Adaptive Federated Learning via Dynamical System Model

Anonymous authors
Paper under double-blind review

Abstract

Hyperparameter selection is critical for stable and efficient convergence of heterogeneous federated learning, where clients differ in computational capabilities, and data distributions are non-IID. Tuning hyperparameters is a manual and computationally expensive process as the hyperparameter space grows combinatorially with the number of clients. To address this, we introduce an end-to-end adaptive federated learning method in which both clients and central agents adaptively select their local learning rates and momentum parameters. Our approach models federated learning as a dynamical system, allowing us to draw on principles from numerical simulation and physical design. Through this perspective, selecting momentum parameters equates to critically damping the system for fast, stable convergence, while learning rates for clients and central servers are adaptively selected to satisfy accuracy properties from numerical simulation. The result is an adaptive, momentum-based federated learning algorithm in which the learning rates for clients and servers are dynamically adjusted and controlled by a single, global hyperparameter. By designing a fully integrated solution for both adaptive client updates and central agent aggregation, our method is capable of handling key challenges of heterogeneous federated learning, including objective inconsistency and client drift. Importantly, our approach achieves fast convergence while being insensitive to the choice of the global hyperparameter, making it well-suited for rapid prototyping and scalable deployment. Compared to state-of-the-art adaptive methods, our framework is shown to deliver superior convergence for heterogeneous federated learning while eliminating the need for hyperparameter tuning both client and server updates.

1 Introduction

Federated learning collaboratively trains a global model across decentralized clients without sharing raw data. In the presence of intermittent client availability, heterogeneous computational capabilities, and non-IID data distributions, federated learning can suffer from issues such as objective inconsistency Wang et al. (2020) and client drift Shi et al. (2022), leading to degraded performance. As data and client heterogeneity increases, balancing efficient and stable global convergence across clients remains a challenge.

Prior work addresses these issues through aggregation adjustments Acar et al. (2021); Pathak & Wainwright (2020); Karimireddy et al. (2020); Li et al. (2020a;b); Malinovsky et al. (2023); Charles & Konečnỳ (2020), momentum-based updates Das et al. (2022); Xu & Huang (2022); Khanduri et al. (2021), and Newton-like methods Li et al. (2019). However, convergence of these client and server-side update strategies often depends on carefully tuned hyperparameters. The performance of federated learning is sensitive to the choice of hyperparameters such as learning rates and regularization terms Kim et al. (2023); Acar et al. (2021); Li et al. (2020b). The challenge of selecting appropriate hyperparameters is compounded as the hyperparameter search space scales combinatorially with the number of clients. Practical implementations often apply a uniform hyperparameter configuration across all clients to avoid searching the entire hyperparameter space; however, this approach often yields suboptimal performance due to data and computational heterogeneity. To address this, adaptive methods have been proposed, typically focusing on either client step size selection or server-side aggregations.

Server-side adaptive methods adjust learning rates and aggregation strategies to account for system heterogeneity. For example, SCAFFOLD Karimireddy et al. (2020) uses control variates to reduce client drift, while

FedYogi, FedAdaGrad, and FedAdam Reddi et al. (2020) extend adaptive gradient methods to federated learning aggregation steps in order to stabilize noisy updates. FedExp Jhunjhunwala et al. (2023) applies exponential moving averages to smooth aggregation, and FedDyn Acar et al. (2021) introduces a dynamic regularizer to promote stability. However, these methods often rely on prior knowledge of client learning rates, limiting their ability to support client-specific and non-uniform step sizes. Moreover, these aggregation strategies introduce additional hyperparameters that affect the convergence and stability of the global model.

On the other hand, client-side adaptive methods focus on dynamically adjusting client updates to handle heterogeneity. FedProx Li et al. (2020b) adds a proximal term to limit client divergence, while MOON Li et al. (2021) modifies client objectives to better align local heterogeneity. Auto-tuning methods like Kim et al. (2023) adapt client learning rates based on local gradients. Although these methods improve client optimization, without a coordinated server aggregation, these client-side methods alone can be prone to issues in heterogeneous federated learning including client drift and degraded performance.

In this work, we introduce a fully adaptive federated learning method that dynamically tunes hyperparameters for *both* individual client updates and central server aggregation for efficient and stable convergence in heterogeneous settings. Our approach is based on a dynamical system model of federated learning Agarwal et al. (2025) and draws on principles from numerical simulation and circuit design to select learning rates as well as momentum parameters. Through this perspective, constructing updates for heterogeneous federated learning is effectively translated into designing an adaptive simulation engine. Specifically, by mapping the dynamical system representation to an equivalent circuit, we are able to adopt well-established circuit design and simulation principles to choose step-sizes and momentum terms.

The key advancement over Agarwal et al. (2025) lies in reformulating hyperparameter selection as a circuit design problem. While Agarwal et al. (2025) introduces the foundational circuit mapping, critical parameters such as time-step sizes and inductance values still require manual tuning. Moreover, allowing each client to adopt its own momentum and step-size parameters would cause the hyperparameter space to grow combinatorially with the number of clients, making conventional tuning approaches impractical. Adaptive FedECADO addresses this limitation by analytically deriving client-specific momentum parameters through the critical damping condition, and by employing local truncation error (LTE)-based adaptive time-stepping for both client and server updates. This results in dynamically adjusted, client-specific learning rates and momentum terms that naturally adapt to local gradient geometry, eliminating the need for problem-specific hyperparameter tuning.

First, our methodology selects client learning rates based on numerical integration principles to ensure accuracy. This allows each client to operate with independent, data-specific learning rates that adapt to the local gradient space. To address objective inconsistency, we employ an integration/extrapolation operator at the central server's aggregation step that aligns the client updates in synchronous timescale. Then, on the server side, we adapt learning rate and momentum parameters using insights from circuit design, to promote fast and stable convergence.

Our main contribution is a fully adaptive, end-to-end federated learning method for heterogeneous settings in which both client and server updates are dynamically controlled by a single global parameter. Crucially, this parameter emerges naturally from a circuit-based dynamical system formulation of federated learning, which provides a physically grounded abstraction for designing momentum values under client heterogeneity. By treating the federated learning trajectory as a simulation, we impose numerical stability and error-control principles from circuit simulation directly on the learning process—capabilities that are difficult to express or enforce within conventional discrete optimization frameworks.

As a result, the performance of our method is largely insensitive to the value of the global control parameter, eliminating the need for hyperparameter tuning and making the system practical to deploy at scale. We benchmark our approach across a wide range of federated learning scenarios with heterogeneous and non-IID data distributions, and demonstrate that our method achieves strong model performance without tuning, consistently outperforming prior adaptive approaches that depend on carefully selected hyperparameters. These results underscore the necessity of the circuit formulation for enabling principled, stable, and robust adaptation in heterogeneous federated learning.

2 Dynamical Model of Federated Learning

The challenges associated with heterogeneous federated learning (such as client drift and objective inconsistency) and hyperparameter tuning stem from discrete client updates that struggle to handle variability in data and client computation. Rather than addressing these issues directly by tuning the hyperparameters of the discrete update, we take a different approach by modeling federated learning as a continuous-time dynamical system.

The dynamical system model captures the evolution of both client and server states over continuous-time, with the goal of modeling the trajectory of the following global optimization problem:

$$\min_x \sum_i p_i f_i(x, \mathcal{D}_i), \tag{1}$$

where x represents the global model parameters and f_i represents the local loss function of a client, i, characterized by a local dataset $\mathcal{D}_i$. For simplicity, we denote $f_i(x) \equiv f_i(x, \mathcal{D}_i)$. The local objective functions are weighted by a scalar, p_i, representing the size of their local dataset Agarwal et al. (2025); Wang et al. (2020).

We adopt a dynamical system model for solving (1) from Agarwal et al. (2025) that represents the states of each client as $x_i(t)$ and the central agent model states as $x_c(t)$. To couple the client states with the central agent states, Agarwal et al. (2025) introduces a coupling flow vector, $I_{L_i}(t)$, that captures the *accumulated* difference between the local client state x_i and the central state x_c over the simulation time according to:

$$I_{L_i}(t) = L_i^{-1} \int_0^t (x_c(s) - x_i(s))\, ds, \tag{2}$$

with its time derivative given by:

$$\dot{I}_{L_i}(t) = L_i^{-1} (x_c(t) - x_i(t)). \tag{3}$$

The coupling vector, $I_{L_i}(t)$, is then integrated into the dynamical system modeling the continuous-time evolution of the client and central states:

$$\frac{d}{dt} x_c(t) + \sum_{i=1}^{|\mathcal{C}|} I_{L_i}(t) = 0 \tag{4}$$

$$L_i \dot{I}_{L_i}(t) = x_c(t) - x_i(t) \quad \forall i \in [1, |\mathcal{C}|] \tag{5}$$

$$-I_{L_i}(t) + \frac{d}{dt} x_i(t) + p_i \nabla f_i(x_i(t)) = 0 \quad \forall i \in [1, |\mathcal{C}|] \tag{6}$$

where $\mathcal{C}$ is the set of clients.

I_{L_i} introduces a second-order dynamic that expedites the speed of convergence to steady-state Agarwal & Pileggi (2023). The hyperparameter L_i represents a client-specific momentum term that controls how quickly the flow variable $I_{L_i}(t)$ responds to discrepancies between the global and local states; larger values of L_i slow the rate of change, introducing smoother, more stable dynamics.

The dynamical system naturally converges toward a steady-state that achieves global consensus across all clients even when their data and compute capacities differ. This naturally accounts for heterogeneity in clients. When the system reaches equilibrium, the vector $I_{L_i} \to 0$, signifying that $x_i \to x_c$ for all clients. The rate at which the system settles is determined by the hyperparameter L_i, which can be tuned to accelerate convergence.

This dynamical system model is inspired by an analog circuit shown in Appendix 6, where I_L is mapped to an electronic component known as an inductor with an inductance of L_i.

2.1 Simulating the Dynamical System in Federated Learning Setting

From the perspective of the dynamical system model, training a federated model in a heterogeneous environment becomes a matter of accurately simulating (4)-(6) to its steady-state. Due to the distributed nature of federated learning, the simulation is partitioned across clients, where each computational node independently models the evolution of a client's state-variables and a central server aggregates these trajectories.

To distribute the simulation, we use the framework in Agarwal et al. (2025) to decouple the full set of coupled ordinary-differential equations (ODEs) (4)-(6) using an iterative Gauss-Seidel (G-S). This method separates each client's subproblem from the central agent by treating the coupling vector, I_L^i, as fixed from the previous iteration.

At the $k+1$ G-S iteration, active clients, denoted by the set $\mathcal{C}_a$, solve their local ODEs by simulating:

$$\dot{x}_i^{k+1}(t) + p_i \nabla f_i(x_i^{k+1}(t)) + I_{L_i}^k(t) = 0, \tag{7}$$

where $I_{L_i}^k(t)$ is treated as a constant coupling term from the previous iteration. Each client is simulated for a client-specific time-window $[0, T_i]$ which varies based on local computational capabilities.

Each active client then communicates the final state $x_i^{k+1}(T_i)$ to the central agent, which then updates the global state, $x_c^{k+1}, (t)$ and the coupling vector, $I_L^{i^{k+1}}(t)$, using the following coupled ODEs:

$$\dot{x}_c^{k+1}(t) = \sum_{i=1}^{n} I_{L_i}^{k+1}(t), \tag{8}$$

$$L_i \dot{I}_{L_i}^{k+1}(t) = x_c^{k+1} - I_{L_i}^{k+1}(t)\hat{G}_i^{th^{-1}} - x_i^{k+1}(t) + I_{L_i}^k(t)\hat{G}_i^{th^{-1}} \quad \forall i \in [1, |\mathcal{C}|]. \tag{9}$$

In this update, each client's state x_i^{k+1} is assumed constant over the server's simulation.

The matrix $\hat{G}_i^{th}$ represents the *first-order sensitivity* of client i to changes in the global state. Inspired by Thevenin impedances in circuit theory (which characterize how the current from a circuit component responds to changes in the voltage), $\hat{G}_i^{th}$ models how a client's state is expected to evolve in response to updates in the central agent state, x_c. This sensitivity allows the central agent to anticipate each client's response, leading to improved convergence as shown in Agarwal et al. (2025). The sensitivity matrix is:

$$\hat{G}_i^{th} = \frac{\partial I_{L_i}}{\partial x_i} = \frac{1}{\Delta t} + p_i \nabla^2 f_i(x_i), \tag{10}$$

where Δt is the client step-size. The full derivation of $\hat{G}_i^{th}$ is provided in Agarwal & Pileggi (2023).

Computing the Hessian, $\nabla^2 f_i$, at every G-S iteration is computationally expensive. Instead, we approximate (10) with a *constant aggregate sensitivity* $\hat{G}_i^{th}$, computed by averaging the Hessian across a representative subset of local datapoints. This results in a client-sensitivity model as follows:

$$\hat{G}_i^{th} = \frac{1}{\Delta t} + p_i \bar{H}^i, \tag{11}$$

with $\bar{H}^i$ denoting the average Hessian over the sampled datapoints. In federated learning with non-IID data, each client's loss is shaped by its unique data distribution, leading to distinct local objectives. The Hessian $\nabla^2 f_i(x_i)$ captures the curvature of each client's loss and as a result, the sensitivity matrix, $\hat{G}_i^{th}$, directly reflects client data heterogeneity since non-IID distributions produce distinct sensitivity matrices across clients. However, computing the full Hessian for large-scale models can become a significant computational bottleneck. To address this, our approach is also compatible with efficient Hessian approximations, such as Fisher information matrices Jhunjhunwala et al. (2024), as discussed in Appendix 19. Since the Hessian primarily serves as a weighting scheme across individual clients, these approximations are shown to have minimal impact on performance.

Using the continuous-time dynamical system model of federated learning, our goal is now to simulate the ODE for each client (7) and server aggregation (8)-(9) without relying on hyperparameters to achieve stable and efficient convergence under heterogeneity.

3 Designing Adaptive Simulation for Heterogeneous Federated Learning

We propose an end-to-end adaptive federated learning method that addresses the challenge of hyperparameter tuning in heterogeneous environments. Our approach is grounded in a continuous-time dynamical systems model (4)–(6), where both client updates and server aggregation are modeled as coupled ODEs. Unlike prior work Agarwal et al. (2025), which required carefully tuned client- and server-specific hyperparameters, we introduce a fully adaptive, momentum-driven federated learning method that eliminates the need for hyperparameter tuning. Our method achieves robust performance across a range of non-IID data distributions and varying client compute capacities. Specifically, we introduce:

1. **Client-side adaptation**: Each client independently simulates its local continuous-time dynamics defined in (7), using non-uniform, adaptive learning rates that are shaped by local gradient space and follow accuracy principles from numerical simulation.
2. **Server-side adaptation**: The server updates its state using client-specific momentum terms, which are chosen to expedite global convergence. Then, the server-side aggregation is performed by a provably stable numerical method with adaptive step-sizes to handle non-uniform client learning rates (to avoid objective inconsistency).

This framework enables stable and efficient training under heterogeneous conditions, and achieves fast convergence without requiring hand-tuned parameters.

3.1 Adaptive Client Updates

By viewing the evolution of client states as a continuous-time model, our goal is to accurately simulate the client ODE. The client ODE in (7) is simulated by marching forward in continuous time at discrete time-steps (or step-sizes) of Δt_i. The client state at the next time point, $t + \Delta t_i$, is defined by:

$$x_i(t + \Delta t) = x_i(t) + \int_t^{t+\Delta t_i} I_{L_i}^k(\tau) + \nabla f_i(x_i(\tau), \mathcal{D}_i), d\tau. \tag{12}$$

In general, the integral in (12) does not have a closed-form solution and instead is approximated using numerical integration methods. To minimize computational overhead, we opt for an explicit Forward-Euler (FE) integration method as follows:

$$x_i(t + \Delta t) = x_i(t) + \Delta t_i \left(I_{L_i}^k(t) + \nabla f_i(x_i(t), \mathcal{D}_i) \right), \tag{13}$$

where Δt_i is the client-specific time step. While the update resembles gradient descent with learning rate Δt_i, it is derived from numerical simulation (further derivations are provided in Appendix 7). As a result, Δt_i is not tuned for convergence to local minimum, but rather selected to ensure an accurate numerical approximation of the continuous-time dynamics in (12). This is guided by error bounds from numerical analysis.

The accuracy of a FE step is measured using the local truncation error (LTE), which captures the difference between the true ODE solution and its numerical approximation. The LTE for FE, denoted as ε_{FE}, is estimated as:

$$\varepsilon_{FE} = \frac{\Delta t_i}{2} |I_{L_i}(t) + \nabla f_i(x_i(t), \mathcal{D}_i) - I_{L_i}(t - \Delta t) - \nabla f_i(x_i(t - \Delta t), \mathcal{D}_i)|. \tag{14}$$

The derivation is provided in Appendix 8. To ensure the discretized steps using FE accurately follow the client ODE in (7), we aim to select Δt_i to maintain the LTE below a pre-defined tolerance γ:

$$\max(\varepsilon_{FE}) \leq \gamma. \tag{15}$$

This follows standard numerical simulation practices, where step sizes are selected based on local error control. To ensure efficient simulation, we use a backtracking line search to find the largest Δt_i satisfying (15)

(Appendix 11). However, Forward-Euler (and similarly gradient descent) updates are prone to divergence as numerical errors accumulate over time Cohen et al. (2024). We adopt prior work from circuit simulation Rohrer & Nosrati (1981) that determines the following Δt that guarantees Forward-Euler stability:

$$\Delta t_{FE} \leq 2\frac{x_i^\top(\nabla f_i(x_i(t)) + I_{L_i})}{\dot{x}_i^\top(t)\dot{x}_i(t)}. \tag{16}$$

We use this bound as an initialization for adaptive step size selection, which is then refined through backtracking to improve LTE accuracy. This creates a client-specific adaptive step size that responds to the local geometry of each client's loss function.

In this work, we assume each active client performs local updates for a fixed wall-clock duration. Due to heterogeneity in compute capabilities, this leads to varying numbers of local epochs and learning rates. This corresponds to each client simulating its local ODE over a total time window of:

$$T_i = \sum_{j=1}^{e_i} \Delta t_i, \tag{17}$$

where e_i is the number of local epochs determined by the client's computational capacity. To accurately coordinate active client updates, clients report their simulation window T_i and $x_i(T_i)$.

3.2 Adaptive Server-Side Aggregation with Momentum

At each communication round, the central server receives both the final simulated state, $x_i(T_i)$, and the simulation window, T_i, from each active client. These updates are aggregated according to the global ODE defined in (8)–(9). Our goal is to design a server-side aggregation that is insensitive to hyperparameters by developing an adaptive simulation that accurately tracks the central agent ODEs.

This is accomplished in two steps. First, we design the momentum parameters, L_i, to ensure fast convergence in continuous-time. Then, we develop an adaptive simulation engine that accurately follows the designed ODE for stable convergence to the steady-state.

3.2.1 Selecting Momentum Parameters

The central agent dynamics, defined in equations (8)–(9), follow a linear ODE where the momentum parameter, L_i, plays a key role in both stability and convergence. This reflects a well-known challenge in momentum-based optimization (e.g., Nesterov's acceleration), where performance is sensitive to the choice of momentum-related hyperparameters. From a dynamical systems perspective, L_i functions as *damping coefficients* that influences the convergence of the global state, $x_c(t)$, by shaping the eigenvalues of the system's linearized dynamics.

These systems can be designed to fall into one of three damping regimes as illustrated in Figure 1:

1. **Over-damped**: The system converges without oscillation and overshooting but more slowly, as the damping suppresses responsiveness.

2. **Under-damped**: The system converges with oscillatory behavior.

3. **Critically damped**: The system converges quickly to steady-state without oscillation.

Our goal is to design the central agent ODE to be *critically damped* for fast convergence to steady-state without oscillations. Optimally selecting L_i to achieve this requires controlling the eigenvalues of the system to lie at the boundary between overdamping and underdamping.

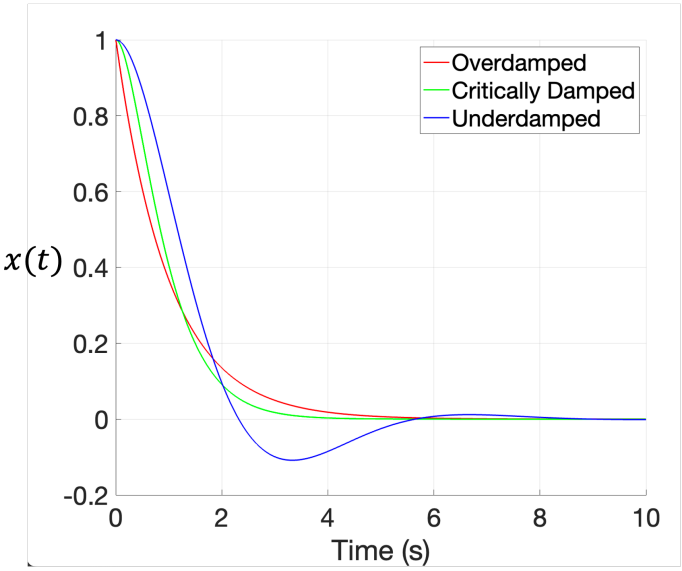

Figure 1: Simulating a single-variable second-order ODE, $\ddot{x}(t) + L\dot{x} + 10x(t) = 0$, we vary the parameter L to achieve overdamped, critically damped, and underdamped systems.

Mathematically, this can be achieved by optimizing:

$$\max_{L} inf \left(Re(-eig(\begin{bmatrix} C, 0 \\ 0, L \end{bmatrix}^{-1} G)) \right) \tag{18}$$

$$s.t. \quad Im(-eig(\begin{bmatrix} C, 0 \\ 0, L \end{bmatrix}^{-1} G)) = 0 \tag{19}$$

where $eig(\cdot)$ is the eigenvalue and $Re(\cdot)$ and $Im(\cdot)$ capture the real and imaginary components. G is the collective client sensitivities. The full derivation is in Appendix 10 Solving (18) is computationally challenging, as eigenvalue computation is impractical with a large number of clients. Rather, we develop an approximate solution that uses the structure of federated learning to select a momentum parameter, L_i, that puts the system close to critical damping. To develop an approximate solution, we map the dynamical system to an analog circuit in Appendix 6 where the momentum parameter translates into selecting an inductance. This enables us to use tools from circuit theory to design the inductance to achieve critical damping.

Our approach is inspired by Thevenin impedances from circuits, which is used to simplify the behavior of complex interconnected components. By treating each client as a branch in a larger system connected through a central aggregator, we can apply similar techniques to analyze how each client's flow vector, I_{L_i}, (modeled by inductor current) responds to the shared central state x_c. To motivate this analysis, we first examine the dynamics of a single client's coupling variable I_{L_i} from the central agent equation (8):

$$I_{L_i}(t) = \dot{x}_c(t) - \sum_{j \neq i} I_{L_j}(t). \tag{20}$$

This reveals two sources of coupling for client i: a direct coupling to all other clients through $\sum_{j \neq i} I_{L_j}(t)$, and a coupling to the central agent dynamics through $\dot{x}_c(t)$. While the cross-client term $\sum_{j \neq i} I_{L_j}(t)$ appears directly in the equation, the underlying mechanism is indirect — each client first updates the central agent state x_c, which then propagates to influence other clients in subsequent aggregation steps. The central agent therefore acts as the intermediary through which all inter-client interactions are mediated.

To quantify the relative strength of these two couplings, we study the sensitivity of I_{L_i} with respect to the central state x_c:

$$\frac{\partial}{\partial x_c} I_{L_i}(t) = \frac{d}{dx_c}\dot{x}_c(t) - \sum_{j \neq i} \frac{\partial}{\partial x_c} I_{L_j}(t). \tag{21}$$

This sensitivity combines:

1. The influence of I_{L_i} on $\dot{x}_c$ (i.e., $\frac{d}{dx_c}\dot{x}_c(t)$), which is the direct path through which client i influences the central state, and
2. The indirect influence of I_{L_i} on all other clients' inductor currents $\frac{\partial}{\partial x_c} I_{L_j}(t)$ $(j \neq i)$, mediated through their shared connection to x_c.

For federated learning, the dominant contribution to this sensitivity comes from changes in the central agent — the magnitude of the sensitivity of I_{L_i} to perturbations in x_c dominates the magnitude of its sensitivity to perturbations in other clients' flow variables I_{L_j} directly (i.e., $\left|\frac{\partial I_{L_i}}{\partial x_c}\right| \gg \left|\frac{\partial I_{L_i}}{\partial I_{L_j}}\right|, \forall j \neq i$). This is numerically demonstrated in Appendix 14.

This approximation is justified by how federated learning operates: clients do not communicate directly with one another but only through the central server. Updates from a client affect the global model, which then influences other clients in future rounds. Thus, at each step, the most immediate and significant effect of a client's update is on the central agent, not other clients. This allows us to approximate the sensitivity of the system to I_L^i using only the central agent's response.

This approximation effectively decouples the clients from one another, allowing each to be analyzed independently in relation to the central server. Specifically, in equation (9), each client "sees" only the dynamics of the central agent's capacitor, $\dot{x}_c(t)$, rather than interacting with other clients. As a result, we can model each client branch as an isolated second-order system as:

$$\dot{x}_c^{k+1}(t) = I_{L_i}^{k+1}(t), \tag{22}$$

$$L_i \dot{I}_{L_i}^{k+1}(t) = x_c^{k+1} - I_{L_i}^{k+1}(t)\hat{G}_i^{th^{-1}} - x_i^{k+1}(t) + I_{L_i}^{k}(t)\hat{G}_i^{th^{-1}}. \tag{23}$$

which can be mapped to a series RLC circuit (Figure 5) as derived in Appendix 14.

This decoupled view allows the momentum parameter L_i to be designed independently for each client, without accounting for client interactions. By modeling each client-server connection in isolation, we can independently select L_i to achieve critical damping. Using the RLC circuit model, we derive a closed-form expression for L_i in Appendix 14 that ensures critical damping:

$$L_i = \frac{1}{4}\hat{G}_{th}^{i^{-2}}. \tag{24}$$

The result is a momentum term that guarantees fast, stable convergence for the central agent dynamics.

3.2.2 Adaptive Aggregation Updates

With momentum parameters set in (24), our next goal is to design an adaptive aggregation method for heterogeneous client updates. This involves building an adaptive simulation engine that (a) aligns client updates on a shared time scale to ensure objective consistency, and (b) uses a stable numerical integration scheme that adaptively selects the central agent's step size to accurately follow the ODE.

Interpolating/Extrapolating Heterogeneous Updates to a Synchronous Timescale: Due to client heterogeneity, active clients simulate their local ODEs for different simulation windows, T_i. Without coordinated aggregation, this mismatch can lead to objective inconsistency Wang et al. (2020). To align the

client updates on a synchronous timescale, we uses a linear interpolation/extrapolation operator, $\Gamma(x_i(t), \tau)$, as shown in Agarwal et al. (2025), that evaluates each client's state at intermediate timepoints τ:

$$\Gamma(x_i(t), \tau) = \frac{x_i(t_2) - x_i(t_1)}{t_2 - t_1}(\tau - t_1) + x_i(t_1), \tag{25}$$

This allows the central agent to evaluate its state over a synchronized window $\tau \in [t_0, t_0 + \max(T_i)]$, ensuring client updates are properly aligned. Without this, mismatched timescales would prevent convergence to a shared steady state, as shown in Agarwal et al. (2025). The central agent dynamics are then:

$$\dot{x}_c^{k+1}(\tau) = \sum_{i=1}^{n} I_{L_i}^{k+1}(\tau) \tag{26}$$

$$L_i \dot{I_{L_i}}^{k+1}(\tau) = x_c^{k+1}(\tau) - (I_{L_i}^{k+1}(\tau) G_i^{th^{-1}} + \Gamma(x_i^{k+1}(t), \tau) - I_{L_i}^{k} G_i^{th^{-1}}). \tag{27}$$

Simulating Central Agent via Backward-Euler: Next, the central agent ODE (26)-(27) is numerically simulated to solve for the central agent states. We propose using a numerically stable, Backward-Euler method that defines each aggregation step as:

$$x_c^{k+1}(\tau + \Delta t) = x_c^{k+1}(\tau) - \Delta t \sum_{i=1}^{n} I_{L_i}^{k+1}(\tau + \Delta t), \tag{28}$$

$$I_{L_i}^{k+1}(\tau + \Delta t) = I_{L_i}^{k+1}(\tau) + \frac{\Delta t}{L}\Big(x_c^{k+1}(\tau + \Delta t) - (I_{L_i}^{k+1}(\tau + \Delta t) G_i^{th^{-1}} + \Gamma(x_i^{k+1}(t), \tau + \Delta t) - I_{L_i}^{k}(\tau + \Delta t) G_i^{th^{-1}})\Big). \tag{29}$$

The Backward-Euler method is unconditionally stable as derived in Appendix 9, meaning it converges to the system's steady-state (i.e., critical point of the objective) regardless of the step size Δt.

Adaptive Time-Step Selection Although the BE provides numerical robustness, our objective is to select a Δt that accurately follows the central agent ODE designed for efficient convergence. We use an adaptive time-stepping scheme for the central agent in Agarwal et al. (2025) that selects the step size, Δt, based on the numerical accuracy of the BE rather than treating it as a hyperparameter.

The accuracy of the BE update is measured using the local truncation error (LTE). For the central agent dynamics in (26), the LTE is estimated as:

$$\varepsilon_{BE}^{C} = -\frac{\Delta t}{2}\left[\sum_{i=1}^{n} I_{L_i}^{k+1}(\tau) - \sum_{i=1}^{n} I_{L_i}^{k+1}(\tau + \Delta t)\right]. \tag{30}$$

The LTE for a BE integration of (27) is:

$$\varepsilon_{BE_i}^{L} = -\frac{\Delta t}{2L}\Big[(x_c^{k+1}(t) - I_{L_i}^{k+1}(t)\bar{G}_i^{-1} + x_i^{k+1}(t) - I_{L_i}^{k}(t)\bar{G}_i^{-1}) - (x_c^{k+1}(t + \Delta t) - I_{L_i}^{k+1}(t + \Delta t)\bar{G}_i^{-1} + x_i^{k+1}(t + \Delta t) - I_{L_i}^{k}(t + \Delta t)\bar{G}_i^{-1})\Big]. \tag{31}$$

To ensure accurate simulation, we adaptively choose Δt using a backtracking line search (Appendix 12) such that the maximum LTE across all components remains below a pre-defined tolerance γ:

$$\max |\varepsilon_{BE}| \leq \gamma, \quad \varepsilon_{BE} = [\varepsilon_{BE}^{C}, \varepsilon_{BE}^{L}]. \tag{32}$$

3.3 Adaptive End-to-End Federated Learning

Our method is an adaptive, end-to-end federated learning algorithm designed to address both non-IID data and heterogeneous client computation. Computational heterogeneity is captured through an interpolation/extrapolation operator that aligns client updates, while data heterogeneity is modeled through the aggregate sensitivity matrix, which encodes differences in local curvature induced by non-IID data distributions.

A key element of our approach is the circuit perspective. By framing the system as an equivalent circuit, we can (i) visualize the topology of client–server interactions, and (ii) directly apply circuit design principles to construct the momentum term such that the overall system is critically damped. This perspective also enables us to leverage circuit simulation techniques to select stable step sizes, ensuring the discretized system remains passive and adapts naturally to both client and server gradient spaces.

This construction allows step sizes across clients and the server to be governed by a single global hyperparameter, γ, which controls the accuracy tolerance for numerical integration. While γ is tunable, we show that the algorithm remains numerically stable across its range. The robustness of our method stems directly from the use of Backward-Euler integration. Unlike explicit methods, Backward-Euler is unconditionally stable and guarantees convergence regardless of the step size. As proven in Appendix 9, this property ensures that our algorithm converges to a stable optimum while tracking the continuous-time ODE dynamics designed by the momentum parameter L_i. Robustness is further reinforced by monitoring the local truncation error (LTE), as detailed in Appendix 12, which preserves the intended ODE behavior even under heterogeneous conditions.

The complete algorithm is provided in Appendix 13.

4 Experiments

Our methodology, called Adaptive FedECADO, is an adaptive federated learning method that achieves fast convergence via critical damping, while remaining robust to hyperparameter selection in heterogeneous settings. We study the effectiveness of our approach under heterogeneity in both data and computation. In our experiments, client data distributions follow a non-IID Dirichlet distribution, $|\mathcal{D}i| \sim \text{Dir16}(0.1)$, and each client performs a random number of local training epochs sampled from a uniform distribution according to $e_i \sim U[1, 50]$. In the following experiments, we employ a diagonal approximation of each client's Hessian to compute the aggregate sensitivity matrix $\hat{G}^i_{th}$. This Hessian approximation preserves relative weighting between client parameters and is computed once from a fixed mini-batch of representative points prior to the start of the simulation, and is reused across all subsequent rounds.

In these settings, Adaptive FedECADO achieves high model performance without requiring any manual tuning. In contrast, prior adaptive methods including SCAFFOLD Karimireddy et al. (2020), FedAdam Reddi et al. (2020), FedAdaGrad Reddi et al. (2020), delta-SGD Kim et al. (2023), FedCAda Zhou et al. (2025), and FAFED Wu et al. (2023) are sensitive to hyperparameter selections, leading to instability or poor convergence. Our results show that Adaptive FedECADO outperforms these baselines across multiple datasets and models, with faster convergence across a wide range of hyperparamter values.

Table 1: Percentage of useable models, defined as achieving over 80% of the highest accuracy, across all hyperparameter selections for CIFAR-10, CIFAR-100, and Sentiment-140 datasets.

Dataset (Model)	Adaptive FedECADO	SCAFFOLD	FedAdam	FedAdaGrad	FedExp	deltaSGD	FedCAda	FAFED
CIFAR-10 (Resnet-18)	100	42	15	15	0.05	50	75	90
CIFAR-100 (Resnet-50)	100	15	0.05	0.05	0	30	85	70
Sentiment-140 (VGG-11)	100	82	85	82	90	80	95	85

4.1 Robustness to Hyperparameter Selection

To evaluate robustness to hyperparameter selection, we perform a random search to identify the best-performing hyperparameters for each baseline method and for Adaptive FedECADO individually across multiple datasets and models, including CIFAR-10, CIFAR-100, and Sentiment-140 (shown in Appendix 16).

Figure 2 shows the final classification accuracies for each trial across the hyperparameters ranges reported in Appendix 15.

Adaptive FedECADO is shown to consistently achieve high accuracy with minimal variance, demonstrating strong robustness to its single hyperparameter, γ. In contrast, baseline methods show high sensitivity, performing well only under carefully tuned settings. Nearly all Adaptive FedECADO runs reach near-optimal performance, effectively removing the need for hyperparameter tuning. This is a combined result of (a) selecting momentum parameters that accelerate convergence and (b) adaptive numerical methods that ensure stable convergence. The contribution of each component is quantified through ablation studies in Appendix 21. We further highlight the advantages of our approach over Agarwal et al. (2025), demonstrating that Adaptive FedECADO matches the performance of a fully tuned FedECADO without requiring any manual tuning, eliminating the substantial effort of identifying client-specific momentum and step-size values that ensure stable and efficient convergence. A detailed breakdown of per-round wall-clock time across all baselines, per-step runtime proportions, and communication overhead are provided in Appendix 22. While Adaptive FedECADO introduces a slight increase in computational cost due to the dynamical system formulation and Backward-Euler steps, we argue that this overhead is justified by the method's improved robustness across hyperparameter settings, particularly as extensive hyperparameter tuning itself can incur greater computational cost in practice.

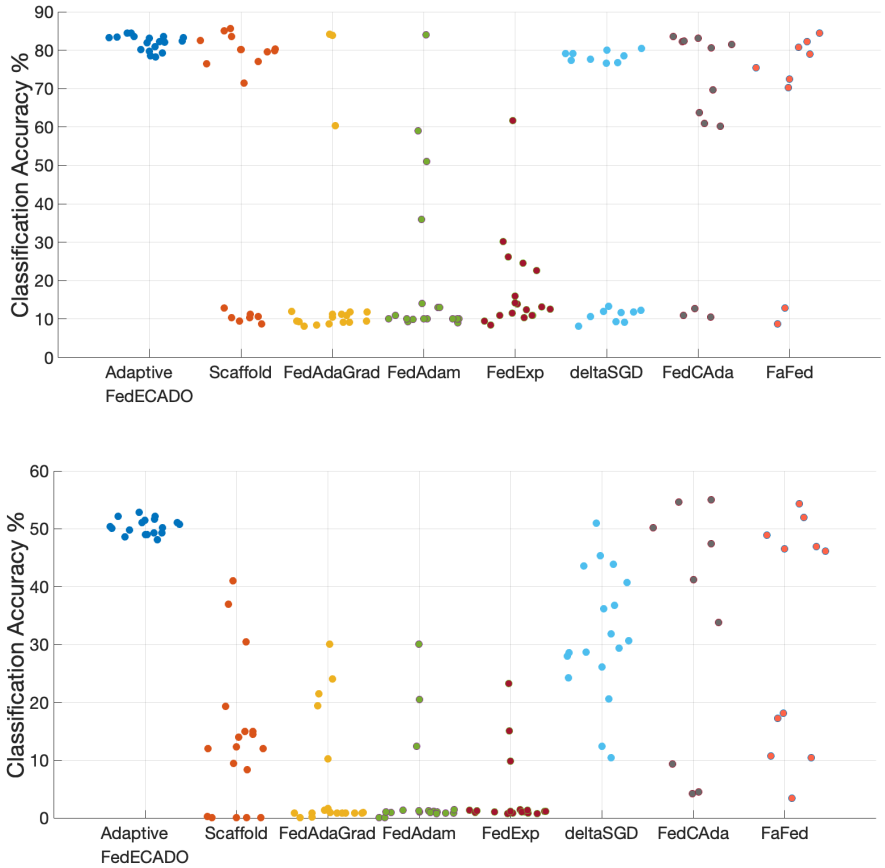

Figure 2: Classification accuracy of Adaptive FedECADO compared to baseline methods across a hyperparameter sweep (via random search for each method) under heterogeneous settings. Results are shown for (a) ResNet-18 on CIFAR-10, (b) ResNet-50 on CIFAR-100.

Usable Rate of Federated Learning Methods: A key advantage of a hyperparameter-free method is its ability to be deployed without additional engineering effort. To demonstrate practical deployability, we examine the number of usable models, defined as those achieving final accuracy above a deployment threshold

of 80% of the highest classification accuracy. Table 1 reports the percentage of hyperparameter configurations that produced usable models for each method.

In all experiments, Adaptive FedECADO achieves a usable rate of 100%, reliably producing high-performance models regardless of hyperparameter choices. In contrast, comparison methods often result in low-performance models that consume significant computational resources during training.

Perturbing Hyperparameter Selections: We also evaluate each method's sensitivity to hyperparameter perturbations by varying optimal values by 20% and measuring the impact on final accuracy (Appendix 17). Adaptive FedECADO remains stable and performant across all perturbations, showing robustness to its single hyperparameter, γ. In contrast, methods like FedAdam and FedAdaGrad exhibit significant performance drops, underscoring their dependence on precise tuning.

Perturbing Hyperparameter Choices in Adaptive FedECADO We further evaluate the convergence behavior of Adaptive FedECADO under varying values of γ. Figure 7 shows training trajectories on the CIFAR-10 dataset using a ResNet-18 model across four orders of magnitude of γ, demonstrating robustness to its sole hyperparameter.

Adding Schedulers to Client Updates Schedulers can be added to client updates; however, they introduce additional hyperparameters can improve stability at the cost of damping convergence speed. The impact of client-side schedulers to FedAdam and FedAdaGrad is shown in Appendix 18.

5 Conclusion

We introduce a fully adaptive federated learning method that eliminates the need for hyperparameter tuning in heterogeneous settings. By modeling federated learning as a dynamical system, our approach derives adaptive momentum and learning rates using principles from critical damping and adaptive simulation. The result is a momentum-based algorithm that achieves fast, stable convergence across non-IID data distributions and varying client compute, all controlled by a single global hyperparameter. We show that our method is highly robust to the choice of this hyperparameter. Compared to existing adaptive methods, Adaptive FedECADO delivers higher classification accuracy without tuning, whereas other methods are highly sensitive to hyperparameter selection. This makes our approach particularly well-suited for rapid prototyping by lowering the engineering overhead typically required to implement federated learning in heterogeneous environments. Future work will investigate Hessian approximations and more frequent curvature updates to better capture evolving client dynamics, as well as evaluation on larger-scale and more complex benchmarks where the client-local adaptivity of the method is expected to be most impactful.

References

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. *arXiv preprint arXiv:2111.04263*, 2021.

Aayushya Agarwal and Larry Pileggi. An equivalent circuit approach to distributed optimization. *arXiv preprint arXiv:2305.14607*, 2023.

Aayushya Agarwal, Carmel Fiscko, Soummya Kar, Larry Pileggi, and Bruno Sinopoli. An equivalent circuit workflow for unconstrained optimization, 2023.

Aayushya Agarwal, Gauri Joshi, and Larry Pileggi. Fedecado: A dynamical system model of federated learning. *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Anant Agarwal and Jeffrey Lang. *Foundations of analog and digital electronic circuits*. Elsevier, 2005.

Zachary Charles and Jakub Konečnỳ. On the outsized importance of learning rates in local update methods. *arXiv preprint arXiv:2007.00878*, 2020.

Jeremy M Cohen, Alex Damian, Ameet Talwalkar, Zico Kolter, and Jason D Lee. Understanding optimization in deep learning with central flows. *arXiv preprint arXiv:2410.24206*, 2024.

Rudrajit Das, Anish Acharya, Abolfazl Hashemi, Sujay Sanghavi, Inderjit S Dhillon, and Ufuk Topcu. Faster non-convex federated learning via global and local momentum. In *Uncertainty in Artificial Intelligence*, pp. 496–506. PMLR, 2022.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Divyansh Jhunjhunwala, Shiqiang Wang, and Gauri Joshi. Fedexp: Speeding up federated averaging via extrapolation. *arXiv preprint arXiv:2301.09604*, 2023.

Divyansh Jhunjhunwala, Shiqiang Wang, and Gauri Joshi. Fedfisher: Leveraging fisher information for one-shot federated learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 1612–1620. PMLR, 2024.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pp. 5132–5143. PMLR, 2020.

Prashant Khanduri, Pranay Sharma, Haibo Yang, Mingyi Hong, Jia Liu, Ketan Rajawat, and Pramod Varshney. Stem: A stochastic two-sided momentum algorithm achieving near-optimal sample and communication complexities for federated learning. *Advances in Neural Information Processing Systems*, 34:6050–6061, 2021.

Junhyung Lyle Kim, Mohammad Taha Toghani, César A Uribe, and Anastasios Kyrillidis. Adaptive federated learning with auto-tuned clients. *arXiv preprint arXiv:2306.11201*, 2023.

Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10713–10722, 2021.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smithy. Feddane: A federated newton-type method. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 1227–1231. IEEE, 2019.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020a.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. 2020b.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*, 2019.

Grigory Malinovsky, Konstantin Mishchenko, and Peter Richtárik. Server-side stepsizes and sampling without replacement provably help in federated optimization. In *Proceedings of the 4th International Workshop on Distributed Machine Learning*, pp. 85–104, 2023.

Reese Pathak and Martin J Wainwright. Fedsplit: An algorithmic framework for fast federated optimization. *Advances in neural information processing systems*, 33:7057–7066, 2020.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

R Rohrer and HASSAN Nosrati. Passivity considerations in stability studies of numerical integration algorithms. *IEEE transactions on circuits and systems*, 28(9):857–866, 1981.

Yong Shi, Yuanying Zhang, Yang Xiao, and Lingfeng Niu. Optimization strategies for client drift in federated learning: a review. *Procedia Computer Science*, 214:1168–1173, 2022.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33: 7611–7623, 2020.

Xidong Wu, Feihu Huang, Zhengmian Hu, and Heng Huang. Faster adaptive federated learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 10379–10387, 2023.

An Xu and Heng Huang. Coordinating momenta for cross-silo federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8735–8743, 2022.

Liuzhi Zhou, Yu He, Kun Zhai, Xiang Liu, Sen Liu, Xingjun Ma, Guangnan Ye, and Hongfeng Chai. Fedcada: Adaptive client-side optimization for accelerated and stable federated learning. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2025.

Appendix

6 Equivalent Circuit Model of Federated Learning

Federated learning is a decentralized optimization framework in which a central server coordinates the training of a global model using gradient updates computed independently on distributed client devices. The global model parameters, x_k at iteration k are updated by gradient descent:

$$x_{k+1} = x_k - \alpha \sum_{i=1}^{N} \nabla f_i(x_k), \tag{33}$$

where $\nabla f_i(x_k)$ is the gradient of the local objective function for client i, and α is a global learning rate. As $\alpha \to 0$, this update can be represented as a time-discretized version of a continuous-time dynamical system:

$$\dot{x}(t) = -\sum_{i=1}^{N} \nabla f_i(x(t)), \tag{34}$$

where $\dot{x}(t)$ denotes the time derivative of the model parameters. While the continuous-time model in (34) offers useful insight into the dynamics of the optimization variables, it has a key limitation in the federated setting: all local clients use the global states, $x(t)$, to compute the local updates $\nabla f_i(x)$, however, in the federated setting, clients compute local updates using their own local states.

To better represent the structure of federated learning, Agarwal et al. (2023) introduced a continuous-time model that separates the global state from the individual client states using auxiliary flow vector, I_{L_i}. In this formulation, the global model, $x_c(t)$, is maintained by the central agent, and each client i maintains its own local state denoted as $x_i(t)$. The interaction between a client and the server is captured through the auxiliary vector $I_{L_i}(t)$. This results in the following dynamical system:

$$\frac{d}{dt}x_c(t) + \sum_{i=1}^{|\mathcal{C}|} I_{L_i}(t) = 0, \tag{35}$$

$$L_i \dot{I_{L_i}}(t) = x_c(t) - x_i(t) \quad \forall i \in [1, |\mathcal{C}|], \tag{36}$$

$$-I_{L_i}(t) + \frac{d}{dt}x_i(t) + p_i \nabla f_i(x_i(t)) = 0 \quad \forall i \in [1, |\mathcal{C}|], \tag{37}$$

where L_i is a scalar parameter that determines the coupling between client i and the central agent.

In this work, we model the dynamical system in (35)-(37) as electrical circuits to leverage physical analysis and simulation techniques to design hyperparameters. Specifically, the dynamical system can be represented by the circuit in Figure 3 which mirrors the structure of the federated learning process. In this circuit, the central server states are modeled as voltages at a central node, $x_c(t)$, connected to node whose voltages represent the client state-variables, $x_i(t)$. These nodes are connected by flow variable $I_{L_i}(t)$ that represents the electrical current between the client and the central node.

In this circuit-based model, each client node is connected to a capacitor, which models the continuous-time dynamics of each client's state. A capacitor is an electrical circuit component that stores energy in an electric field and whose voltage changes based on the net current flowing into it. Mathematically, the current-voltage behavior of a capacitor is described by:

$$I_C = C\dot{V}, \tag{38}$$

where I_C is the current, C is the capacitance, and $\dot{V}$ is the rate of voltage change. In our analogy, the voltage across this capacitor corresponds to the local client states, $x_i(t)$. Each client node is connected to a voltage-controlled current source, which outputs a current equal to the local gradient $p_i \nabla f_i(x_i(t))$.

The client nodes are connected to the central node via inductors. An inductor is another basic circuit element that stores energy in a magnetic field and resists changes in current. It follows the current-voltage relation:

$$V_L = L\dot{I}, \tag{39}$$

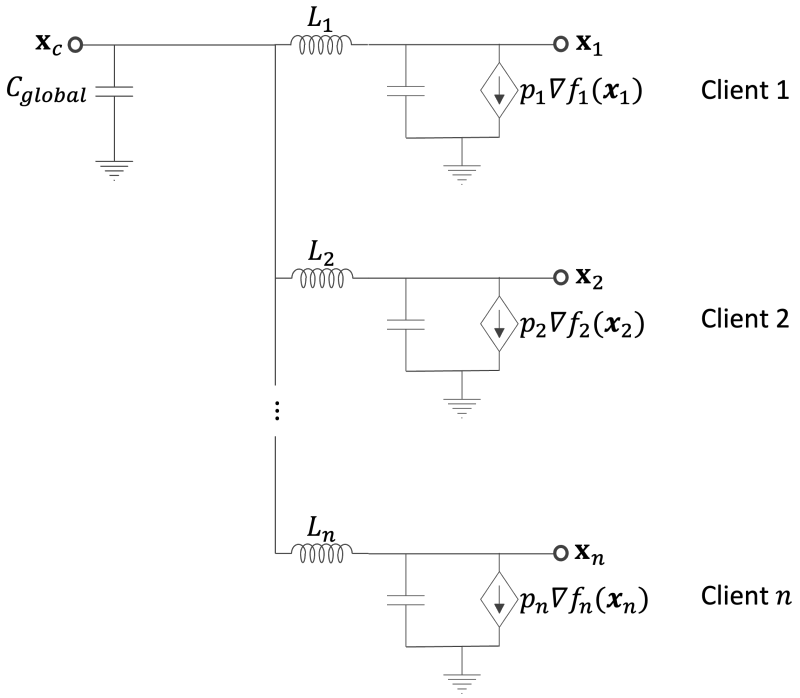

Figure 3: The dynamical system model of federated learning can be represented by an electrical circuit. In the circuit representation, each global state, x_c, is modeled by a node-voltage connected to multiple nodes whose voltage represents the client states, x_i. These nodes are connected by an inductor whose dynamics are modeled by (5).

where V_L is the voltage across the inductor, L is the inductance, and I is the current. In our model, the voltage across the inductor is the difference between the global and client states, $x_c(t) - x_i(t)$, and the resulting current is $I_{L_i}(t)$. This implies that the inductor accumulates the difference between the global and local models over time, scaled by the inductance parameter L_i. Adding the inductors introduces a momentum-like effect into the client-server interaction.

Together, these components form a complete physical circuit, whose behavior is governed by Kirchhoff's Current Law (KCL). KCL is a fundamental principle in circuit theory which states that the total current entering a node must equal the total current leaving it.

Applying KCL at the central node yields the equation $\dot{x}_c(t) = -\sum_i I_{L_i}(t)$, representing the accumulation of all client currents into the global capacitor.

Applying KCL at each client node, the inductor current flowing into each client equals the sum of the capacitor current and the gradient-induced current. This equates to the client ODE $-I_{L_i}(t) + \frac{d}{dt}x_i(t) + p_i \nabla f_i(x_i(t)) = 0$.

By modeling the federated learning system using these physical components, we obtain a circuit-based representation that is structurally identical to the federated learning process. This perspective enables us to derive adaptive step sizes and momentum terms directly from physical principles, rather than relying on empirical tuning or heuristic modifications.

6.1 Modeling the Central Agent Aggregation as a Circuit

The continuous-time dynamics at aggregation round $k+1$, which can be described by the following system of equations:

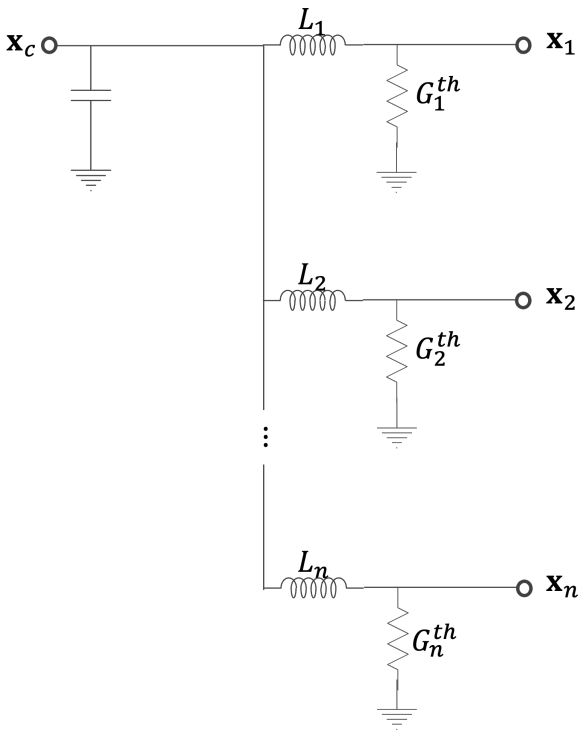

Figure 4: Equivalent circuit represented of the aggregation step for central agents in (8)-(9)

$$\dot{x}_c^{k+1}(t) = \sum_{i=1}^{n} I_{L_i}^{k+1}(t), \tag{40}$$

$$\dot{LI}_{L_i}^{k+1}(t) = x_c^{k+1}(t) - \left(I_{L_i}^{k+1}(t)\hat{G}_i^{\text{th}^{-1}} + x_i^{k+1}(t) - I_{L_i}^{k}(t)\hat{G}_i^{\text{th}^{-1}} \right), \tag{41}$$

where $\hat{G}_i^{\text{th}}$ represents a linear sensitivity of each client, and L_i is the momentum term for each client branch. This set of linear ODEs can be modeled by an electrical circuit shown in Figure 4.

This circuit differs from the previous nonlinear circuit representation of Figure 3 in that the client models are linearized via a first-order approximation using the sensitivity matrix $\hat{G}_i^{\text{th}}$. Physically, the sensitivity term, $\hat{G}_i^{\text{th}}$, behaves like a **resistor** in the circuit, which relates voltage and current linearly through Ohm's Law. This linear sensitivity is derived using a circuit concept known as **Thevenin impedance**, which models the sensitivity of current of a circuit network with respect to the node-voltage as an impedance and is used to simplify the analysis of linear electrical networks.

6.1.1 Client Perspective via Thevenin Impedance

The linear circuit in Figure 4 presents a complex interconnected network where the clients and central server states are coupled. The coupled interactions makes it challenging to derive momentum term, L_i, that makes the entire linear circuit critically damped.

Instead, we study the behavior of each individual client during aggregation using a *Thevenin impedance* looking outward from a single client branch into the rest of the circuit. From the client's perspective, this impedance includes the contribution of the central agent capacitor as well as all the other clients' branches connected in parallel. In our experiments we find that this Thevenin impedance is dominated by the central agent's capacitor, due to its relatively large capacitance compared to the resistance and inductance in the client branches.

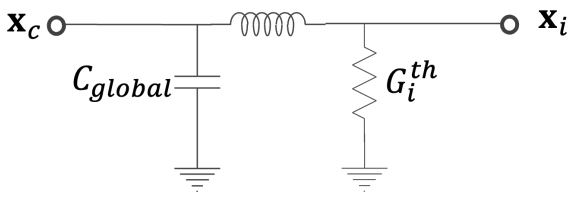

Figure 5: Each client branch in the aggregation circuit (Figure 4) can be reduced to a series RLC since the Thevenin impedance looking out of each client branch effectively looks like a capacitance for the central agent.

This observation allows us to approximate the Thevenin impedance looking out of the client branch as the central agent capacitor. As a result, each client branch looks like a **series RLC circuit**, consisting of:

- a resistor $R = \hat{G}_i^{\text{th}^{-1}}$, representing the client's linearized sensitivity,
- an inductor L_i, representing the momentum term, and
- a capacitor $C_{global} = 1$, representing the central agent's state.

This reduced model is depicted in Figure 5. The series RLC circuit abstraction provides a highly interpretable local model for each client, enabling us to analyze convergence, damping, and oscillation behavior using well-known results from second-order systems. By modeling each client-server interaction as a damped oscillator, we can derive stability conditions and design update rules with predictable behavior, without relying on heuristic hyperparameter tuning.

6.2 Series RLC Behavior and Designing for Critical Damping

Each client-server connection in our federated learning framework can be modeled as a series RLC circuit. The behavior of this circuit is governed by a second-order differential equation that can be damped by tuning the inductance, L_i, as shown by Figure 1. Ideally, we want the system to be critically damped, meaning it returns to equilibrium as quickly as possible without oscillation. In the context of federated optimization, this corresponds to designing a momentum term for the aggregation step that achieves fast global convergence without overshooting.

To achieve critical damping in a series RLC circuit, we must design the inductance L_i relative to the client sensitivity, $\hat{G}_i^{th}$, and the capacitance $C_{global} = 1$. We begin by recalling that the differential equation governing the series RLC circuit is:

$$L_i \ddot{q}(t) + R\dot{q}(t) + \frac{1}{C} q(t) = 0, \tag{42}$$

where $q(t)$ is the charge on the capacitor, $\dot{q}(t)$ is the capacitor current , L_i is the inductance, $R = \hat{G}_i^{th^{-1}}$ is the resistance, and $C = 1$ is the capacitance. This equation has the same form as a damped second-order linear system and is derived in circuit literature Agarwal & Lang (2005). From this, we can derive a damping ratio ζ, which determines whether the system is underdamped, critically damped, or overdamped, is given by:

$$\zeta = \frac{R}{2}\sqrt{\frac{C}{L_i}}. \tag{43}$$

Critical damping occurs when $\zeta = 1$, leading to the condition:

$$1 = \frac{R}{2}\sqrt{\frac{C}{L_i}}. \tag{44}$$

Solving for L_i, we square both sides:

$$1 = \frac{R^2 C}{4L_i} \quad \Rightarrow \quad L_i = \frac{CR^2}{4}. \tag{45}$$

Thus, to achieve critical damping, the inductance must be set according to:

$$L_i = \frac{CR^2}{4} = \frac{1}{4}\hat{G}_i^{th^{-2}}. \tag{46}$$

This equation gives us a principled way to design the inductance (i.e., momentum term) based on the client sensitivity $\hat{G}_i^{th}$. Rather than relying on heuristics or manual tuning, we can compute the inductance directly to ensure optimal convergence behavior. In practice, this means that each client can be configured with a momentum that improves convergence speed in the global learning process.

7 Derivation of Client Update Rule from Numerical Simulation

The client update rule in (13) is obtained by applying a Forward-Euler (FE) numerical integration scheme to the coupled client ODE system. We walk through this derivation pointwise below.

Step 1: The Coupled Client ODE System. The client state $x_i(t)$ and inductor current $I_{L_i}(t)$ evolve according to the following coupled system of ODEs:

$$\dot{x}_i^{k+1}(t) = -p_i \nabla f_i(x_i^{k+1}(t)) - I_{L_i}^k(t), \tag{47}$$

$$L_i \dot{I}_{L_i}^{k+1}(t) = x_c^{k+1} - I_{L_i}^{k+1}(t)\hat{G}_i^{th^{-1}} - x_i^{k+1}(t) + I_{L_i}^k(t)\hat{G}_i^{th^{-1}}, \tag{48}$$

where $x_i(t)$ is the client model state, x_c is the central server state, $I_{L_i}(t)$ is the inductor current representing the momentum state of client i, p_i is a client-specific scaling parameter, L_i is the inductance governing the momentum dynamics, and $\hat{G}_i^{th}$ is the client's aggregate sensitivity. The goal is to accurately simulate the evolution of this system forward in continuous time using discrete time-steps.

Step 2: Integral Form Over One Time-Step. The exact solution for x_i over one time-step of size Δt_i is given by:

$$x_i(t + \Delta t_i) = x_i(t) + \int_t^{t+\Delta t_i} \left(-p_i \nabla f_i(x_i(\tau), \mathcal{D}_i) - I_{L_i}^k(\tau)\right) d\tau. \tag{49}$$

In general, this integral does not admit a closed-form solution due to the nonlinearity of ∇f_i.

Step 3: Forward-Euler Approximation. From numerical simulation, the integral in (49) can be approximated by using an explicit first-order Forward-Euler scheme, which evaluates the integrand at the current time point t as:

$$\int_t^{t+\Delta t_i} \left(-p_i \nabla f_i(x_i(\tau), \mathcal{D}_i) - I_{L_i}^k(\tau)\right) d\tau \approx \Delta t_i \left(-p_i \nabla f_i(x_i(t), \mathcal{D}_i) - I_{L_i}^k(t)\right). \tag{50}$$

Substituting this approximation into (49) yields:

$$x_i(t + \Delta t_i) = x_i(t) - \Delta t_i \left(p_i \nabla f_i(x_i(t), \mathcal{D}_i) + I_{L_i}^k(t)\right). \tag{51}$$

Step 4: Correspondence to Gradient Descent. Equation (51) reveals that the Forward-Euler discretization of the client ODE takes the form of a momentum-augmented gradient descent step, where Δt_i plays the role of the learning rate and $I_{L_i}^k(t)$ contributes a momentum correction term. Crucially, Δt_i is not a manually specified learning rate but is instead determined by the numerical integration scheme: it is the time-step size required to accurately simulate the client ODE, and is set adaptively via local truncation error (LTE) control. This is the key distinction from standard gradient descent — the learning rate emerges from the simulation rather than being tuned by the practitioner.

8 Derivation of Local Truncation Error for Forward-Euler

To numerically solve the client ODE in (7), we apply the Forward Euler method with time step Δt. The discretized update rule is:

$$x_i(t + \Delta t) = x_i(t) - \Delta t \left(\nabla f(x_i(t)) + I_{L_i}(t)\right), \tag{52}$$

The local truncation error (LTE) measures the error introduced in a single step of the numerical method, defined as:

$$\varepsilon_{FE} = \frac{1}{\Delta t}\left[x_i(t + \Delta t) - x_i(t) + \Delta t \left(\nabla f(x_i(t)) + I_{L_i}(t)\right)\right]. \tag{53}$$

To compute this, we expand $x_i(t + \Delta t)$ using a Taylor series about t:

$$x_i(t + \Delta t) = x_i(t) + \Delta t \dot{x}_i(t) + \frac{\Delta t^2}{2}\ddot{x}_i(t) + \mathcal{O}(\Delta t^3). \tag{54}$$

From the ODE (7), we know:

$$\dot{x}_i(t) = -\nabla f(x_i(t)) - I_{L_i}(t). \tag{55}$$

Substituting into the expansion:

$$x_i(t + \Delta t) = x_i(t) - \Delta t \left(\nabla f(x_i(t)) + I_{L_i}(t)\right) + \frac{\Delta t^2}{2}\ddot{x}_i(t) + \mathcal{O}(\Delta t^3). \tag{56}$$

Now, substituting into the expression for ε_{FE}:

$$\begin{aligned}
\varepsilon_{FE_n} &= \frac{1}{\Delta t}\left[-\Delta t \left(\nabla f(x_i(t)) + I_{L_i}(t)\right) + \frac{\Delta t^2}{2}\ddot{x}_i(t) + \mathcal{O}(\Delta t^3) + \Delta t \left(\nabla f(x_i(t)) + I_{L_i}(t)\right)\right] \\
&= \frac{1}{\Delta t}\left[\frac{\Delta t^2}{2}\ddot{x}_i(t) + \mathcal{O}(\Delta t^3)\right] \\
&= \frac{\Delta t}{2}\ddot{x}_i(t) + \mathcal{O}(\Delta t^2).
\end{aligned} \tag{57}$$

In the expression, $\ddot{x}_i(t)$ can be evaluated as:

$$\ddot{x}_i(t) = I_{L_i}(t) + \nabla f_i(x_i(t)) - I_{L_i}(t - \Delta t) - \nabla f_i(x_i(t - \Delta t)). \tag{58}$$

Using this expression, the LTE for Forward-Euler is approximated by the first-order term as:

$$\varepsilon_{FE_n} = \frac{\Delta t}{2}\left[I_{L_i}(t) + \nabla f_i(x_i(t)) - I_{L_i}(t - \Delta t) - \nabla f_i(x_i(t - \Delta t))\right]. \tag{59}$$

9 Derivation for Backward-Euler Accuracy and Stability

The aggregation of client updates is modeled by a linear set of ODEs in (8)-(9). To numerically determine the central agent state, we discretize these ODEs using the Backward Euler method with a time step of Δt. The numerical update is:

$$x_c^{k+1}(\tau + \Delta t) = x_c^{k+1}(\tau) - \Delta t \sum_{i=1}^{n} I_{L_i}^{k+1}(\tau + \Delta t), \tag{60}$$

$$\begin{aligned} I_{L_i}^{k+1}(\tau + \Delta t) = I_{L_i}^{k+1}(\tau) + \frac{\Delta t}{L}\Big(x_c^{k+1}(\tau + \Delta t) - (I_{L_i}^{k+1}(\tau + \Delta t) G_i^{th^{-1}} \\ + \Gamma(x_i^{k+1}(t), \tau + \Delta t) - I_{L_i}^{k}(\tau + \Delta t) G_i^{th^{-1}})\Big). \end{aligned} \tag{61}$$

To compute the local truncation error, we assume the previous value is exact and define the error for the first equation as:

$$\varepsilon_{BE}^{C} = \frac{1}{\Delta t}\left(x_c^{k+1}(t + \Delta t) - x_c^{k+1}(t) - \Delta t \sum_i I_{L_i}^{k+1}(t + \Delta t)\right), \tag{62}$$

and for the second equation:

$$\begin{aligned} \varepsilon_{BE}^{L} = \frac{1}{\Delta t}\big(L_i(I_{L_i}^{k+1}(t + \Delta t) - I_{L_i}^{k+1}(t)) \\ -\Delta t\left[x_c^{k+1}(t + \Delta t) - \left(I_{L_i}^{k+1}(t + \Delta t) G_i^{\mathrm{th}^{-1}} + \Gamma(x_i^{k+1}(t + \Delta t), \Delta t) - I_{L_i}^{k} G_i^{\mathrm{th}^{-1}}\right)\right]\Big). \end{aligned} \tag{63}$$

Expanding the exact solutions using Taylor series:

$$x_c^{k+1}(t + \Delta t) = x_c^{k+1}(t) + \Delta t \dot{x}_c^{k+1}(t) + \frac{(\Delta t)^2}{2}\ddot{x}_c^{k+1}(t) + \mathcal{O}((\Delta t)^3), \tag{64}$$

$$I_{L_i}^{k+1}(t + \Delta t) = I_{L_i}^{k+1}(t) + \Delta t \dot{I_{L_i}}^{k+1}(t) + \frac{(\Delta t)^2}{2}\ddot{I_{L_i}}^{k+1}(t) + \mathcal{O}((\Delta t)^3). \tag{65}$$

Substituting into ε_{BE}^{C}:

$$\begin{aligned} \varepsilon_{BE}^{C} &= \frac{1}{\Delta t}\left(\Delta t \dot{x}_c^{k+1}(t) + \frac{(\Delta t)^2}{2}\ddot{x}_c^{k+1}(t) - \Delta t \sum_i I_{L_i}^{k+1}(t) - (\Delta t)^2 \sum_i \dot{I_{L_i}}^{k+1}(t) + \mathcal{O}((\Delta t)^3)\right) \\ &= \left(\dot{x}_c^{k+1}(t) - \sum_i I_{L_i}^{k+1}(t)\right) + \mathcal{O}(\Delta t). \end{aligned} \tag{66}$$

Using the ODE definition, $\dot{x}_c^{k+1}(t) = \sum_i I_{L_i}^{k+1}(t)$, so:

$$\tau_n^{(x)} = \mathcal{O}(\Delta t). \tag{67}$$

A similar analysis for $\tau_n^{(I_{L_i})}$ shows that it is also $\mathcal{O}(\Delta t)$. Therefore, Backward Euler remains first-order accurate when applied to this linear system.

9.1 Numerical Stability of Backward Euler

The Backward-Euler integration is a numerically stable algorithm, which means for a linear system, we can guarantee convergence to the steady-state (or stationary point) regardless of the choice of Δt. To analyze property of numerical stability, we examine the linear system:

$$\dot{y}(t) = Ay(t), \tag{68}$$

where $A \succ 0$ is a linear system. The Backward Euler update is:

$$y_{n+1} = (I - \Delta t A)^{-1} y_n. \tag{69}$$

This scheme is stable if $eig(I - \Delta t A)^{-1} \leq 1$. For all $A \succ 0$, the value of $eig(I - \Delta t A)^{-1}$ is less than one. This implies that Backward Euler is **A-stable**: it remains stable for all step sizes $\Delta t > 0$, in the presence of stiff dynamics.

In the context of our aggregation ODE system, where the linear dynamics are governed by parameters $G_i^{\text{th}} \succ 0$ and $L_i > 0$, the eigenvalues of the linearized operator have negative real parts. Therefore, the Backward Euler method guarantees stability regardless of the time step Δt, making it a robust choice for simulating the dynamics of client-server interactions in heterogeneous and stiff federated learning systems.

10 Derivation of Damping Condition

Consider the linear set of ODEs representing the central agent aggregation (8)-(9). We abstract the linear ODEs to the form:

$$C\dot{x}(t) = Gx(t), \tag{70}$$

where $x(t) = \begin{bmatrix} x_c(t) \\ I_L(t) \end{bmatrix}$, $C = \begin{bmatrix} \mathcal{I}, 0 \\ 0, L \end{bmatrix}$ is a symmetric positive definite matrix representing capacitance and inductances. $G = \begin{bmatrix} 0, 0, ... \\ 0, \hat{G}_i^{th}, 0 \\ 0, 0, \ddots \end{bmatrix}$ is a positive semi-definite matrix representing client sensitivities.

Rewriting this system in standard form gives:

$$\dot{x}(t) = C^{-1} G x(t). \tag{71}$$

The dynamics of this system are governed by the eigenvalues of the matrix $C^{-1}G$. Let $\lambda_i \in \mathbb{C}$ denote the eigenvalues of $C^{-1}G$. These eigenvalues determine the transient behavior of the system: real negative eigenvalues lead to exponential decay, while complex eigenvalues with nonzero imaginary parts introduce oscillations.

In control theory, a system is said to be **critically damped** when it returns to equilibrium as quickly as possible without oscillation. For a linear system, this corresponds to all eigenvalues of the system matrix being real and negative.

In the context of Equation (70), achieving critical damping involves two primary goals:

1. **Ensure all eigenvalues of $C^{-1}G$ are real and non-positive.** This removes imaginary components to avoid oscillations in the solution trajectories.
2. **Maximize the smallest (least negative) real eigenvalue of $C^{-1}G$.** This improves the convergence rate of the slowest mode and ensures the system returns to equilibrium quickly and uniformly.

Formally, if $\Lambda(C^{-1}G) = \{\lambda_1, \ldots, \lambda_d\} \subset \mathbb{R}$ denotes the eigenvalues of $C^{-1}G$, we aim to solve:

$$\max_{L \succ 0} \quad \inf_i \ \text{Re}(\lambda_i(C^{-1}G)) \quad \text{subject to} \quad \text{Im}(\lambda_i(C^{-1}G)) = 0, \ \forall i. \tag{72}$$

This objective seeks the choice of inductance L that spreads the eigenvalues of $C^{-1}G$ onto the negative real axis and pushes smallest $|\lambda_i|$ as far left as possible.

11 Adaptive Step Size Selection for Client Steps

The adaptive step-size selection procedure for an individual client is outlined in Algorithm 1. The algorithm is initialized with a time step Δt based on the passivity conditions defined in Equation (16). A backtracking line search (line 4) is then employed to iteratively adjust the step size, ensuring that the local truncation error of the Forward-Euler integration for each client's ODE remains within the prescribed global tolerance γ (line 3). The backtracking line-search damps the time-step according to the ratio, $0 < \frac{\gamma}{\max|\varepsilon_{FE}|} < 1$, which scales the step-size by how far the local truncation error is from the tolerance, γ.

Algorithm 1 Adaptive Step Size Selection for Client Step

1: **Input:** $x_i^{k+1}(t)$, $I_{L_i}^k(t)$,$x_c^{k+1}(t)$, $\gamma > 0$
2: $\Delta t \leftarrow Equation(16)$
3: **while** $\max|\varepsilon_{FE}| \geq \gamma$ **do**
4: $\quad \Delta t = \frac{\gamma}{\max|\varepsilon_{FE}|}\Delta t$
5: $\quad \varepsilon_{FE} \leftarrow$ Equation (13)(Δt)
6: **end while**
7: **return** Δt

12 Adaptive Step Size Selection for Central Agent Aggregation

The adaptive step-size selection of the central agent aggregation is shown in Algorithm 2. This algorithm uses a backtracking line-search in line 4 to adaptive select a step-size that ensures the local truncation error of the Backward-Euler integration of the central agent ODEs are within a global tolerance, γ (line 3).

Algorithm 2 Adaptive Step Size Selection for Central Agent Aggregation

1: **Input:** $x_c^{k+1}(t)$, $I_{L_i}^k(t)$,$x_i^{k+1}(t)$, $\gamma > 0$, Δt_k
2: $\Delta t \leftarrow (\Delta t)_k$
3: **while** $\max|\varepsilon_{BE}| \geq \gamma$ **do**
4: $\quad \Delta t = \frac{\gamma}{\max|\varepsilon_{BE}|}\Delta t$
5: $\quad \varepsilon_{BE} \leftarrow$ (30), (31)
6: **end while**
7: **return** Δt

13 Adaptive Federated Learning Algorithm

The workflow for Adaptive FedECADO is shown in Algorithm 3. The input to the algorithm include each client's loss function and a global, scalar hyperparamter, $\gamma > 0$ that controls the local truncation errors for the central agent aggregation and all client simulations.

Prior to beginning the client communication, Adaptive FedECADO computes each client's sensitivity in line 5 as well as the corresponding momentum parameters in line 6. Then, a subset of active clients perform local updates using a Forward-Euler integration as shown in line 13, where the local step sizes adapt according to Algorithm 1. Each active client then communicates the final state vector and simulation window in line 14 to the central server. The aggregation of the client updates are then performed in line 18-19 according to a Backward-Euler integration of the central agent ODEs. The step-sizes for the central agent are determined using Algorithm 2, which is designed to control the local truncation error within a predefined tolerance, γ.

Algorithm 3 Adaptive FedECADO Algorithm

Input: $\nabla f_i(\cdot)$,$x(0)$, $\gamma > 0$

$x_c \leftarrow x(0)$
$x_i \leftarrow x(0)$
$I_i^L \leftarrow 0$
$t \leftarrow 0$
Precompute $\bar{G}_i^{th}$ $\forall i \in C$ via (11)
Precompute L_i $\forall i \in C$ via (24)
do while $\|\dot{x}_c\|^2 > 0$
  $x_c^k \leftarrow x_c^{k+1}$
  $x_i^k \leftarrow x^{i^{k+1}}$
  Parallel Solve for active client states, $x_i^{k+1}(t + T_i) \forall i \in C_a$*, by simulating:*
    for e_i **epochs:**
      $\Delta t_i \leftarrow$ Algorithm 1
      $x_i^{k+1}(t + \Delta t_i) = x_i^{k+1}(t) - \Delta t_i \nabla f(x_i^{k+1}(t)) - \Delta t_i I_i^{L^k}(t)$
    Communicate $x_i(T_i)$ and T_i to central server
  Solve central agent aggregation by simulating:
  for $\tau \in [t, t + \max(T_i)]$
    Select Δt according to Algorithm 2
    Evaluate active client states at timepoint τ: $\Gamma(x_i^{k+1}, \tau)$ $\forall i \in C_a$
    Solve for $x_c^{k+1}(\tau + \Delta t), I_i^{L^{k+1}}(\tau + \Delta t)$ according to (28)-(29)
    $\tau = \tau + \Delta t$
Return x_c

14 Estimating Critical Damping

In Section 3.2.1, we develop an approximate solution to (18) that relies on the structure of the analog circuit model of federated learning. The approximation is developed by concluding that the sensitivity of I_{L_i} with respect to the global state variable, x_c, is mainly attributed to the dynamics of the global state, $\frac{d}{dx_c}\dot{x}_c$.

To connect this to the sensitivity decomposition in Section 3.2.1, we recall that from equation (8), the coupling variable I_{L_i} can be written as:

$$I_{L_i}(t) = \dot{x}_c(t) - \sum_{j \neq i} I_{L_j}(t). \tag{73}$$

This reveals two sources of coupling for client i: a coupling to the central agent dynamics through $\dot{x}_c(t)$, and a direct coupling to all other clients through $\sum_{j \neq i} I_{L_j}(t)$. The approximation in Section 3.2.1 is justified if the sensitivity of I_{L_i} to perturbations in x_c dominates its sensitivity to perturbations in the other clients' flow variables I_{L_j} $(j \neq i)$ directly. The negligible sensitivities $\frac{\partial I_{L_1}}{\partial I_{L_2}}$ and $\frac{\partial I_{L_1}}{\partial I_{L_3}}$ in Table 2 directly correspond to the cross-client terms $\sum_{j \neq i} \frac{\partial I_{L_j}}{\partial x_c}$ dropped in the sensitivity decomposition of Section 3.2.1. Their negligible magnitude empirically validates the decoupling assumption, confirming that the influence of other clients on I_{L_i} is negligible relative to the central agent.

We can demonstrate this numerically by training a ResNet-18 model on the CIFAR-10 dataset through the federated setting across 3 clients. In this experiment, we create the linearized aggregation model in (8)-(9). Then we study the sensitivity of the flow variable of the first client, I_{L_1}, due to perturbations in the central agent state, x_c, and the flow variables of the other clients, I_{L_2} and I_{L_3}. Table 2 demonstrates the normalized sensitivities across all state variables. We observe that the dominant sensitivity is due to x_c as opposed to changes in the other clients' flow variables, with $\frac{\partial I_{L_1}}{\partial x_c} = 0.97 \gg \frac{\partial I_{L_1}}{\partial I_{L_2}} = 0.014$ and $\frac{\partial I_{L_1}}{\partial I_{L_3}} = 0.016$. This experimentally justifies that the dominant sensitivity term of any client is due to changes in the central agent state, and that cross-client coupling is negligible by comparison, thus allowing us to make the decoupling approximation in Section 3.2.1.

Perturbation Variable	Normalized Sensitivity of I_{L_1} to Perturbation Variable
x_c	0.97
I_{L_2}	0.014
I_{L_3}	0.016

Table 2: Sensitivity analysis of a single client flow vector I_{L_1} in the central agent aggregation dynamics (8)–(9). We perturb the central agent state x_c and the flow variables of clients 2 and 3, I_{L_2} and I_{L_3}, and measure the resulting change in I_{L_1}. Sensitivities are computed as partial derivatives and normalized by the sum of all absolute sensitivities: $|\frac{\partial I_{L_1}}{\partial x_c}| + |\frac{\partial I_{L_1}}{\partial I_{L_2}}| + |\frac{\partial I_{L_1}}{\partial I_{L_3}}|$. The negligible sensitivities to I_{L_2} and I_{L_3} directly correspond to the cross-client terms dropped in the approximation of Section 3.2.1, empirically validating the decoupling assumption.

15 Varying Hyperparameters in Federated Learning Methods

We vary each federated learning method's hyperparameters to study their sensitivity to the model performance. The hyperparameters are randomly selected within the bounds listed in the following tables.

Hyperparameter	Minimum Value	Maximum Value
LTE tolerance (γ)	0	10^6

Table 3: Hyperparameter ranges for Adaptive FedECADO

Hyperparameter	Minimum Value	Maximum Value
Global Step Size	0	1
Local Step Size	0	1
Initial control variate (c)	0	1

Table 4: Hyperparameter ranges for SCAFFOLD

Hyperparameter	Minimum Value	Maximum Value
Local Step Size	0	1

Table 5: Hyperparameter ranges for FedExp

Hyperparameter	Minimum Value	Maximum Value
Global Step Size	0	1
Local Step Size	0	1
Momentum Factor β_1	0.9	1
Momentum Factor β_2	0.9	1

Table 6: Hyperparameter ranges for FedAdam

Hyperparameter	Minimum Value	Maximum Value
Global Step Size	0	1
Local Step Size	0	1
Momentum Factor β	0.9	1

Table 7: Hyperparameter ranges for FedAdaGrad

16 Heterogeneous Federated Learning for Sentiment-140

We train a VGG-11 model on a Sentiment-140 dataset across 50 clients with an active client particition of 20%. We vary the hyperparameters of each adaptive federated learning method according to Appendix 15 and denote the final classification accuracies in Figure 6.

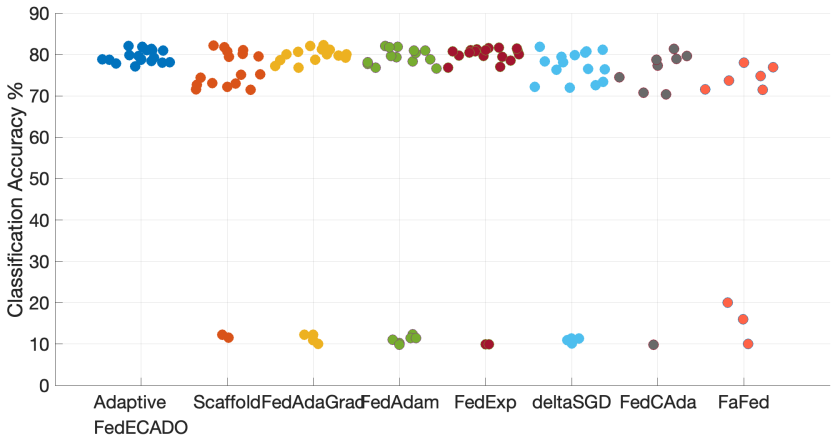

Figure 6: Hyperparameters for Adaptive FedECADO, SCAFFOLD, FedAdam, FedAdaGrad, FedExp and deltaSGD are swept during heterogeneous training for LSTM model for Sentiment140 dataset. The resulting classification accuracies of the random sweep are illustrated

Dataset (Model)	Adaptive FedECADO	SCAFFOLD	FedAdam	FedAdaGrad	FedExp	deltaSGD	FedCAda	FaFed
CIFAR-10 (ResNet-18)	81.9 (2.1)	54.5 (34.6)	21.2 (25.8)	20.4 (21.3)	27.7 (12.6)	44.7 (34.7)	60.1 (29.2)	62.9 (29.9)
CIFAR-100 (ResNet-50)	50.4 (1.37)	13.4 (12.3)	6.5 (1.2)	4.31 (8.21)	3.6 (6.2)	31.6 (10.9)	33.3 (21.6)	32.2 (19.7)
Sentiment-140 (VGG-11)	79.6 (1.49)	73.32 (21.2)	64.5 (29.3)	62.4 (29.1)	72.7 (22.9)	63.4 (27.8)	69.1 (22.5)	54.7 (29.7)

Table 8: Mean (Standard Deviation) of classification accuracies across all hyperparameter selections for training CIFAR-10, CIFAR-100, and Sentiment-140 datasets.

17 Perturbing Hyperparameter Selections

We study the effect of perturbing hyperparameters from their optimal values. The optimal hyperparemter selections for training CIFAR-10 on ResNet-18 model is determined from a hyperparameter sweep, with the final test accuracies of the sweep presented in Figure 2. We then perturb the hyperparameter values by 20% to see the effect on the model performance. As shown in Table 9, Adaptive FedECADO is highly insensitive to any perturbations where as many comparison methods can have large model performances.

The insensitivity of Adaptive FedECADO's single hyperparameter (γ) is especially highlighted by varying the value of γ by four orders of magnitude in Figure 7, where we observe stable and efficient global convergence. The insensitivity to γ is partly due to using a numerically stable Backward-Euler integration method for aggregating client updates.

CIFAR-10 (Resnet-18)	Adaptive FedECADO	SCAFFOLD	FedAdam	FedAdaGrad	FedExp	deltaSGD	FedCAda	FaFed
Mean (Std)	84.5 (1.3)	85.1 (3.2)	75.1 (10.2)	70.4 (8.8)	80.4 (6.4)	82.2 (5.7)	82.3 (6.4)	80.9 (4.9)

Table 9: Distribution of classification accuracies of federated learning methods trained under heterogeneous settings with hyperparameters randomly selected within 20% of the optimal values.

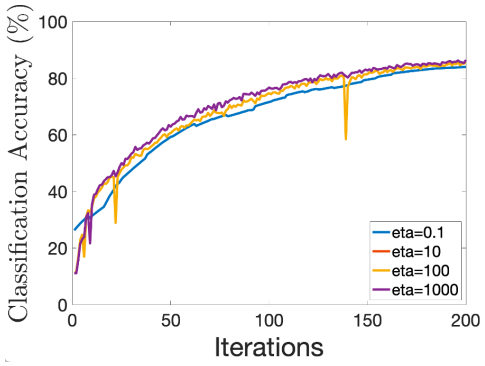

Figure 7: The hyperparameter of Adaptive FedECADO, γ is swept across four order of magnitude in training CIFAR-10 dataset with ResNet-18 model under heterogeneous conditions. A wide range of γ results in similar convergence plots.

18 Effect of Client Schedulers

In non-federated settings, schedulers are often used to adjust learning rates to achieve stable convergence. However, the challenge in including schedulers for each client in federated settings is that this creates non-uniform step sizes that requires proper coordination with the central agent aggregation step. Additionally, schedulers introduce their own hyperparameters that influence convergence rate, creating larger space of potential hyperparameter selections.

To demonstrate the challenge of introducing schedulers, we add exponential learning rate scheduler Li & Arora (2019) to client updates of FedAdaGrad and FedAdam aggregations and perform a random search of hyperparameter selections. The results, shown in Figure 8, indicate that schedulers results in fewer divergent cases, but slow the overall convergence for achieving a high-performing model. The hyperparameter for the exponential learning rate scheduler (γ) is randomly sampled from a uniform distribution, U as $\gamma \sim U[0, 1]$.

19 Approximating Hessian Computation

Adaptive FedECADO requires computing the Hessian of each client via a smaller sample set before training to compute the optimal inductance parameters, L_r. However, computing the full Hessian may become computationally infeasible for large-scale models. In such circumstances, we can extend our framework to use Hessian approximations such as diagonal Fisher matrix Jhunjhunwala et al. (2024). We note that the main goal of the Hessian is not to accurately characterize the loss function, but is instead used as a relative weighting between clients. As a result, approximate hessian methods can still be applied with relatively high accuracy.

In this experiment, we study the the robustness of Adaptive FedECADO for training CIFAR-10 dataset on ResNet-18 model with random selections of LTE tolerance, η using both full Hessian and approximate Hessian via Fischer matrix Jhunjhunwala et al. (2024). Figure 9 demonstrates that approximate Hessians offer similar performance to that of full Hessian. Our work provides a foundation for applying Hessian approximations in large models toward automated hyperparameter selection.

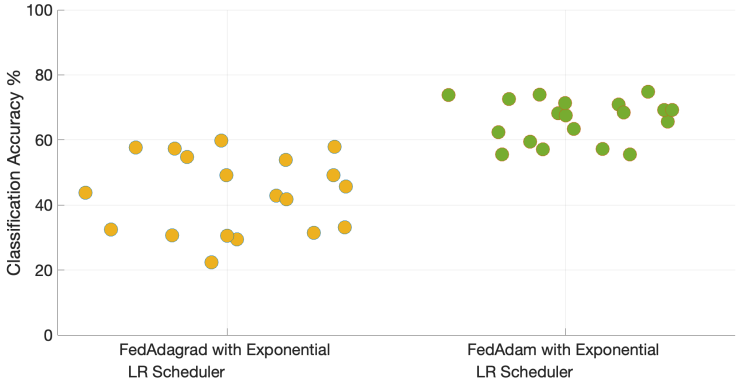

Figure 8: An exponential learning rate scheduler is added to the client updates with server aggregation performed by FedAdam and FedAdagrad Reddi et al. (2020). The hyperparameters of the server aggregation and client schedulers are randomly selected and the final classification accuracies of each run is shown.

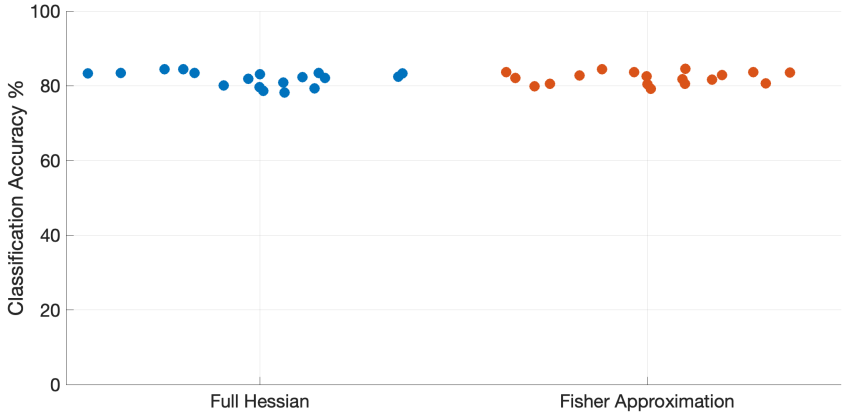

Figure 9: A ResNet-18 model is trained for CIFAR-10 dataset with Adaptive FedECADO using (a) full Hessian and (b) Fisher matrix approximation Jhunjhunwala et al. (2024) to compute the chord model.

20 Varying Configurations

To evaluate the generalizability of Adaptive FedECADO, we test its robustness under varying conditions of client participation and data heterogeneity. These experiments are designed to assess stability of our method under varying real-world settings. For each configuration, hyperparameters (listed in Appendix 15) are randomly selected for both baseline methods and Adaptive FedECADO, and we report the mean and standard deviation of classification accuracy across 20 independent runs (Tables 10–12). The results demonstrate the robustness of Adaptive FedECADO, evidenced by its consistently low variance across runs, while also achieving competitive or superior mean classification accuracy compared to baseline methods.

$[C, k]$	Adaptive FedECADO	SCAFFOLD	FedAdam	FedAdaGrad	FedExp	deltaSGD	FedCAda	FaFed
$C = 100, k = 0.3$	84.5 (1.5)	53.6 (34.6)	26.8 (25.8)	27.8 (21.3)	39.1 (32.6)	51.8 (34.7)	35.0 (29.2)	53.5 (29.9)
$C = 100, k = 0.1$	81.1 (2.3)	57.2 (39.9)	21.0 (35.3)	42.9 (28.7)	29.7 (44.6)	41.0 (28.5)	46.8 (17.4)	57.2 (19.2)
$C = 100, k = 0.05$	78.1 (2.1)	50.4 (34.1)	20.6 (39.3)	21.2 (40.2)	33.7 (31.2)	46.8 (19.5)	32.0 (24.1)	45.3 (27.9)
$C = 20, k = 0.3$	87.4 (3.5)	56.7 (23.0)	33.2 (32.2)	22.8 (31.3)	23.5 (46.4)	57.4 (15.7)	31.3 (23.4)	57.2 (14.2)
$C = 20, k = 0.1$	84.7 (5.4)	47.1 (33.2)	30.3 (40.2)	26.0 (28.8)	24.8 (32.8)	56.5 (22.1)	39.6 (26.7)	50.4 (15.1)

Table 10: Mean (Std) Classification Accuracy of CIFAR-10 (Resnet-18) trained under varying client participation settings (number of clients, C, participation ratio, k) with hyperparameters of federated learning methods randomly selected within operational range.

δ	Adaptive FedECADO	SCAFFOLD	FedAdam	FedAdaGrad	FedExp	deltaSGD	FedCAda	FaFed
0.05	80.9 (1.3)	53.6 (24.7)	22.1 (36.1)	29.2 (37.8)	26.6 (38.6)	53.5 (14.2)	53.8 (19.4)	57.4 (12.7)
0.1	84.2 (4.1)	55.2 (23.2)	25.0 (34.9)	33.1 (26.6)	26.5 (43.9)	50.2 (15.8)	60.6 (20.8)	58.8 (14.2)
0.5	86.6 (2.6)	58.0 (20.8)	37.5 (43.1)	45.1 (26.3)	33.1 (18.5)	57.6 (21.0)	62.5 (26.5)	59.3 (13.3)

Table 11: Classification Accuracy of CIFAR-10 (Resnet-18) trained under varying data distributions (Dir16(δ)) with hyperparameters of federated learning methods randomly selected within operational range.

$\bar{e}$	Adaptive FedECADO	SCAFFOLD	FedAdam	FedAdaGrad	FedExp	deltaSGD	FedCAda	FaFed
80	87.4 (1.9)	51.1 (32.1)	39.8 (26.7)	40.5 (18.1)	40.2 (20.4)	51.3 (27.7)	51.2 (29.1)	50.0 (24.2)
30	83.1 (3.5)	46.1 (36.8)	24.9 (42.8)	39.4 (43.6)	35.4 (37.8)	55.2 (13.5)	40.9 (28.6)	39.8 (26.1)

Table 12: Mean (Std) Classification Accuracy of CIFAR-10 (Resnet-18) trained under varying distributions of number of local epochs ($e_i \sim U(0, \bar{e}]$) with hyperparameters of federated learning methods randomly selected within operational range.

Furthermore, we study the effect of different sampling distributions over the hyperparameter space. Figure 10 presents the performance of each optimizer under logarithmic sampling of server and client learning rates, drawn from LogUniform$(10^{-4}, 10^{0})$. Consistent with the uniform sampling results, baseline federated learning methods remain highly sensitive to hyperparameter selection under logarithmic sampling, with a large proportion of configurations yielding unusable or divergent models. In contrast, Adaptive FedECADO consistently achieves near-optimal classification accuracy regardless of the sampled learning rate, as both client and server parameters are derived automatically from the local gradient geometry and dynamical system formulation rather than being drawn from a distribution.

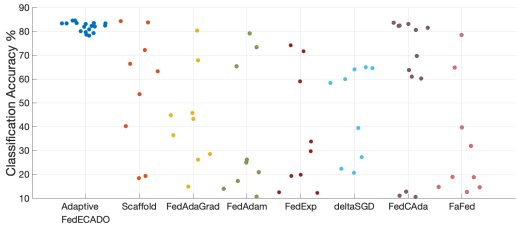

Figure 10: The classification accuracies of a ResNet-18 model trained for CIFAR-10 dataset is shown using learning rates sampled from a logarithmic distribution.

Inductance Value, L_i	Final Classification Accuracy
Adaptive client-specific L_i from (24)	85.1
FedECADO with constant inductance $L = 0.1$	81.6
FedECADO with constant inductance $L = 1$	84.7
FedECADO with constant inductance $L = 10$	80.2

Table 13: Final classification accuracies for ResNet-18 on CIFAR-10 under varying inductance choices.

21 Comparison with FedECADO

Our work extends the circuit-theoretic formulation of federated learning in FedECADO Agarwal et al. (2025) by introducing a fully adaptive federated optimization scheme tailored to heterogeneous settings. In contrast to FedECADO, which requires careful manual tuning of momentum parameters and client step sizes to achieve good performance, our approach attains competitive—and often superior—accuracy without any manual hyperparameter tuning.

Specifically, our method adaptively controls both (i) the continuous-time dynamics through client-specific inductance parameters L_i, and (ii) the discrete-time simulation through coordinated choices of local client and central server time-steps. This joint adaptation ensures numerical stability of the discretized dynamics and reliable convergence to the steady state across heterogeneous clients.

While FedECADO relies on carefully selected inductance values to balance convergence speed and stability, our framework selects near-optimal inductance values, L_i, for fast convergence speed in continuous time without compromising stability of the underlying dynamics. To isolate the effect of the inductance parameters, we perform an ablation study in which all momentum parameters of FedECADO are varied by multiple orders of magnitude and is compared to our adaptive, client-specific inductance selection (computed via (24)).

As shown in Table 13-14, the performance of FedECADO is sensitive to the choice of inductance, with suboptimal values leading to noticeable degradation in final accuracy. In contrast, our adaptive inductance selection consistently achieves both faster convergence and the highest final accuracy, without requiring any momentum tuning.

Furthermore, the stability of our method is enforced by the proposed Backward-Euler numerical discretization scheme and LTE-controlled client step size selection. FedECADO is susceptible to divergence in local clients

Inductance Value, L_i	Final Classification Accuracy
Adaptive client-specific L_i from (24)	50.4
FedECADO with constant inductance $L = 0.1$	44.3
FedECADO with constant inductance $L = 1$	47.7
FedECADO with constant inductance $L = 10$	50.6

Table 14: Final classification accuracies for ResNet-50 on CIFAR-100 under varying inductance choices.

Local Client Step Size	Final Classification Accuracy
Adaptive client-specific step-size (LTE-controlled)	50.4
FedECADO with constant client step $\Delta t_i = 0.01$	43.56
FedECADO with constant client step $\Delta t_i = 0.05$	45.23
FedECADO with constant client step $\Delta t_i = 0.1$	46.68
FedECADO with constant client step $\Delta t_i = 0.2$	51.00
FedECADO with constant client step $\Delta t_i = 0.3$	Divergence

Table 15: Final classification accuracies for ResNet-50 on CIFAR-100 under varying local client step-size choices.

due to poorly selected step-sizes. Our methodology ensures that local clients adaptively select their step-sizes based on the gradient-space and is bounded by the global LTE value. As shown in Table 15

Furthermore, the stability of our method is enforced by the proposed Backward-Euler numerical discretization combined with LTE-controlled adaptive client step-size selection. In contrast, FedECADO is susceptible to divergence at local clients when step-sizes are poorly chosen, particularly in heterogeneous settings.

Our methodology ensures that each local client adaptively selects its step-size based on local gradient space and the resulting numerical error is globally bounded by an LTE threshold. To quantify the impact of client step-size selection, we compare our adaptive scheme against FedECADO with fixed client step-sizes spanning multiple orders of magnitude. The results for training ResNet-50 on CIFAR-100 are reported in Table 15.

Overall, our method uses insights from the circuit formalism derived in FedECADO but demonstrates a principled numerical discretization and circuit designed momentum values to yield stable and reliable convergence without the tuning requirements inherent to FedECADO.

22 Wallclock Runtime

The relative wall-clock times for all methods are presented in Table 16. While our method incurs slightly higher computational cost due to the dynamical system formulation and Backward-Euler steps, we believe that this overhead is compensated by the elimination of the hyperparameter tuning phase, which can be time-consuming and costly in federated environments.

Method	Normalized Wallclock Time
Adaptive FedECADO	1
FedAdam	0.94
FedAdaGrad	0.93
SCAFFOLD	0.96
FedExp	0.94
deltaSGD	0.97
FedCAda	0.98
FAFED	0.95

Table 16: Wallclock runtime (normalized to Adaptive FedECADO) for training a ResNet-18 model on the CIFAR-10 dataset over 100 epochs using different baseline methods.

Per-Round Runtime Breakdown: On the client side, the dominant cost is standard backpropagation, accounting for approximately 90–95% of client-side time and identical to all baselines. The remaining 5–10% is attributable to the local ODE time-stepping update. On the server side, the Backward-Euler integration step dominates at approximately 70–80% of server-side time; however, this step reduces to a pair of triangular solves against a pre-factorized LU decomposition computed once at the start of training, making it cheap in absolute terms. Server-side time-stepping accounts for approximately 15–20% of server-side time, with the LTE check comprising the remaining 5–10%.

Communication Overhead: Adaptive FedECADO introduces negligible additional communication cost relative to standard federated baselines. The only quantity communicated beyond gradients and model parameters is a single scalar, Δt_i, per client per round, representing the client's simulated time-period. This constitutes a near-minimal communication overhead regardless of model size, as the additional payload is independent of the model dimensionality d and remains constant across rounds.

Memory Requirements: Regarding memory requirements, Adaptive FedECADO employs a diagonal approximation of the client sensitivity matrix $\hat{G}_{th}^i$, which reduces storage from $\mathcal{O}(d^2)$ for a full Hessian to $\mathcal{O}(d)$ per client. For the architectures considered in this work, the diagonal Hessian requires approximately 22.4MB (ResNet-18, $d = 11,689,512$), 48.8MB (ResNet-50, $d = 25,557,032$), and 253.9MB (VGG-11, $d = 132,863,336$) per client in 16-bit floating point precision. Since the LU factorization of a diagonal matrix is trivial — with $L = I$ and $U = \text{diag}(\hat{G}_{th}^i)$ — no additional storage beyond the diagonal itself is required, and the pre-computed factors are reused across all rounds at negligible per-round cost. The diagonal approximation is well-motivated in our setting, as $\hat{G}_{th}^i$ serves primarily to provide relative weighting between parameter dimensions rather than capturing full second-order interactions, and is consistent with widely used diagonal Hessian approximations in adaptive optimization (Goodfellow et al., 2016).

23 Using Default Hyperparameters

To further motivate the need for adaptive hyperparameter selection, we present an experiment training a ResNet-18 model on the CIFAR-10 dataset under heterogeneous data distributions. In this experiment, each client uses Adam with default hyperparameters and FedAvg aggregation with a server-side learning rate of 1, reflecting the setting a practitioner might naively adopt without problem-specific tuning. As shown in Figure 11, while the default configuration yields a functional model, it is far from optimal. Varying the client learning rates reveals that significantly better convergence properties are achievable. However, many alternative configurations produce unusable or divergent models, demonstrating that the default setting is neither reliably safe nor reliably optimal. This illustrates the core challenge: without knowledge of the data heterogeneity, model architecture, and client dynamics, there is no principled basis for selecting a default learning rate that generalizes across federated deployments, and practitioners risk poor convergence or divergence with no clear diagnostic signal.

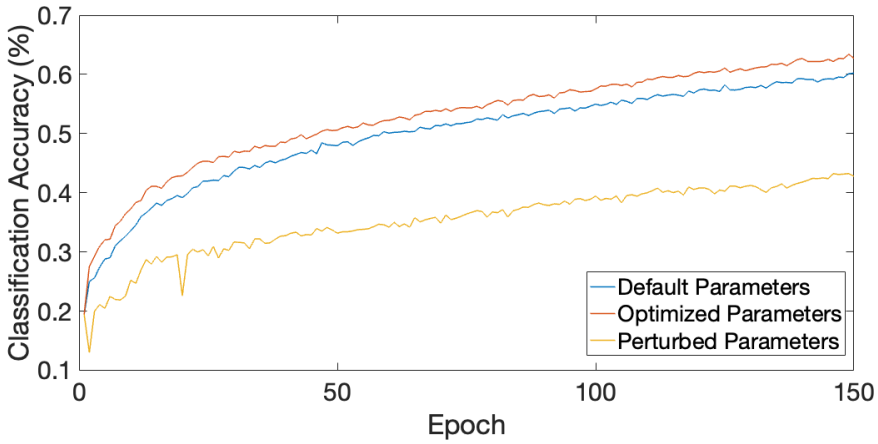

Figure 11: Classification accuracy over 150 epochs for ResNet-18 trained on CIFAR-10 under heterogeneous data distributions using FedAvg with a server-side learning rate of 1. Three client-side SGD configurations are shown: default learning rates, optimized client-specific learning rates, and perturbed configurations around the optimized values.