module v is a trainable network with a structured prior: its output for each category is formed from monotone sigmoidal response terms whose directions are fixed by domain knowledge, while a parameter-generating network maps market features to the magnitudes and locations of these responses. Training therefore adjusts the response strengths but cannot reverse their directions. The bundle-level node u has no trainable parameters and computes the probability of each bundle from the category-level probabilities under a conditional-independence assumption across categories. The terminal node t aggregates the bundle-level probabilities into the city-level output.

Source of mismatch. In practice, user choices across categories are correlated, so the conditional-independence assumption in u is a mechanistic simplification that introduces systematic bias. Training v adjusts the category-level probabilities, while u combines them through a fixed conditional-independence structure.

Correction node. The correction node c_u is placed at $m = u$, between u and t:

$$\widetilde{x}_u = \text{clip}_{[0,1]}\big(x_u + R_\theta(z_u)\big), \qquad z_u = (X, x_v), \tag{64}$$

where X denotes the market features and R_θ is a two-layer fully connected network with 32 hidden units, using ReLU and Tanh activations, that outputs one residual for each bundle. The clipping keeps the corrected probabilities in $[0, 1]$.

Placement and graph structure. The correction is placed immediately downstream of u to adjust the bundle probabilities produced by the conditional-independence model before terminal aggregation. Besides the main path from u, it receives the category-level outputs x_v and the market features X through two skip edges, since the magnitude of the mismatch depends on how strongly user choices across categories are correlated, which varies with the category-level probabilities and market conditions. The resulting graph is a general DAG rather than a chain, so the terminal gradient reaches v through both $v \to u \to c_u$ and $v \to c_u$, as described by the path-sum Jacobian in Eq. 15.

Training and supervision. Both methods are trained end to end from scratch with the same node-wise loss,

$$\mathcal{L} = \mathcal{L}_v + \mathcal{L}_{\text{bundle}} + \mathcal{L}_t, \tag{65}$$

where each term is the mean squared error against the observed values at the corresponding level, and the three terms are weighted equally. The bundle-level term is applied to x_u in E2E Only and to $\widetilde{x}_u$ in GraphBC. Both therefore fit the same ground truth, and the two methods differ only in whether the correction is present. All parameters are optimized with Adam, and the checkpoint with the lowest validation loss is used for testing.

Data. The data cover 40 cities over 333 consecutive days at a city–day resolution, and are split chronologically into training, validation, and test sets with a 6:2:2 ratio. Input features consist of aggregated city-level indicators of supply, demand, and operational efficiency, together with static city-level features. All results are averaged over three random seeds. The data are proprietary and cannot be released.

Metrics. State MSE is computed between each node's output and the observed values, averaged over samples and output dimensions. MAPE is computed for each reported output as $\frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|/|y_i|$. The three reported outputs of v correspond to three representative service categories.

Additional results. Fig. 13 compares the output distributions on the test set with the ground truth. Under E2E Only, the distributions of v shift toward lower values and that of t toward higher values, mirroring the systematic biases in Fig. 8, whereas GraphBC closely matches the ground truth for all outputs.

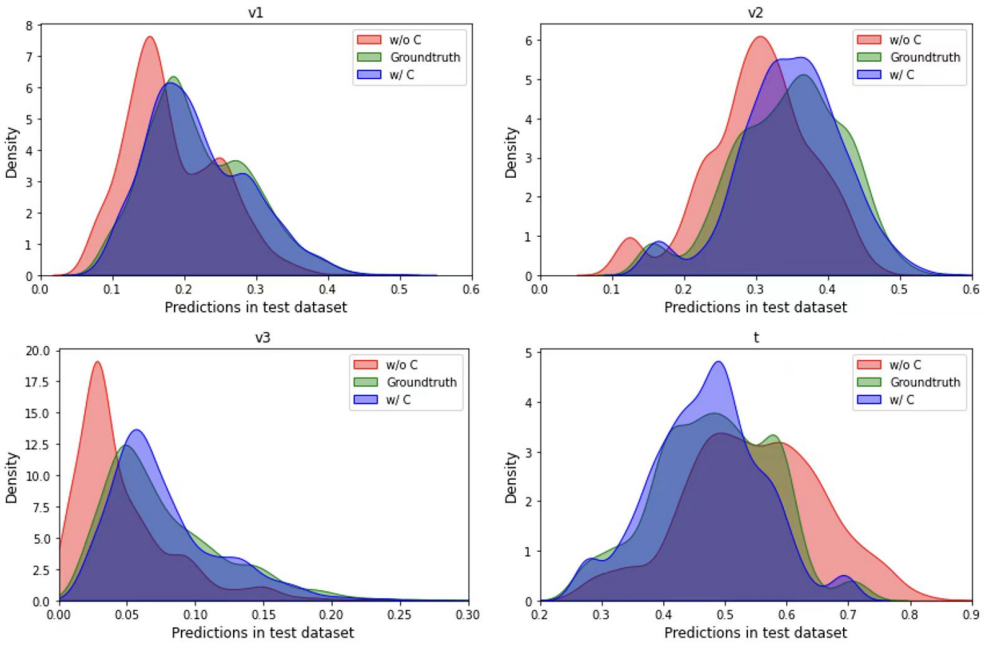


Figure 13: Density of test-set predictions for three outputs of v and for t. "w/o C": E2E Only; "w/ C": GraphBC.