SCALIP: BRIDGING THE SCALABILITY–TIGHTNESS GAP IN LIPSCHITZ ESTIMATION OF DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tight Lipschitz constant estimation is fundamental to the certified robustness of deep neural networks. While state-of-the-art Semidefinite Programming (SDP) formulations yield tight formal bounds, their monolithic Linear Matrix Inequality (LMI) constraints couple decision variables globally, scaling poorly with network depth and width. Recent compositional approaches mitigate this by breaking the massive LMI into smaller layer-wise SDPs; however, to achieve this decoupling, they should rely on intermediate surrogate objectives that inherently degrade the final bound. Moreover, despite this decomposition, the resulting layer-wise SDPs remain computationally intractable for *convolutional* layers. In this paper, we propose ScaLip, a novel reformulation of the exact SDP problem into an unconstrained optimization objective. By introducing a recursive framework, the strict feasibility of the underlying LMI constraints is guaranteed by construction. This reformulation allows the global Lipschitz bound to be computed efficiently, end-to-end, using highly scalable first-order methods. We mathematically prove that our unconstrained objective preserves the optimal value of the original SDP formulation. Empirical results demonstrate that ScaLip matches the tightness of monolithic SDPs while requiring only a fraction of their computational cost. Remarkably, it scales substantially better than even existing layer-wise SDP approaches, which have so far offered the most scalable SDP-based alternative. This enhanced scalability extends tight SDP-based Lipschitz estimation to convolutional neural networks and paves the way for analyzing much larger architectures.

1 INTRODUCTION

Deep neural networks have achieved remarkable success across numerous domains, yet their severe vulnerability to adversarial perturbations (Szegedy et al., 2014; Goodfellow et al., 2014; Kurakin et al., 2017; Ilyas et al., 2019; Zhang et al., 2024; Rando et al., 2026) remains a critical bottleneck for safety-critical applications. This fragility necessitates rigorous mathematical frameworks to formally bound the worst-case behavior of these models against input perturbations. The global Lipschitz constant provides exactly this foundation, serving as a definitive theoretical metric that bounds the maximum amplification of any input perturbation and thereby quantifies the network's perturbation invariance (Hein & Andriushchenko, 2017; Tsuzuku et al., 2018).

While computing the exact Lipschitz constant is NP-hard (Virmaux & Scaman, 2018; Wang et al., 2022), Semidefinite Programming (SDP) offers a mathematically rigorous alternative. In particular, by bounding the behavior of nonlinear activations via quadratic constraints, SDP provides a foundation for extracting tight Lipschitz estimations (Fazlyab et al., 2019; Pauli et al., 2024). However, standard end-to-end SDP formulations couple decision variables across all layers into a single, massive Linear Matrix Inequality (LMI). This global coupling severely bottlenecks scalability.

Previous attempts to resolve this bottleneck have introduced their own severe limitations. Specifically, while promising, recent efforts to apply first-order methods, such as LipDiff (Wang et al., 2024), face challenges in handling the LMI constraint efficiently: their direct optimization relies on a penalty method to enforce the constraint over the massive, globally coupled network matrix, resulting in severe computational overhead and poor empirical convergence to the optimal bound. Alternatively,

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

notable compositional approaches such as EclipsE (Xu & Sivaranjani, 2024) break the large LMI into layer-wise constraints to improve scalability; however, achieving this decomposition requires intermediate surrogate objectives that degrade the tightness of the final bound.

In this paper, we propose ScaLip, a scalable, mathematically equivalent reformulation of the Lipschitz bound estimation framework of Fazlyab et al. (2019) that avoids both the computational bottleneck of monolithic LMIs and the bound degradation of intermediate surrogate objectives.

1.1 Contributions

Specifically, we develop a novel mathematical reformulation of the SDP relaxation for the Lipschitz bound estimation problem, transforming the globally coupled LMI of standard SDP approaches into an equivalent, unconstrained optimization objective. To maximize the practical utility of this theoretical equivalence, we introduce specialized numerical techniques tailored to our unconstrained framework, enabling the recursive bound calculation to be highly accelerated on modern GPUs using scalable first-order gradient methods. Our comprehensive empirical evaluations demonstrate that ScaLip achieves formal robustness bounds that recover the rigorous guarantees of the monolithic LipSDP formulation. Crucially, it does so at a fraction of the computational cost, successfully extending tight Lipschitz constant calculation to much larger neural network architectures, including convolutional neural networks, a domain where traditional SDP solvers routinely run out of memory for all but the smallest architectures.

1.2 Related Work

The exact computation of global Lipschitz bounds for deep neural networks is fundamentally NP-hard (Virmaux & Scaman, 2018; Wang et al., 2022). While exact formulations exist, including Mixed-Integer Programming (Jordan & Dimakis, 2020), polynomial optimization methods (Latorre et al., 2020), and completely positive and copositive programming (Noori et al., 2025), their computational demands restrict their applicability to trivially small networks. Conversely, naive layer-wise products of spectral norms scale effortlessly but yield notoriously loose bounds. To navigate this trade-off, various relaxations have been proposed, including sequential spectral bounds on the network Jacobian (Virmaux & Scaman, 2018) and operator-theoretic analyses of activation functions (Combettes & Pesquet, 2020).

In recent years, SDP has significantly advanced the state of the art. By relaxing nonlinear activation functions into global quadratic constraints, methods like LipSDP (Fazlyab et al., 2019) provide highly rigorous certificates. However, scaling these monolithic SDPs remains a formidable challenge. The compositional framework EclipsE (Xu & Sivaranjani, 2024) improves efficiency by decoupling layers, but suffers from bound degradation due to intermediate surrogate objectives. Meanwhile, the first-order adaptation LipDiff (Wang et al., 2024) attempts to directly cast LipSDP as a first-order optimization problem via penalty methods, but remains computationally expensive and yields suboptimal convergence.

In parallel, first-order methods have seen immense success in the closely related domain of *local* robustness verification. While the exact formal verification problem is NP-complete (Katz et al., 2017), first-order bound propagation methods such as α-CROWN, β-CROWN, and SDP-CROWN (Xu et al., 2021; Wang et al., 2021; Chiu et al., 2025) have emerged as the state of the art. These highly optimized first-order frameworks now consistently rival, and often outperform, traditional SDP-based verifiers (Batten et al., 2021; Fazlyab et al., 2022; Brown et al., 2022) in both scalability and tightness for local robustness problems. Yet, despite their dominance in local verification, the computational efficiency of scalable first-order optimization has not been successfully translated to the calculation of *global* Lipschitz constants. Our ScaLip framework bridges this gap by structurally eliminating the coupled LMI constraints, enabling first-order methods to achieve the exact tightness of SDP-based global Lipschitz bounds while scaling to substantially larger neural network architectures.

2 Problem Statement and Background

Notation. Let $\mathbb{R}^n$ and $\mathbb{R}^n_+$ denote the n-dimensional Euclidean space and its non-negative orthant, respectively, while $\mathbb{R}_{++}$ denotes the set of strictly positive real scalars. For vectors $u, v \in \mathbb{R}^n$,

inequalities such as $u \leq v$ are evaluated element-wise. For a symmetric matrix P, $P \succeq 0$ and $P \succ 0$ signify that P is positive semidefinite and positive definite, respectively, while $\lambda_{\max}(P)$ denotes its maximum eigenvalue. The sets $\mathbb{D}_+^n$ and $\mathbb{D}_{++}^n$ denote the spaces of $n \times n$ positive semidefinite and strictly positive definite *diagonal* matrices, respectively. For a vector $v \in \mathbb{R}^n$, $\mathrm{diag}(v)$ denotes the diagonal matrix with v on its main diagonal, while $\exp(v)$ represents the element-wise exponential. The notation $\mathbf{I}_n$ denotes the $n \times n$ identity matrix, $\mathbf{0}_{n\times m}$ the $n \times m$ zero matrix, and $\mathbf{0}$ a zero matrix or vector of appropriate dimensions. The symbol $\odot$ denotes the element-wise (Hadamard) product, while $\|\cdot\|_2$ represents the standard Euclidean norm. Finally, in symmetric block matrices, $\star$ represents the corresponding transposed off-diagonal block.

2.1 Problem Statement

Consider a feedforward neural network with D layers defined as $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_D}$. The recursive transformation in the network is given by

$$x_i = \phi_i(W_i x_{i-1} + b_i), \quad i = 1, \ldots, D-1, \quad f(x_0) = W_D x_{D-1} + b_D, \tag{1}$$

where $x_0 \in \mathbb{R}^{n_0}$ is the network input. For each layer i, the matrix $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and the vector $b_i \in \mathbb{R}^{n_i}$ are the weights and biases, $x_i \in \mathbb{R}^{n_i}$ is the output, and $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ is a nonlinear activation function applied element-wise. We assume these scalar activations, denoted by $\phi_{i,j}$, are slope-restricted in $[0, 1]$, satisfying $0 \leq \frac{\phi_{i,j}(x)-\phi_{i,j}(y)}{x-y} \leq 1$ for all $x \neq y$. This standard assumption covers many standard activation functions including ReLU, Sigmoid, and tanh.

An ℓ_2-norm Lipschitz bound for this neural network is defined as a non-negative scalar $L \geq 0$ such that $\|f(x_0) - f(y_0)\|_2 \leq L\|x_0 - y_0\|_2$, for all $x_0, y_0 \in \mathbb{R}^{n_0}$.

2.2 Lipschitz Constant Estimation via Semidefinite Programming

By rewriting the slope restrictions as quadratic constraints, the problem of finding the tightest Lipschitz bound for the neural network in (1) can be formulated as the following quadratically constrained quadratic program (QCQP):

$$\begin{aligned} \max_{\Delta x_i} \quad & \|W_D \Delta x_{D-1}\|_2^2 \\ \text{s.t.} \quad & \|\Delta x_0\|_2^2 \leq 1, \\ & \Delta x_i \odot (\Delta x_i - W_i \Delta x_{i-1}) \leq \mathbf{0}, \quad i = 1, \ldots, D-1, \end{aligned} \tag{2}$$

where Δ denotes the difference between the layer outputs corresponding to two distinct inputs x_0 and y_0 ($\Delta x_i = x_i - y_i \in \mathbb{R}^{n_i}$). The tightest Lipschitz bound obtained from this formulation is the square root of its optimal value.

Solving this exact QCQP remains NP-hard (Wang et al., 2022). To resolve this, Fazlyab et al. (2019) relax the intractable QCQP into the following tractable SDP:

$$\begin{aligned} \min_{\rho, \Theta_i} \quad & \rho \\ \text{s.t.} \quad & \begin{bmatrix} \rho\mathbf{I} & -\frac{1}{2}W_1^\top \Theta_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \star & \Theta_1 & -\frac{1}{2}W_2^\top \Theta_2 & \cdots & \mathbf{0} \\ \mathbf{0} & \star & \Theta_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & -\frac{1}{2}W_{D-1}^\top \Theta_{D-1} \\ \mathbf{0} & \mathbf{0} & \cdots & \star & \Theta_{D-1} - W_D^\top W_D \end{bmatrix} \succeq 0, \end{aligned} \tag{3}$$

where $\Theta_i \in \mathbb{D}_+^{n_i}$ and $\rho \in \mathbb{R}_+$.

The primary limitation of directly solving (3) is that the dimension of the LMI is $\sum_{i=0}^{D-1} n_i$. This global coupling becomes computationally prohibitive as the number of neurons in the network increases. To mitigate this computational bottleneck, Fazlyab et al. (2019) also introduce LipSDP-Layer, a relaxed variant that restricts the decision variables to uniform scalar multiples of the identity matrix ($\Theta_i = \zeta_i \mathbf{I}_{n_i}$). While this restriction slightly reduces the computational complexity, it severely suppresses the expressivity of the decision variables, leading to noticeably suboptimal Lipschitz bounds.

2.3 Compositional Decomposition

To alleviate the scalability limits of the end-to-end SDP formulation, Theorem 3 in Xu & Sivaranjani (2024) shows that the monolithic LMI constraint can be equivalently decomposed into smaller layer-wise constraints. Assuming strict feasibility of the LMI in (3) (i.e., $\succ 0$), the theorem decomposes the problem as

$$\begin{aligned} \min_{\Theta_i} \quad & \lambda_{\max}\left(W_D M_{D-1}^{-1} W_D^\top\right) \\ \text{s.t.} \quad & M_i \succ 0, \quad \forall i = 0, \dots, D-1, \end{aligned} \tag{4}$$

where

$$M_i = \begin{cases} \mathbf{I}_{n_0}, & i = 0, \\ \Theta_i - \frac{1}{4}\Theta_i W_i M_{i-1}^{-1} W_i^\top \Theta_i, & i = 1, \dots, D-1. \end{cases}$$

For completeness, a proof of this equivalence using the Schur complement is provided in Appendix A.1. Xu & Sivaranjani (2024) utilize this decomposition to compute the Lipschitz constant in a compositional manner, sequentially finding decision variables Θ_i that satisfy $M_i \succ 0$, while optimizing intermediate objective functions. Although breaking the massive monolithic LMI into smaller layer-wise LMIs significantly reduces computational complexity, the EclipsE framework presents two notable drawbacks. First, the surrogate intermediate objective functions do not directly align with the true objective of minimizing the *global* Lipschitz bound, leading to looser bounds than those obtained using the end-to-end LipSDP formulation proposed by Fazlyab et al. (2019). Second, EclipsE still necessitates solving semidefinite programs of size n_i at each step, which can remain computationally expensive for wide layers, particularly in convolutional networks (see Section 3.3).

To address this computational bottleneck, the authors also propose EclipsE-Fast, a highly accelerated variant. Analogous to LipSDP-Layer, EclipsE-Fast restricts the decision variables to scalar multiples of the identity matrix ($\Theta_i = \zeta_i \mathbf{I}_{n_i}$) and derives a closed-form optimal value for a specific intermediate surrogate objective as $\zeta_i = \frac{2}{\lambda_{\max}\left(W_i M_{i-1}^{-1} W_i^\top\right)}$. While this closed-form step can be computed rapidly and improves upon the naive multiplication of layer-wise spectral norms, the severe structural limitation imposed on the decision variables yields inherently loose global bounds.

Remark 1. *The problem in* (4) *is mathematically equivalent to the strictly feasible variant of* (3)*, obtained by replacing $\succeq 0$ with $\succ 0$. In practice, this distinction is negligible because standard interior-point solvers traverse the strict interior of the positive semidefinite cone, making the assumption consistent with numerical implementations without introducing additional conservatism.*

In the next section, we leverage the algebraic decomposition in (4) to develop a highly scalable, end-to-end first-order optimization framework that exactly recovers the global Lipschitz bound produced by the monolithic LipSDP formulation.

3 Proposed Method: Scalable yet Accurate Lipschitz Calculation

The monolithic SDP formulation in (3) yields a tight bound for the Lipschitz constant of the neural network but imposes a severe computational burden. While the recursive equations in (4) successfully decompose the large LMI into smaller layer-wise LMIs, the presence of the nonlinear term $\Theta_i W_i M_{i-1}^{-1} W_i^\top \Theta_i$ prevents the simultaneous optimization of all decision variables Θ_i via standard SDP solvers in an end-to-end manner. Furthermore, it is intractable to solve (3) or (4) directly using scalable first-order methods, as the positive (semi)definite constraints ($M \succeq 0$ or $M_i \succ 0$) cannot be readily enforced during unconstrained gradient descent.

3.1 Equivalent Unconstrained Formulation

Theorem 1 resolves these bottlenecks by reformulating the constrained LMI conditions into an equivalent, fully unconstrained optimization problem.

Theorem 1. *Let $\theta_i \in \mathbb{D}_{++}^{n_i}$ and $c_i \in (0,1)$ for $i = 1, \dots, D-1$. Consider the optimization problem*

$$\min_{c_i, \theta_i} \quad \lambda_{\max}\left(\tilde{W}_D H_{D-1}^{-1} \tilde{W}_D^\top\right), \tag{5}$$

where the transformed weights are defined as

$$\tilde{W}_i = \begin{cases} \theta_1 W_1, & i = 1, \\ \theta_i W_i \theta_{i-1}^{-1}, & i = 2, \dots, D-1, \\ W_D \theta_{D-1}^{-1}, & i = D, \end{cases}$$

and the sequence is initialized by $H_0 = \mathbf{I}_{n_0}$ with the following recursive updates for $i = 1, \dots, D-1$:

$$r_i = \frac{c_i}{\lambda_{\max}\left(\frac{1}{4}\tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top\right)}, \quad H_i = r_i \mathbf{I}_{n_i} - \frac{1}{4} r_i^2 \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top. \tag{6}$$

Then the optimal value of this problem is exactly equal to the optimal value of (3)*.*

Proof sketch. The complete proof is provided in Appendix A.2. Briefly, this equivalence is established through a change of variables. Specifically, by decomposing the original decision variables Θ_i into strictly positive diagonal components and parameterizing the margins using the scalar $c_i \in (0, 1)$, the recursive formulation guarantees the strict feasibility of the underlying LMI constraints by construction, thereby eliminating the need for constrained optimization. □

Building upon Theorem 1, we can now construct an end-to-end differentiable pipeline to optimize the global Lipschitz bound. To strictly enforce the domains $\theta_i \in \mathbb{D}_{++}^{n_i}$ and $c_i \in (0, 1)$ during unconstrained first-order optimization, we introduce the unconstrained variables $\gamma_i \in \mathbb{R}^{n_i}$ and $u_i \in \mathbb{R}$ and parameterize the variables via $\theta_i = \mathrm{diag}(\exp(\gamma_i))$ and $c_i = \sigma(u_i)$, where $\sigma(\cdot)$ is the standard sigmoid function.

We note that while the presence of the maximum eigenvalue operator $\lambda_{\max}(\cdot)$ in the recursive equations renders the objective non-smooth, the function remains piecewise smooth and differentiable almost everywhere (Overton, 1988). Consequently, the unconstrained parameters γ_i and u_i can be optimized directly using standard automatic-differentiation frameworks (e.g., PyTorch), which compute valid subgradients for the maximum eigenvalue under the hood. Furthermore, we significantly enhance computational efficiency by employing a warm-started Lanczos algorithm to approximate the maximum eigenvalue. By computing the dominant eigenvector outside the computational graph, we can efficiently evaluate the required subgradients while completely avoiding the overhead of back-propagating through the iterative solver. The complete forward-backward procedure is summarized in Algorithm 1.

3.2 Weight-Rescaling Interpretation

As discussed in Section 2, methods like LipSDP-Layer and EclipsE-Fast provide faster Lipschitz bounds by heavily restricting the decision variables Θ_i to scaled identity matrices. However, this restriction fundamentally suppresses the expressivity of the optimization space, resulting in loose bounds. In ScaLip, imposing the same restriction would correspond to fixing $\theta_i = \mathbf{I}_{n_i}$, thereby reducing $\Theta_i = r_i \theta_i^2$ to a scalar multiple of the identity. ScaLip instead optimizes each diagonal entry of θ_i, restoring the lost expressivity and enabling tighter bounds.

Consider a neural network parameterized by the transformed weights $\tilde{W}_i = \theta_i W_i \theta_{i-1}^{-1}$ and biases $\tilde{b}_i = \theta_i b_i$. By defining the scaled variables $\tilde{x}_i = \theta_i^{-1} x_i$ (with $\theta_0 = \mathbf{I}_{n_0}$), the network mappings can be equivalently rewritten using the original structural weights W_i and b_i as

$$\tilde{x}_i = \tilde{\phi}_i(W_i \tilde{x}_{i-1} + b_i), \quad i = 1, \dots, D-1, \quad \tilde{f}(\tilde{x}_0) = W_D \tilde{x}_{D-1} + b_D,$$

governed entirely by the following modified activation functions (where $z \in \mathbb{R}^{n_i}$):

$$\tilde{\phi}_i(z) = \theta_i^{-1} \phi_i(\theta_i z).$$

Crucially, because $\theta_i \in \mathbb{D}_{++}^{n_i}$ is a strictly positive diagonal matrix, this transformation perfectly preserves the $[0, 1]$ slope-restriction of the original activation function ϕ_i. The resulting QCQP in (2) therefore remains exactly the same.

As illustrated in Figure 1, this modified activation function perfectly captures the rescaling mechanism. In the case of positively homogeneous activation functions (e.g., $\phi(\alpha x) = \alpha\phi(x)$ for $\alpha > 0$) such as ReLU, this matched scaling cascades perfectly through the activation. Consequently, while the output values of the intermediate layers are scaled, the overall input-output relation of the network stays exactly the same, leaving the underlying information flow functionally invariant while fundamentally altering the numerical conditioning for optimization.

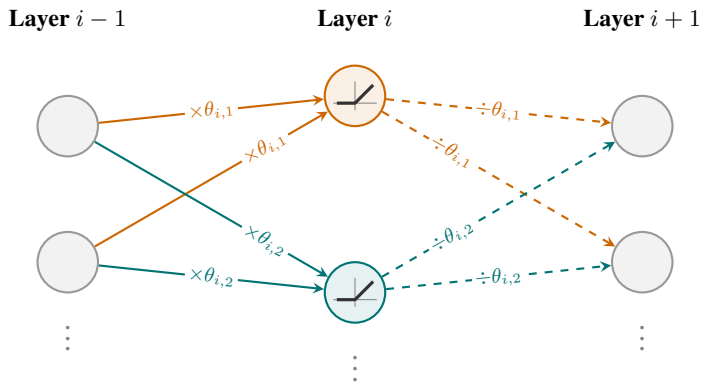


Figure 1: Neuron-level view of weight rescaling. For any individual neuron j in hidden layer i, the incoming weights and biases, and outgoing weights are scaled by $\theta_{i,j}$ and its inverse, respectively. For positively homogeneous activation functions such as ReLU, this scaling passes perfectly from the input of the neuron to its output. The variables θ_i in ScaLip intuitively act as this dynamic rescaling.

Ultimately, ScaLip lifts the efficient closed-form logic underlying EclipsE-Fast into an end-to-end optimization framework that restores full diagonal expressivity through dynamic weight rescaling and directly optimizes the final global Lipschitz bound. Crucially, instead of locking the scalar decision variables ζ_i to satisfy specific surrogate intermediate objectives, ScaLip dynamically tunes both the scalar margins (c_i) and the rescaling variables (θ_i) end-to-end, strictly optimizing the final global objective.

3.3 Scaling to Convolutional Networks

While convolutional layers can be unrolled within fully connected frameworks, the resulting dimensional explosion renders monolithic SDP solvers intractable even for trivially small networks. The same limitation applies to the layer-wise SDPs of EclipsE: although they reduce the computational complexity of LipSDP, they remain unable to handle the SDP corresponding to a convolutional layer. Furthermore, although direct gradient methods such as LipDiff can be applied to unrolled convolutions, they struggle to achieve competitive bounds or acceptable runtimes, as demonstrated in Section 4. ScaLip's unconstrained recursive formulation fundamentally circumvents these bottlenecks by accommodating the high-dimensional structure of unrolled convolutional layers without requiring explicit LMI formulations.

3.4 Numerical Acceleration Techniques

To maximize the practical benefits of this theoretical equivalence, we introduce numerical procedures tailored to the unconstrained formulation that efficiently execute the recursive bound computations on modern GPUs using scalable first-order optimization.

Initialization. We initialize the unconstrained parameters as $\gamma_i = \mathbf{0}_{n_i}$ (implying $\theta_i = \mathbf{I}_{n_i}$) and $u_i = 0$ (implying $c_i = 0.5$). Remarkably, the Lipschitz bound produced at this specific initialization exactly recovers the analytical bound provided by EclipsE-Fast (Xu & Sivaranjani, 2024) (see Appendix A.3 for proof). Since EclipsE-Fast already yields a strong baseline, this theoretically grounded initialization provides an excellent warm-start. Fine-tuning from this anchor accelerates convergence and consistently yields tighter bounds than initializing from a random seed.

Accelerated Subroutine and Cholesky Guardrails. Evaluating the recursive sequence in (6) requires computing $K_i = \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top$ and extracting its maximum eigenvalue, $\lambda_{\max}(K_i)$, at each layer. For small networks, computing the explicit matrix inverse and exact eigendecomposition is entirely feasible. However, to scale to deeper and wider architectures, we introduce an accelerated numerical routine detailed in Algorithm 1.

To bypass explicit matrix inversion, we maintain the Cholesky decomposition $L_{i-1} L_{i-1}^\top = H_{i-1}$ throughout the forward pass, allowing us to compute K_i efficiently via a Cholesky solver. However, the primary computational bottleneck remains the recursive extraction of the maximum eigenvalue for each layer. To accelerate this, we approximate $\lambda_{\max}(K_i)$ using a warm-started Lanczos algorithm, initializing the solver with the dominant eigenvector cached from the previous epoch. The exact details of this process are provided in Algorithm 2 in Appendix B.2. While it is also possible to use other iterative methods such as power iteration, the Lanczos algorithm is known to converge in fewer iterations.

This approximation is highly accurate in practice. However, to guarantee that our formulation remains mathematically sound, our continuous Cholesky factorization acts as a built-in guardrail. In particular, in the rare event of a severe under-approximation, H_i could theoretically lose strict positive definiteness, rendering the method mathematically unsound. If this occurs, the Cholesky solver gracefully catches the factorization failure and triggers an immediate fallback to exact eigendecomposition (Algorithm 1). Finally, the objective function $\lambda_{\max}(K_D)$ is always computed exactly to preserve strict accuracy. A complete mathematical analysis of this guardrail mechanism is provided in Appendix B.1. Remarkably, due to the high precision of the warm-started Lanczos method, this exact fallback was never triggered during our empirical evaluations.

4 Experiments

In this section, we conduct an extensive empirical evaluation of ScaLip in terms of scalability, computational efficiency, and bound tightness. We organize the evaluation into three sets of experiments. First, we isolate and quantify the computational speedup achieved by our numerical acceleration techniques. Second, we benchmark ScaLip against state-of-the-art baselines on randomly initialized networks with varying depths and widths. Finally, we demonstrate its practical applicability on multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) trained on the MNIST and CIFAR-10 datasets.

Experimental Setup. Throughout our evaluation, we compare ScaLip against the monolithic LipSDP-Neuron formulation (Fazlyab et al., 2019), the compositional SDP approaches EclipsE and EclipsE-Fast (Xu & Sivaranjani, 2024), and the first-order optimization method LipDiff (Wang et al., 2024). The SDP-based baselines are implemented using CVXPY (Diamond & Boyd, 2016) and solved via MOSEK (ApS, 2025). All first-order methods, including ScaLip, are implemented in PyTorch. We evaluate ScaLip using both CPU (ScaLip-CPU) and GPU (ScaLip-GPU) implementations. The "Naive" baseline in our experiments refers to the product of the layer-wise spectral norms of the weight matrices. For all experiments, ScaLip is optimized for 250 iterations using the Adam optimizer with an initial learning rate of 0.1 and a `ReduceLROnPlateau` scheduler, while the warm-started Lanczos subroutine utilizes a fixed number of 10 iterations. All experiments are executed on a machine equipped with an 8-core Intel Xeon processor, 64 GB of RAM, and an NVIDIA L4 GPU with 24 GB of VRAM.

4.1 Experiment 1: Impact of Computational Acceleration

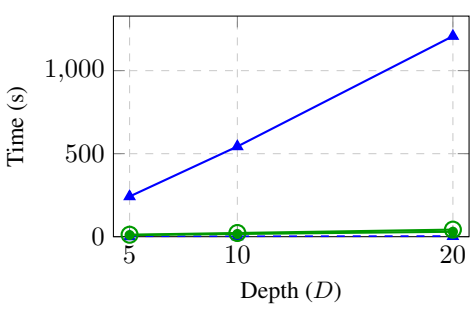
In our first experiment, we evaluate the accelerated subroutine for maximum eigenvalue estimation and matrix inversion detailed in Section 3.4. Table 1 reports the computation times across three randomly generated networks. Crucially, both the exact and accelerated implementations achieve near-identical Lipschitz bounds (L), confirming that our Cholesky guardrails and warm-started Lanczos approximations preserve theoretical rigor without compromising tightness.

The acceleration gains are substantially more pronounced for larger networks, particularly on the GPU. For example, GPU acceleration reduces the computation time by approximately two-thirds for the medium network and by more than one-half for the large network. This demonstrates that our warm-started Lanczos procedure effectively mitigates the fundamental computational bottleneck

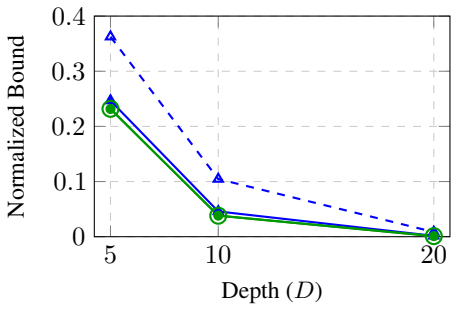
associated with recursive eigendecompositions, thereby enabling the framework to scale efficiently while fully exploiting hardware parallelism.

Table 1: **Experiment 1**: Effect of numerical acceleration on computation time (seconds) across three random networks with weights drawn from $\mathcal{N}(0, 1/\sqrt{n_i})$. The accelerated approach provides substantial speedups, particularly for larger networks. All configurations converge to the same Lipschitz bound L within a negligible deviation ($\pm 0.3\%$).

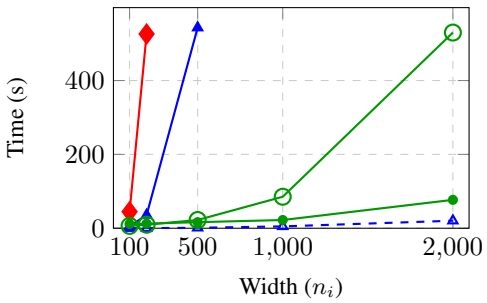
Network Size	**CPU Time (s)**		**GPU Time (s)**	
	Exact	**Accelerated**	**Exact**	**Accelerated**
Small ($D = 4, n_i = 200, L = 5.26$)	4.92	7.74	6.48	5.22
Medium ($D = 8, n_i = 500, L = 18.82$)	35.68	17.82	37.39	13.65
Large ($D = 12, n_i = 2000, L = 79.45$)	1116.81	640.55	198.73	95.56



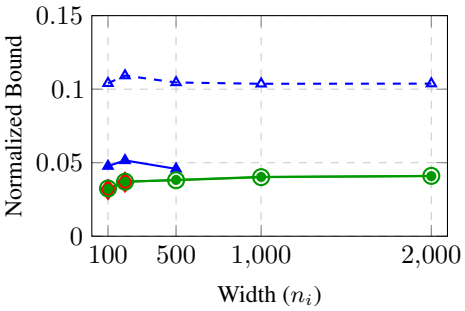
(a) Runtime vs. Depth (Fixed $n_i = 500$)



(b) Normalized Bound vs. Depth (Fixed $n_i = 500$)



(c) Runtime vs. Width (Fixed $D = 10$)



(d) Normalized Bound vs. Width (Fixed $D = 10$)



Figure 2: **Experiment 2:** Comparison of runtime (left) and normalized Lipschitz bounds (right). Bound values are presented as a fraction of the loose Naive baseline. Missing data points indicate out-of-memory (OOM) or timeout (TO $>$ 30 min) failures.

4.2 Experiment 2: Scalability and Bound Tightness on Random Networks

Next, we compare the scalability of ScaLip with that of the baseline methods as network depth and width increase. We consider neural networks with uniform layer sizes $n_i \in \{100, 200, 500, 1000, 2000\}$ and depths $D \in \{5, 10, 20\}$, drawing weights randomly from $\mathcal{N}(0, 1/\sqrt{n_i})$. Complete tabulated results are provided in Appendix B.3, while Figure 2 highlights the key trends.

As illustrated in Figures 2a and 2c, ScaLip consistently computes the bounds orders of magnitude faster than LipSDP, which frequently encounters out-of-memory (OOM) failures on wider and deeper

networks (indicated by missing data points). While the fast method EclipsE-Fast provides tighter bounds than the naive approach, Figures 2b and 2d show that its bounds remain loose in practice compared to those of other methods. By contrast, ScaLip exactly matches the bounds of LipSDP across all tested depths and widths while entirely avoiding its memory bottlenecks. Furthermore, the bounds produced by LipDiff offer only marginal improvements over the naive baseline. Given this limited performance, we omit LipDiff from Figure 2 and the subsequent experiment; however, its complete results are provided in Appendix B.3. We use the hyperparameter settings from the second example in the LipDiff implementation, and our additional tuning efforts did not yield improved results.

4.3 Experiment 3: Evaluation on Trained Models

Finally, we evaluate the practical scalability of our framework on models trained on standard datasets. Table 2 presents the performance on multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs) across the MNIST and CIFAR-10 classification tasks. Full architectural details for the models are provided in Appendix B.4.

The scalability limitations imposed by LMI constraints become immediately apparent in these practical settings. Specifically, LipSDP runs out of memory on all but the smallest architecture, while even the compositionally decoupled EclipsE runs out of memory on the widest MLP ($n_i = 2048$). Indeed, unrolling convolutional layers for CNNs exacerbates this dimensionality explosion, rendering SDP solvers entirely intractable for standard vision models.

In stark contrast, ScaLip's unconstrained formulation effortlessly accommodates these high-dimensional structures. ScaLip-GPU scales to both deep MLPs and CNNs, successfully extending rigorous LipSDP-equivalent certificates to architectures where traditional solvers fail entirely.

Table 2: **Experiment 3**: Lipschitz bounds and computation times for MLP and CNN architectures trained on the MNIST and CIFAR-10 datasets. OOM indicates out-of-memory errors encountered by the monolithic and layer-wise SDP solvers. Except for the Naive baseline itself, all reported bounds are normalized by the corresponding Naive bound.

Network	Metric	Naive	LipSDP	EclipsE	EclipsE-Fast	ScaLip-CPU	ScaLip-GPU
MLP-MNIST, 4FC, $n_i = 256$	Bound	8.58×10^2	0.511	0.560	0.660	0.511	0.511
	Time (s)	0.01	556.82	34.45	0.14	4.30	5.73
MLP-MNIST, 8FC, $n_i = 512$	Bound	4.06×10^7	OOM	0.101	0.277	0.0741	0.0741
	Time (s)	0.03	OOM	604.14	1.48	16.32	9.57
MLP-CIFAR, 11FC, $n_i = 2048$	Bound	4.33×10^{11}	OOM	OOM	0.136	0.00718	0.00718
	Time (s)	0.50	OOM	OOM	28.96	630.61	74.21
CNN-MNIST, 2C1FC	Bound	1.16×10^2	OOM	OOM	0.690	0.410	0.410
	Time (s)	0.05	OOM	OOM	1.73	30.45	5.08
CNN-CIFAR, 3C3FC	Bound	2.30×10^4	OOM	OOM	0.302	0.0897	0.0895
	Time (s)	0.31	OOM	OOM	18.72	729.99	55.85
CNN-CIFAR, 6C3FC	Bound	1.03×10^6	OOM	OOM	0.214	0.0637	0.0648
	Time (s)	1.09	OOM	OOM	78.63	2557.77	215.28

5 Conclusion and Future Work

In this paper, we introduced a highly scalable framework for computing the ℓ_2-Lipschitz constant of deep neural networks. By reformulating the monolithic, globally coupled LMI of standard SDP approaches into a sequence of unconstrained recursive operations, we successfully bridged the gap between the tightness of semidefinite relaxations and the computational efficiency of first-order optimization methods. Our theoretical analysis proves that this canonical change of variables eliminates the need for expensive interior-point solvers and avoids the bound degradation inherent in surrogate intermediate objectives, yielding the exact LipSDP bound.

While the global Lipschitz constant provides a foundational and mathematically rigorous metric for network robustness, it can sometimes be overly conservative for analyzing specific, *localized* input regions. A promising direction for future work is to extend this approach to compute local Lipschitz constants, which could yield tighter local robustness certificates.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing and text editing. We did not use these tools to generate research ideas, write the underlying code, or produce empirical results. The remaining disclosure categories are not applicable to this work. We have reviewed all AI-assisted content to ensure that the polished text accurately reflects our original scientific intent. We take full responsibility for the final content of this work, including all text polished with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

To ensure rigorous reproducibility of our proposed method and empirical findings, detailed pseudocode for the ScaLip optimization framework and its accelerated eigenvalue estimation procedure is provided in Algorithms 1 and 2, respectively. Furthermore, the complete source code required to reproduce the results presented in Section 4 is included in the supplementary material (and will be open-sourced upon publication). Finally, complete mathematical proofs of all theoretical claims, including the equivalence established in Theorem 1, are provided in Appendix A.

REFERENCES

MOSEK ApS. *The MOSEK Python Fusion API manual. Version 11.0.*, 2025. URL `https://docs.mosek.com/latest/pythonfusion/index.html`.

Ben Batten, Panagiotis Kouvaros, Alessio Lomuscio, and Yang Zheng. Efficient neural network verification via layer-based semidefinite relaxations and linear cuts. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2184–2190, 2021.

Robin A Brown, Edward Schmerling, Navid Azizan, and Marco Pavone. A unified view of sdp-based neural network verification through completely positive programming. In *International conference on artificial intelligence and statistics*, pp. 9334–9355. PMLR, 2022.

Hong-Ming Chiu, Hao Chen, Huan Zhang, and Richard Y Zhang. Sdp-crown: Efficient bound propagation for neural network verification with tightness of semidefinite programming. *arXiv preprint arXiv:2506.06665*, 2025.

Patrick L Combettes and Jean-Christophe Pesquet. Lipschitz certificates for layered network structures driven by averaged activation operators. *SIAM Journal on Mathematics of Data Science*, 2(2): 529–557, 2020.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. *Advances in neural information processing systems*, 32, 2019.

Mahyar Fazlyab, Manfred Morari, and George J Pappas. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, 67(1):1–15, 2022.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. *Advances in neural information processing systems*, 30, 2017.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Matt Jordan and Alexandros G Dimakis. Exactly computing the local lipschitz constant of relu networks. *Advances in Neural Information Processing Systems*, 33:7344–7353, 2020.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations (ICLR) Workshops*, 2017.

Fabian Latorre, Paul Rolland, and Volkan Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations*, 2020.

Sahel Vahedi Noori, Bin Hu, Geir Dullerud, and Peter Seiler. A complete set of quadratic constraints for repeated relu and generalizations. *IEEE Transactions on Automatic Control*, 2025.

Michael L Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 9(2):256–268, 1988.

Patricia Pauli, Aaron Havens, Alexandre Araujo, Siddharth Garg, Farshad Khorrami, Frank Allgöwer, and Bin Hu. Novel quadratic constraints for extending lipsdp beyond slope-restricted activations. In *International Conference on Learning Representations*, volume 2024, pp. 57238–57263, 2024.

Javier Rando, Jie Zhang, Nicholas Carlini, and Florian Tramèr. Position: Adversarial ML for LLMs is not making any progress. In *Forty-third International Conference on Machine Learning Position Paper Track*, 2026.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Advances in neural information processing systems*, 31, 2018.

Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in neural information processing systems*, 31, 2018.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suchi Saria, Giulio Corsi, and Luca Daniel. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29909–29921, 2021.

Zi Wang, Gautam Prakriya, and Somesh Jha. A quantitative geometric approach to neural-network smoothness. *Advances in Neural Information Processing Systems*, 35:34201–34215, 2022.

Zi Wang, Bin Hu, Aaron Havens, Alexandre Araujo, Yang Zheng, Yudong Chen, and Somesh Jha. On the scalability and memory efficiency of semidefinite programs for lipschitz constant estimation of neural networks. In *International Conference on Learning Representations*, volume 2024, pp. 19529–19555, 2024.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

Yuezhu Xu and S Sivaranjani. Eclipse: Efficient compositional lipschitz constant estimation for deep neural networks. *Advances in Neural Information Processing Systems*, 37:10414–10441, 2024.

Jie Zhang, Christian Schlarmann, Kristina Nikolić, Nicholas Carlini, Francesco Croce, Matthias Hein, and Florian Tramèr. Evaluating the robustness of the "ensemble everything everywhere" defense. *arXiv preprint arXiv:2411.14834*, 2024.

A Appendix: Proof of statements

A.1 Proof of Compositional Decomposition

For completeness, we provide the step-by-step derivation of the compositional decomposition in (4), demonstrating its equivalence to the monolithic LipSDP formulation in (3).

Theorem 2 (Theorem 3, Xu & Sivaranjani (2024))**.** *The global Lipschitz bound estimation problem in* (3)*, assuming strict feasibility of its LMI constraint (i.e., $\succ 0$), is mathematically equivalent to*

$$\begin{aligned} \min_{\Theta_i} \quad & \lambda_{\max}\left(W_D M_{D-1}^{-1} W_D^\top\right) \\ \text{s.t.} \quad & M_i \succ 0, \quad \forall i = 0, \dots, D-1, \end{aligned}$$

where the state matrices are defined recursively as follows:

$$M_i = \begin{cases} \mathbf{I}_{n_0}, & i = 0 \\ \Theta_i - \frac{1}{4}\Theta_i W_i M_{i-1}^{-1} W_i^\top \Theta_i, & i = 1, \dots, D-1. \end{cases}$$

Proof. Assuming strict feasibility of the end-to-end LMI ($\succ 0$), all diagonal blocks should be positive definite, indicating $\rho \in \mathbb{R}_{++}$ and $\Theta_i \in \mathbb{D}_{++}^{n_i}$. Since $\rho > 0$, scaling the entire matrix by $1/\rho$ preserves its definiteness. By absorbing this scaling into the unconstrained decision variables (i.e., redefining $\Theta_i \leftarrow \Theta_i/\rho$), the feasibility condition is equivalent to

$$M = \left[\begin{array}{c|cccc} \mathbf{I}_{n_0} & -\frac{1}{2}W_1^\top\Theta_1 & \mathbf{0} & \dots & \mathbf{0} \\ \hline \star & \Theta_1 & -\frac{1}{2}W_2^\top\Theta_2 & \dots & \mathbf{0} \\ \mathbf{0} & \star & \Theta_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & -\frac{1}{2}W_{D-1}^\top\Theta_{D-1} \\ \mathbf{0} & \mathbf{0} & \dots & \star & \Theta_{D-1} - \frac{1}{\rho}W_D^\top W_D \end{array}\right] \succ 0.$$

To decouple this global constraint, a Schur complement can be applied iteratively, starting from the top-left block $M_0 = \mathbf{I}_{n_0} \succ 0$. The positive definiteness of M is equivalent to $M_0 \succ 0$ and the positive definiteness of the corresponding Schur complement $S_1 = M/M_0$. The $(1,1)$ block of S_1 becomes

$$\begin{aligned} M_1 &= \Theta_1 - \left(-\frac{1}{2}\Theta_1 W_1\right) M_0^{-1} \left(-\frac{1}{2}W_1^\top\Theta_1\right) \\ &= \Theta_1 - \frac{1}{4}\Theta_1 W_1 M_0^{-1} W_1^\top \Theta_1. \end{aligned}$$

The rest of the tridiagonal structure in M remains perfectly intact:

$$S_1 = \left[\begin{array}{c|cccc} M_1 & -\frac{1}{2}W_2^\top\Theta_2 & \mathbf{0} & \dots & \mathbf{0} \\ \hline \star & \Theta_2 & -\frac{1}{2}W_3^\top\Theta_3 & \dots & \mathbf{0} \\ \mathbf{0} & \star & \Theta_3 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & -\frac{1}{2}W_{D-1}^\top\Theta_{D-1} \\ \mathbf{0} & \mathbf{0} & \dots & \star & \Theta_{D-1} - \frac{1}{\rho}W_D^\top W_D \end{array}\right] \succ 0.$$

Therefore, the condition $S_1 \succ 0$ is now equivalent to $M_1 \succ 0$ and the strict positivity of the subsequent Schur complement $S_2 = S_1/M_1$. Proceeding recursively, at step i, given that the top-left block of S_{i-1} is $M_{i-1} \succ 0$, applying the Schur complement with respect to M_{i-1} yields:

$$M_i = \Theta_i - \frac{1}{4}\Theta_i W_i M_{i-1}^{-1} W_i^\top \Theta_i \succ 0, \quad \text{for } i = 1, \dots, D-2.$$

At the final step $D-1$, applying the final Schur complement operation collapses the entire remaining matrix into a single bounding condition:

$$\begin{aligned} S_{D-1} &= \Theta_{D-1} - \frac{1}{4}\Theta_{D-1} W_{D-1} M_{D-2}^{-1} W_{D-1}^\top \Theta_{D-1} - \frac{1}{\rho}W_D^\top W_D \\ &= M_{D-1} - \frac{1}{\rho}W_D^\top W_D \succ 0. \end{aligned}$$

This condition requires $M_{D-1} \succ 0$ and $\rho M_{D-1} - W_D^\top W_D \succ 0$. By applying the properties of the Schur complement one final time to this block, the strict positivity is equivalent to

$$\begin{bmatrix} \rho \mathbf{I}_{n_D} & W_D \\ W_D^\top & M_{D-1} \end{bmatrix} \succ 0 \iff \rho \mathbf{I}_{n_D} - W_D M_{D-1}^{-1} W_D^\top \succ 0$$

This symmetric LMI holds if and only if:

$$\rho > \lambda_{\max}\left(W_D M_{D-1}^{-1} W_D^\top\right).$$

Optimizing the lower limit of ρ recovers the exact compositional objective presented in (4), which completes the proof. □

A.2 Proof of Theorem 1

This section provides the formal proof of Theorem 1, demonstrating that our unconstrained parameterization exactly preserves the optimal value of the original LMI constraint.

Proof. We show equivalence by a change of variables. To do so, let us decompose the diagonal matrix $\Theta_i \in \mathbb{D}_{++}^{n_i}$ as $\Theta_i = r_i \theta_i^2$, where $r_i \in \mathbb{R}_{++}$ and $\theta_i \in \mathbb{D}_{++}^{n_i}$. This change of variables preserves the entire feasible region, as the mapping is surjective.

Rewriting M_i for $i = 1, \ldots, D-1$ with these new variables yields:

$$M_i = r_i \theta_i^2 - \frac{1}{4}(r_i \theta_i^2) W_i (\theta_{i-1} H_{i-1} \theta_{i-1})^{-1} W_i^\top (r_i \theta_i^2).$$

Since $\theta_i \in \mathbb{D}_{++}^{n_i}$ is positive definite and therefore nonsingular, the congruence transformation $H_i = \theta_i^{-1} M_i \theta_i^{-1}$ preserves positive definiteness ($M_i \succ 0 \iff H_i \succ 0$). Fixing $H_0 = \mathbf{I}_{n_0}$, the recursive sequence for the transformed H_i could be written as

$$H_i = r_i \mathbf{I}_{n_i} - \frac{1}{4} r_i^2 \theta_i W_i \theta_{i-1}^{-1} H_{i-1}^{-1} \theta_{i-1}^{-1} W_i^\top \theta_i.$$

Substituting $\tilde{W}_i = \theta_i W_i \theta_{i-1}^{-1}$ yields exactly the recursion $H_i = r_i \mathbf{I}_{n_i} - \frac{1}{4} r_i^2 \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top$.

The constraint in (4) requires $M_i \succ 0$, which is equivalent to $H_i \succ 0$. This strict positivity holds if and only if:

$$r_i \mathbf{I}_{n_i} \succ \frac{1}{4} r_i^2 \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top.$$

Since $r_i > 0$, dividing both sides by r_i^2 preserves the strict inequality. To satisfy this LMI, the maximum eigenvalue of the right-hand side should be strictly less than $1/r_i$, yielding

$$r_i < \frac{1}{\lambda_{\max}\left(\frac{1}{4} \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top\right)}.$$

Introducing the scalar $c_i \in (0, 1)$ allows this feasible range to be parameterized as $r_i = c_i / \lambda_{\max}(\frac{1}{4} \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top)$, which guarantees $H_i \succ 0$ by construction while eliminating the need to enforce an explicit positive-definiteness constraint during optimization.

Finally, rewriting the objective function based on the new variables as $\lambda_{\max}(\tilde{W}_D H_{D-1}^{-1} \tilde{W}_D^\top)$ yields exactly the proposed unconstrained problem, thereby concluding the proof. □

A.3 Equivalence to EclipsE-Fast at Initialization

In Section 3, we state that initializing the ScaLip framework with $\theta_i = \mathbf{I}_{n_i}$ and $c_i = 0.5$ exactly recovers the EclipsE-Fast (Xu & Sivaranjani, 2024) analytical bound.

To demonstrate this, we start with the EclipsE-Fast formulation. This method restricts the decision variables Θ_i in (4) to positive scalar multiples of the identity matrix (i.e., $\Theta_i = \zeta_i \mathbf{I}_{n_i}$) and derives the closed-form optimal scalar as

$$\zeta_i = \frac{2}{\lambda_{\max}\left(W_i M_{i-1}^{-1} W_i^\top\right)},$$

Algorithm 1 ScaLip: First-order optimization of the global Lipschitz bound

Require: Original network weights $\{W_i\}_{i=1}^{D}$, learning rate η, epochs T
Initialize variables $\gamma_i \leftarrow \mathbf{0}_{n_i}$ and $u_i \leftarrow 0$ for $i = 1, \ldots, D-1$
Initialize eigenvector caches $v^{(i)} \sim \mathcal{N}(0, \mathbf{I})$ for $i = 1, \ldots, D-1$
for $t = 1, \ldots, T$ **do**
  $H_0 \leftarrow \mathbf{I}_{n_0}, \quad L_0 \leftarrow \mathbf{I}_{n_0}$
  $\gamma_0 \leftarrow \mathbf{0}_{n_0}$ ▷ **Forward Pass: Recursive State Propagation**
  for $i = 1, \ldots, D-1$ **do**
    $\tilde{W}_i \leftarrow \mathrm{diag}(\exp(\gamma_i)) W_i \mathrm{diag}(\exp(-\gamma_{i-1}))$
    $Y_i \leftarrow \mathrm{CholeskySolve}(L_{i-1}, \tilde{W}_i^\top)$ ▷ Solves $H_{i-1} Y_i = \tilde{W}_i^\top$
    $K_i \leftarrow \tilde{W}_i Y_i$ ▷ Computes $K_i = \tilde{W}_i H_{i-1}^{-1} \tilde{W}_i^\top$
    $\lambda_{\max}(K_i), v^{(i)} \leftarrow \mathrm{WarmLanczos}(K_i, v^{(i)})$ ▷ Algorithm 2
    $r_i \leftarrow \frac{\sigma(u_i)}{\frac{1}{4}\lambda_{\max}(K_i)}$
    $H_i \leftarrow r_i \mathbf{I}_{n_i} - \frac{1}{4} r_i^2 K_i$
    Attempt Cholesky factorization: $L_i \leftarrow \mathrm{Cholesky}(H_i)$
    if Cholesky fails (Not Positive Definite) **then** ▷ Guardrail Fallback
      $\lambda_{\max}(K_i) \leftarrow \mathrm{ExactMaxEigenvalue}(K_i)$
      $r_i \leftarrow \frac{\sigma(u_i)}{\frac{1}{4}\lambda_{\max}(K_i)}$
      $H_i \leftarrow r_i \mathbf{I}_{n_i} - \frac{1}{4} r_i^2 K_i$
      $L_i \leftarrow \mathrm{Cholesky}(H_i)$
    end if
  end for
  ▷ **Final Layer & Objective Calculation**
  $\tilde{W}_D \leftarrow W_D \mathrm{diag}(\exp(-\gamma_{D-1}))$
  $Y_D \leftarrow \mathrm{CholeskySolve}(L_{D-1}, \tilde{W}_D^\top)$
  $K_D \leftarrow \tilde{W}_D Y_D$
  $\rho \leftarrow \mathrm{ExactMaxEigenvalue}(K_D)$
  ▷ **Backward Pass & Unconstrained Update**
  Compute gradients $\nabla_\gamma \rho$ and $\nabla_u \rho$ via autograd ▷ Using detached eigenvectors
  Update $\{\gamma_i\}_{i=1}^{D-1}$ and $\{u_i\}_{i=1}^{D-1}$ using Adam optimizer
end for
Ensure: Optimal Lipschitz bound $L^* = \sqrt{\rho}$

with the final Lipschitz estimate derived from $\rho = \lambda_{\max}\left(W_D M_{D-1}^{-1} W_D^\top\right)$.

Under our proposed initialization, we fix $c_i = 0.5$ and $\theta_i = \mathbf{I}_{n_i}$, indicating that $\tilde{W}_i = W_i$. Consequently, our scalar r_i evaluates to

$$r_i = \frac{0.5}{\lambda_{\max}\left(\frac{1}{4} W_i H_{i-1}^{-1} W_i^\top\right)} = \frac{2}{\lambda_{\max}\left(W_i H_{i-1}^{-1} W_i^\top\right)},$$

and the corresponding squared Lipschitz bound is computed as $\lambda_{\max}\left(W_D H_{D-1}^{-1} W_D^\top\right)$. Since the base cases are identical ($H_0 = M_0 = \mathbf{I}_{n_0}$), it follows inductively that the sequence of matrices H_i exactly recovers M_i, and the exact analytical Lipschitz bound of EclipsE-Fast is recovered at epoch zero.

B Appendix: Supplementary Materials

B.1 Main Algorithm and the Validity of the Lanczos Approximation Guardrails

Algorithm 1 details the complete forward and backward passes of the ScaLip optimization framework. It outlines how the recursive state matrices are propagated, how the Cholesky guardrails ensure numerical stability during eigenvalue approximation, and how the final Lipschitz bound is optimized end-to-end.

Algorithm 2 WarmLanczos: Accelerated maximum eigenvalue estimation

Require: Symmetric matrix $H \in \mathbb{R}^{n\times n}$, cached vector $v_{\text{init}} \in \mathbb{R}^n$, max iterations k

$q \leftarrow v_{\text{init}}/\|v_{\text{init}}\|_2$
$q_{\text{prev}} \leftarrow \mathbf{0}, \beta_0 \leftarrow 0$
Initialize arrays $\alpha \in \mathbb{R}^k, \beta \in \mathbb{R}^{k-1}$, and matrix $Q \in \mathbb{R}^{n\times k}$
for $j = 1, \ldots, k$ **do**
  $Q_{:,j} \leftarrow q$
  $z \leftarrow Hq$
  $\alpha_j \leftarrow q^\top z$
  $z \leftarrow z - \alpha_j q - \beta_{j-1} q_{\text{prev}}$
  if $j < k$ **then**
    $\beta_j \leftarrow \|z\|_2$
    if $\beta_j < 10^{-8}$ **then break**
    end if
    $q_{\text{prev}} \leftarrow q$
    $q \leftarrow z/\beta_j$
  end if
end for
Construct tridiagonal matrix T from α (diagonal) and β (off-diagonal)
Compute exact eigenvalues and eigenvectors of T: $Ty = \lambda y$
Let $y_{\max}$ be the eigenvector corresponding to the maximum eigenvalue of T
$v_{\text{new}} \leftarrow Qy_{\max}$
$v_{\text{new}} \leftarrow v_{\text{new}}/\|v_{\text{new}}\|_2$
$\lambda_{\max} \leftarrow v_{\text{new}}^\top H v_{\text{new}}$ ▷ Rayleigh quotient for extreme eigenvalue

Ensure: $\lambda_{\max}$ and updated cache vector v_{new}

Although the warm-started Lanczos algorithm provides rapid convergence and is highly accurate in practice, it inherently returns a lower bound on the true maximum eigenvalue, i.e., $\tilde{\lambda}_{\max}(K_i) \le \lambda_{\max}(K_i)$. With a sufficient number of iterations, any potential approximation error is negligible. Even when the under-approximation is more pronounced, the ScaLip framework remains mathematically valid as long as the state matrix H_i maintains strict positive definiteness.

Specifically, evaluating the recursive scalar r_i in (6) using the approximated eigenvalue yields

$$r_i = \frac{c_i}{\frac{1}{4}\tilde{\lambda}_{\max}(K_i)}.$$

Algebraically, this is equivalent to evaluating the recursion using the *true* eigenvalue alongside a scaled margin parameter $\tilde{c}_i$, such that

$$r_i = \frac{\tilde{c}_i}{\frac{1}{4}\lambda_{\max}(K_i)}, \quad \text{where} \quad \tilde{c}_i = c_i \frac{\lambda_{\max}(K_i)}{\tilde{\lambda}_{\max}(K_i)}.$$

As established in the proof of Theorem 1, the recursive update inherently preserves the strict positive definiteness of H_i as long as the effective parameter satisfies $\tilde{c}_i < 1$.

Theoretical validity is therefore only compromised in extremely rare cases where a severe under-approximation from the Lanczos solver couples with an already large margin c_i, yielding an effective $\tilde{c}_i \ge 1$. If this occurs, the update may push H_i outside the positive definite cone. To safeguard against this possibility, the exact Cholesky fallback mechanism detailed in Algorithm 1 is designed to handle precisely such rare cases. By attempting a Cholesky factorization at every step, the framework immediately detects if H_i has lost positive definiteness, discards the approximation, and rigorously corrects the step using an exact eigendecomposition.

B.2 Warm-Started Lanczos Algorithm for Maximum Eigenvalue Estimation

The detailed pseudocode for our accelerated maximum eigenvalue estimation is provided in Algorithm 2. This routine leverages a warm-started Lanczos iteration, which approximates the extreme eigenvalues of a symmetric matrix H by constructing a small tridiagonal matrix T over a Krylov

subspace. By initializing the algorithm with the dominant eigenvector cached from the previous optimization step (v_{init}), the sequence converges rapidly in just a few iterations, dramatically reducing computational overhead compared to full eigendecomposition while maintaining high precision via the final Rayleigh quotient.

B.3 Extended Results for Experiment 2: Scalability on Random Networks

This section provides the comprehensive tabular data corresponding to the graphical results of Experiment 2 presented in Section 4. We report the exact computation times and computed Lipschitz bounds across all evaluated network depths and widths. These extended results further demonstrate ScaLip's ability to recover the bounds of the monolithic LipSDP formulation while highlighting its memory efficiency in high-dimensional regimes where traditional solvers encounter OOM failures.

Table 3: **Extended results for Experiment 2:** Full scalability and bound tightness results on random networks. Missing entries denote Out-of-Memory (OOM) or timeout failures (TO $>$ 30 min). For LipDiff, optimization iterations are terminated after 30 minutes, and TO is reported when even a single iteration cannot be completed within this limit.

Network	Metric	Naive	LipDiff	EclipsE-Fast	EclipsE	LipSDP	ScaLip-CPU	ScaLip-GPU
$n_i = 100, D = 5$	Bound	30.63	15.70	11.18	7.66	6.45	6.48	6.47
	Time (s)	0.01	91.83	0.02	3.01	8.82	3.24	7.03
$n_i = 100, D = 10$	Bound	909.15	801.10	94.57	43.34	28.90	29.45	29.34
	Time (s)	0.02	193.25	0.04	5.09	44.80	6.42	11.58
$n_i = 100, D = 20$	Bound	705788.70	705683.88	6058.82	1315.14	556.51	566.27	570.54
	Time (s)	0.04	1058.54	0.08	11.73	312.96	13.08	23.46
$n_i = 200, D = 5$	Bound	30.27	16.80	11.23	7.94	7.03	7.04	7.04
	Time (s)	0.02	181.26	0.09	15.31	82.52	4.75	6.39
$n_i = 200, D = 10$	Bound	878.42	770.38	95.95	45.28	32.33	32.57	32.54
	Time (s)	0.03	1024.79	0.16	35.80	526.06	9.60	12.45
$n_i = 200, D = 20$	Bound	782823.06	782813.13	7202.58	1470.25	OOM	747.24	764.78
	Time (s)	0.06	1800	0.32	81.31	OOM	22.30	24.41
$n_i = 500, D = 5$	Bound	31.17	19.03	11.30	7.69	OOM	7.23	7.22
	Time (s)	0.04	1800	0.73	242.02	OOM	11.90	9.73
$n_i = 500, D = 10$	Bound	940.04	923.14	98.28	43.06	OOM	35.80	36.01
	Time (s)	0.09	1800	1.24	543.16	OOM	22.25	16.38
$n_i = 500, D = 20$	Bound	891097.79	TO	7587.26	1357.30	OOM	897.62	901.29
	Time (s)	0.18	TO	2.44	1207.75	OOM	42.60	29.73
$n_i = 1000, D = 5$	Bound	30.97	24.24	11.29	TO	OOM	7.51	7.51
	Time (s)	0.28	1800	2.57	TO	OOM	46.26	11.85
$n_i = 1000, D = 10$	Bound	984.00	TO	102.00	TO	OOM	39.52	39.69
	Time (s)	0.25	TO	4.98	TO	OOM	85.29	22.23
$n_i = 1000, D = 20$	Bound	955751.11	TO	8138.40	TO	OOM	1067.80	1111.11
	Time (s)	0.95	TO	9.84	TO	OOM	163.97	43.09
$n_i = 2000, D = 5$	Bound	31.59	TO	11.55	OOM	OOM	7.73	7.69
	Time (s)	0.35	TO	10.33	OOM	OOM	278.34	42.82
$n_i = 2000, D = 10$	Bound	991.68	TO	102.90	OOM	OOM	40.69	40.53
	Time (s)	0.98	TO	20.34	OOM	OOM	529.95	76.86
$n_i = 2000, D = 20$	Bound	988684.33	TO	8289.77	OOM	OOM	1234.37	1202.56
	Time (s)	1.79	TO	39.09	OOM	OOM	1043.92	146.34

B.4 Network Architectures and Training Details

All neural networks evaluated in Section 4 are trained using the Adam optimizer with an initial learning rate of 10^{-3} and a standard cross-entropy loss function. Training is conducted for 20 epochs with a batch size of 128. For convolutional layers, we unroll the weight matrices into their equivalent doubly block-circulant dense representations to compute the spectral bounds. All hidden layers employ ReLU activation functions.

To formalize the architectures, we define FC(d) as a fully connected layer with d output features, and Conv(c, k, s, p) as a 2D convolutional layer with c output channels, kernel size $k \times k$, stride s, and padding p. The evaluated architectures are structured as follows:

- **MLP Architectures:** Parameterized by depth D and uniform width n_i. The network is formulated as a sequence of $D-1$ hidden FC(n_i) layers, and a final FC(10) classification head.
- **CNN-MNIST (2C1FC):** Conv$(8, 3, 2, 1) \rightarrow$ Conv$(16, 3, 2, 1) \rightarrow$ Flatten $\rightarrow$ FC(10).
- **CNN-CIFAR (3C3FC):** Conv$(16, 3, 2, 1) \rightarrow$ Conv$(32, 3, 2, 1) \rightarrow$ Conv$(32, 3, 1, 1) \rightarrow$ Flatten $\rightarrow$ FC$(512) \rightarrow$ FC$(128) \rightarrow$ FC(10).
- **CNN-CIFAR (6C3FC):** Conv$(16, 3, 2, 1) \rightarrow$ [Conv$(16, 3, 1, 1)] \times 4 \rightarrow$ Conv$(32, 3, 2, 1) \rightarrow$ Flatten $\rightarrow$ FC$(512) \rightarrow$ FC$(128) \rightarrow$ FC(10).