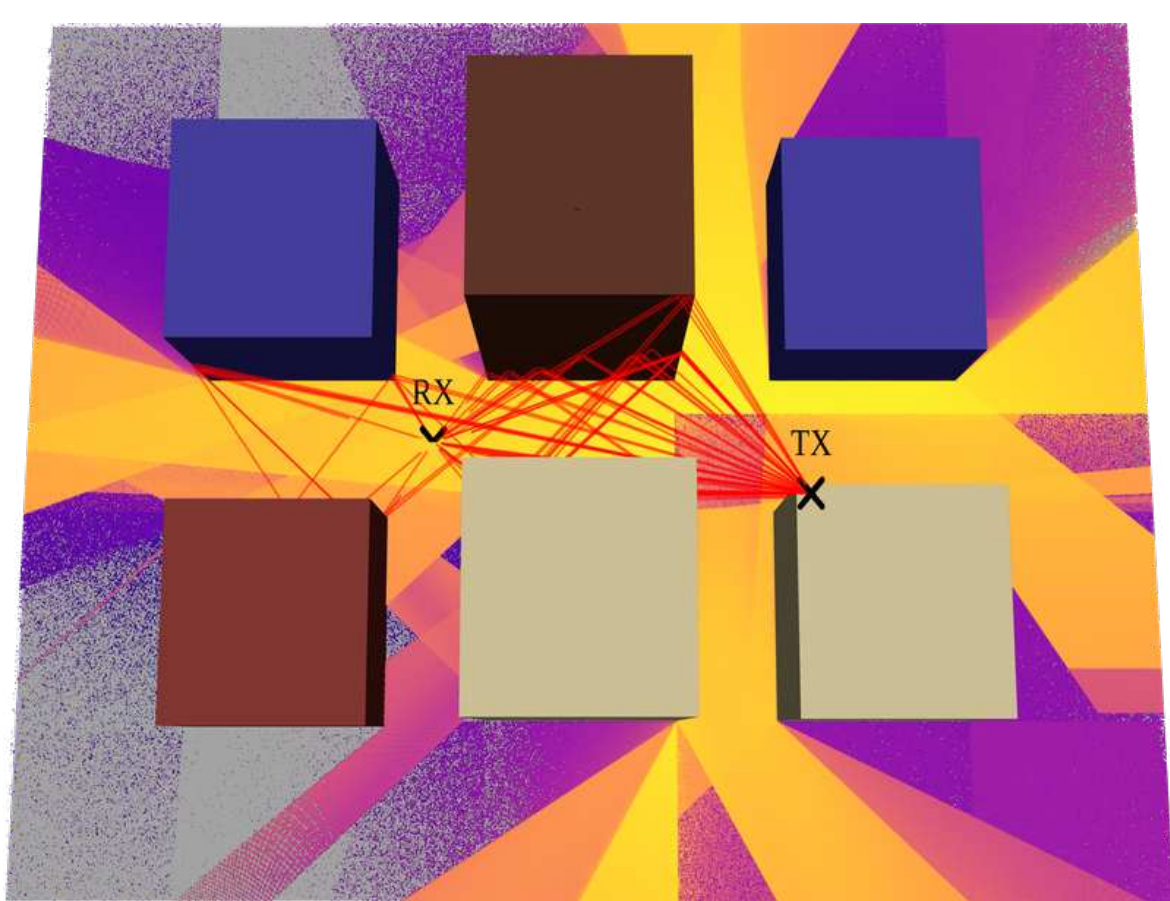


Figure 1: From traditional to GPU-accelerated differentiable ray tracing.

## Motivation

Differentiable ray tracing is a key tool for modeling radio propagation.

- **Fermat's principle:** Ray paths are extrema of optical path length.
- **Inverse problems:** Automatic differentiation (AD) enables gradient-based localization and calibration.
- **Scalability:** GPUs handle millions of concurrent ray paths.



## Challenges:

- **Speed:** Millions of paths require efficient tracing techniques.
- **Differentiability:** Inverse solvers need an end-to-end AD pipeline.
- **GPU constraints:** Performance needs static structures to limit branching, warp divergence, and memory overhead.

→ **Solution:** one unified formulation for all interaction types.

## Main contributions

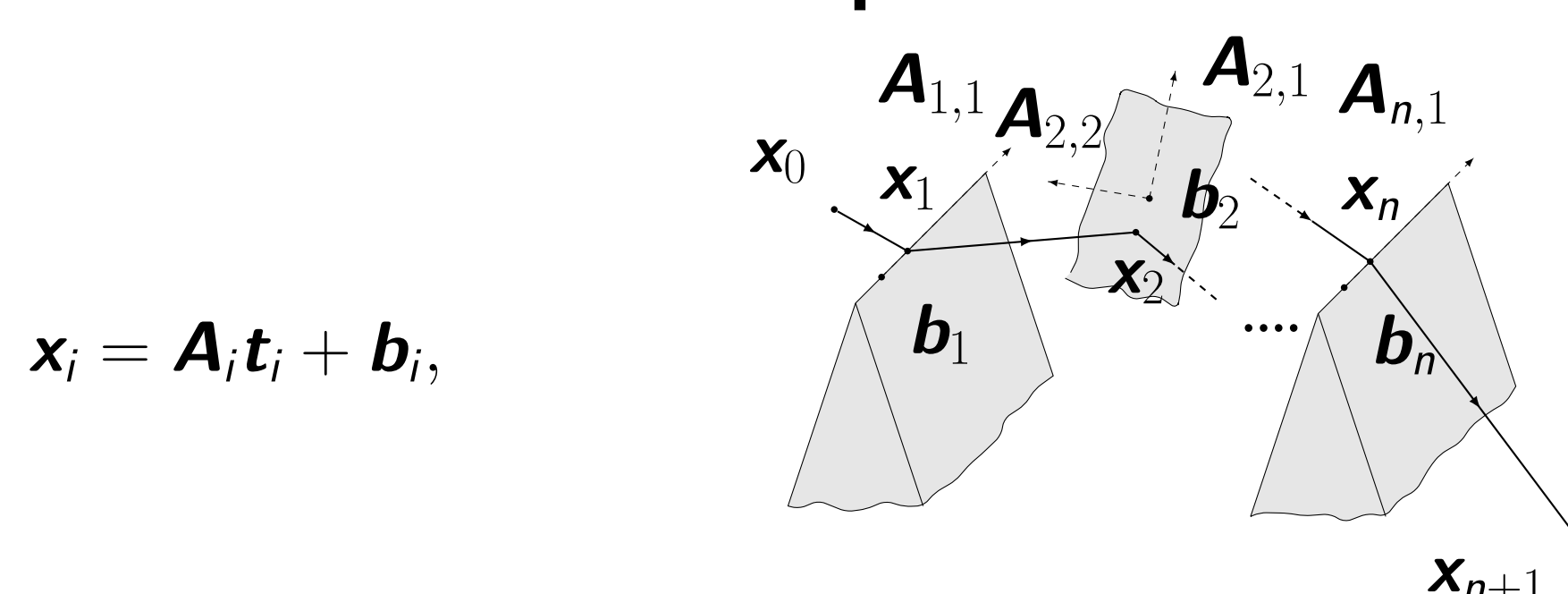
- **Unified formulation:** one optimization for any reflection–diffraction sequence.
- **GPU-friendly:** fixed-size problem across interaction types.
- **Fast gradients:** implicit differentiation avoids unrolling solver iterations.
- **Open-source:** JAX implementation in DiffeRT [1].

## Problem formulation

For planar reflectors and straight diffraction edges, Fermat's principle leads to **convex path-length minimization**

$$\mathbf{X}^* = \arg \min_{\mathbf{X} \in \mathbb{R}^{(n+2) \times 3}} L(\mathbf{X}) \quad \text{with} \quad L(\mathbf{X}) = \sum_{i=0}^n \|\mathbf{x}_{i+1} - \mathbf{x}_i\|,$$

where  $n$  is the **number of interactions** and each interaction uses a **uniform parametrization**



where  $\mathbf{A}_i$  is a local basis (2D reflections, 1D diffractions with zero-padding),  $\mathbf{b}_i$  are reference points on the objects, and  $\mathbf{t}_i$  are parameters.

**Goal:** finding optimal parameters  $\mathbf{T}^*$  minimizing the total path length.

**Key advantage:** reflections and diffractions share one tensor layout for unified GPU computation.

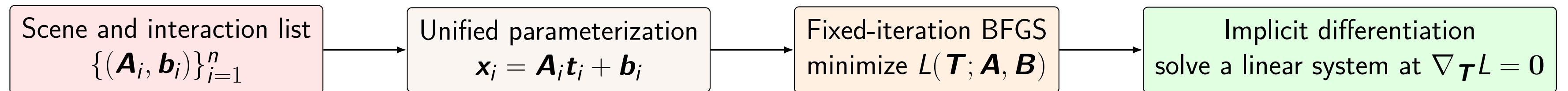


Figure 2: Single convex formulation for reflection–diffraction paths, with gradients from implicit differentiation.

## Algorithm comparison benchmark

We compare accuracy and runtime across interaction count ( $n$ ) and type (1D diffractions, 2D reflections), against gradient descent (GD), Carluccio–Albani (CA)'s approach mixing GD and Newton's method [2, 3], and L-BFGS. Our method keeps competitive accuracy with lower runtime, especially at higher  $n$ .

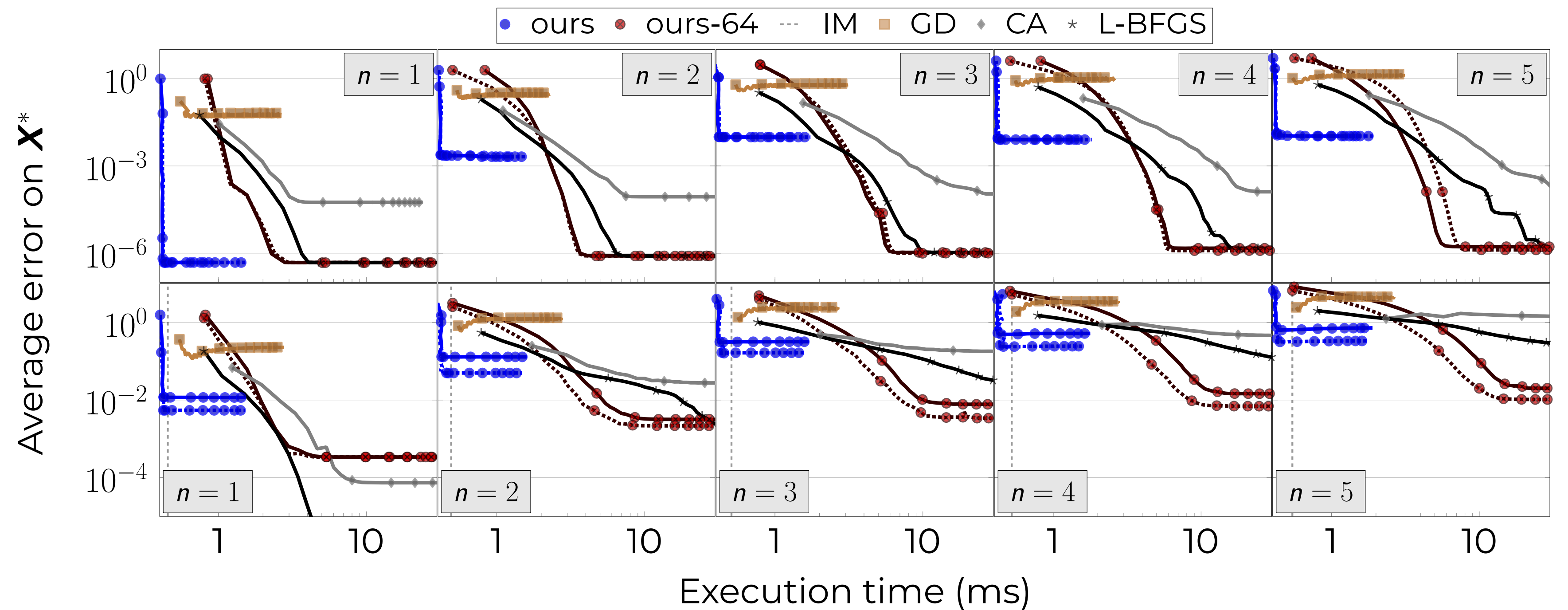


Figure 3: **Accuracy vs. execution time.** Top: 1D diffractions; bottom: 2D reflections ( $n = 1$ –5). Dashed curves denote  $d = 2$  diffractions. We report standard (ours) and 64 fixed-point iterations (ours-64). IM is shown as a vertical line (exact solution).

## Solver

### BFGS iterative procedure (fixed $K$ iterations):

- 1 At iteration  $k$ , compute  $\mathbf{g}_k = \nabla_{\mathbf{T}} L(\mathbf{T}_k)$  and direction  $\mathbf{p}_k = -\mathbf{H}_k \mathbf{g}_k$ .
- 2 Choose step size  $\alpha_k$  with **fixed-point line search** (start from  $\alpha_0 = 1$ ).
- 3 Update parameters  $\mathbf{T}_{k+1} = \mathbf{T}_k + \alpha_k \mathbf{p}_k$ .
- 4 Update inverse-Hessian approximation  $\mathbf{H}_{k+1}$  from  $\mathbf{s}_k = \mathbf{T}_{k+1} - \mathbf{T}_k$  and  $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$ .

- **GPU-friendly:** Fixed iteration count ensures uniform execution across threads.
- **Robust:** Stable with diffractions and close interaction points.
- **Tunable:** Increase fixed-point iterations to improve accuracy as needed.

## Implicit Differentiation

At convergence, we have  $\nabla_{\mathbf{T}} L(\mathbf{T}^*(\theta); \theta) = 0$ .

**Key idea:** solve one linear system with the Hessian at the solution, instead of backpropagating through iterations. This yields exact gradients at low overhead.

## Extension to Refraction

The method extends to refraction by multiplying path segments with local refractive indices  $n_i$

$$L(\mathbf{X}) = \sum_{i=0}^n n_i \|\mathbf{x}_{i+1} - \mathbf{x}_i\|.$$

The objective **stays convex**, so the same solver applies.

## Open-Source Code

**Fully reproducible:** code is on GitHub and integrated in DiffeRT.



GitHub



DiffeRT

## Implicit vs Automatic Differentiation

**Key advantage:** implicit differentiation avoids unrolling solver iterations, giving up to **10x** faster backward passes than AD while keeping exact gradients.

Why it matters:

- Solves one linear system at convergence, independent of iteration count.
- Lowers memory use by avoiding intermediate solver states.
- Enables faster inverse design and optimization.

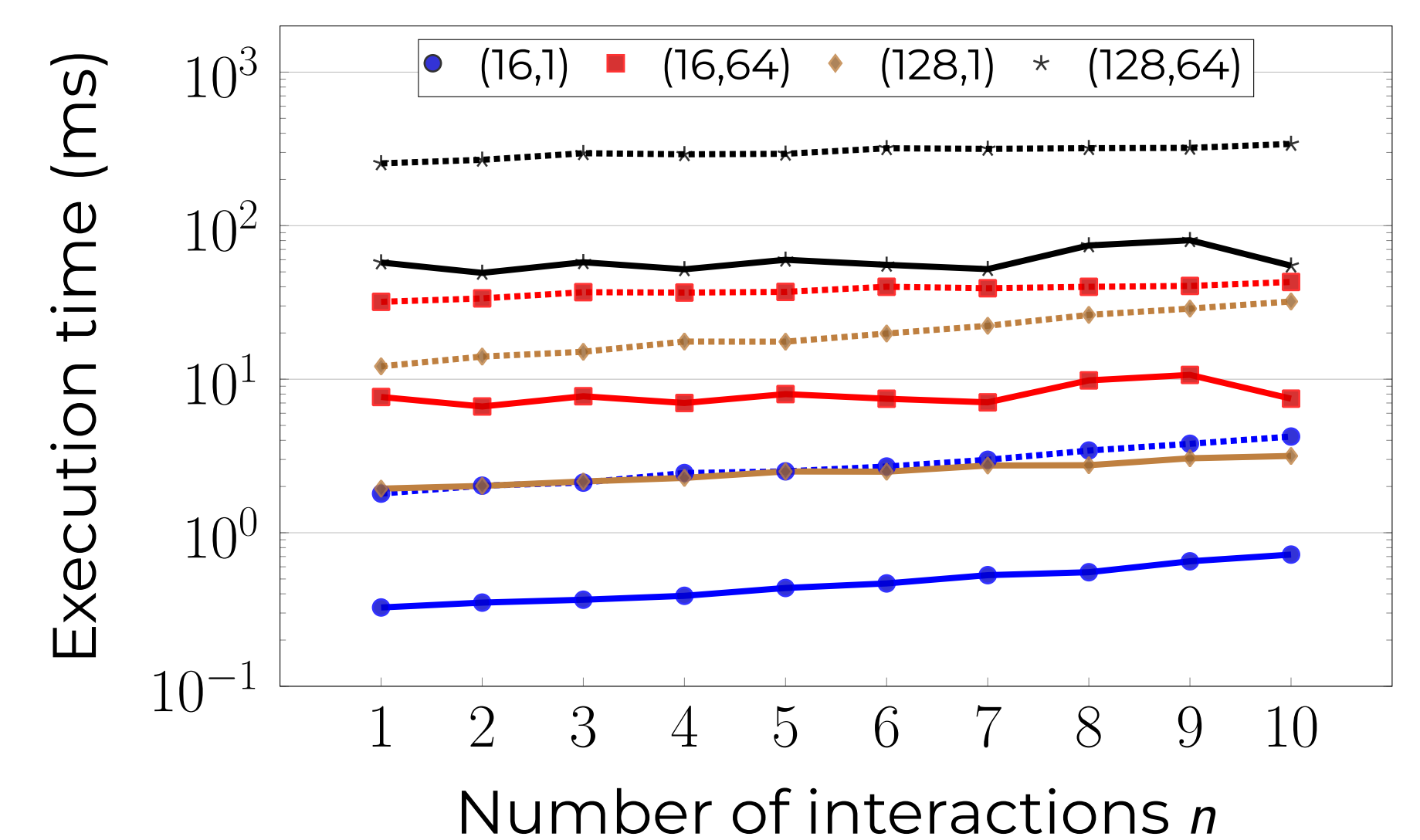


Figure 4: **Implicit (solid) vs. AD (dashed)** backward-pass runtime. Implicit differentiation is iteration-independent; AD scales linearly with iteration count.

## Conclusion

We unify reflection–diffraction ray tracing as one optimization problem compatible with GPU acceleration and efficient differentiation.

**Future work:** using second-order cone programming for better robustness and convergence, improved initialization, and refined line search.

## References

- [1] J. Eertmans, C. Oestges and L. Jacques, "Demonstrating DiffeRT: An Open-Source Library for Optimizing Radio Networks with Differentiable Ray Tracing," *2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, Barcelona, Spain, pp. 1–2, 2025.
- [2] G. Carluccio and M. Albani, "An efficient ray tracing algorithm for multiple straight wedge diffraction," *IEEE Trans. Antennas Propagat.*, vol. 56, no. 11, pp. 3534–3542, Nov. 2008.
- [3] F. Puggelli, G. Carluccio, and M. Albani, "A novel ray tracing algorithm for scenarios comprising pre-ordered multiple planar reflectors, straight wedges, and vertexes," *IEEE Trans. Antennas Propagat.*, vol. 62, no. 8, pp. 4336–4341, Aug. 2014.