UCLouvain

Fast, Differentiable, GPU-Accelerated Ray Tracing for Multiple Diffraction and Reflection Paths

Jérome Eertmans, Sophie Lequeu, Benoît Legat, Laurent Jacques, Claude Oestges

ICTEAM, Université catholique de Louvain - EuCAP 2026

1. Motivation

1. Motivation

- Wireless simulation is moving from static CPU pipelines to real-time GPU pipelines.

1. Motivation

- Wireless simulation is moving from static CPU pipelines to real-time GPU pipelines.
- Differentiable ray tracing transforms propagation from analysis to optimization.

1. Motivation

- Wireless simulation is moving from static CPU pipelines to real-time GPU pipelines.
- Differentiable ray tracing transforms propagation from analysis to optimization.
- Key examples: Sionna RT and our DiffeRT ecosystem.

1. Motivation

- Wireless simulation is moving from static CPU pipelines to real-time GPU pipelines.
- Differentiable ray tracing transforms propagation from analysis to optimization.
- Key examples: Sionna RT and our DiffeRT ecosystem.
- Modern JIT + autodiff stacks (TensorFlow, PyTorch, JAX, DrJIT) unlock scalability.

1. Motivation

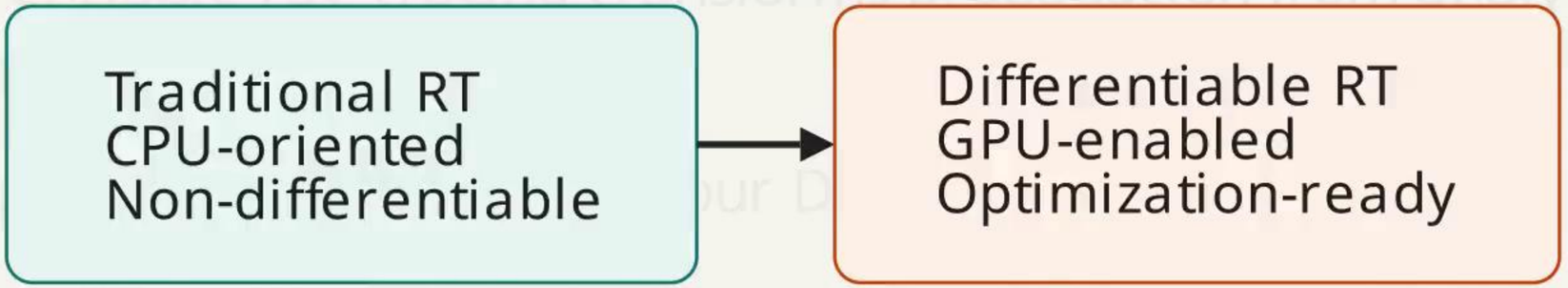
- Wireless simulation is moving from static CPU pipelines to real-time GPU pipelines.
- Differentiable ray tracing transforms propagation from analysis to optimization.
- Key examples: Sionna RT and our DiffeRT ecosystem.
- Modern JIT + autodiff stacks (TensorFlow, PyTorch, JAX, DrJIT) unlock scalability.
- If path tracing is not GPU-aware and differentiable, these opportunities vanish.

1. Motivation

- Wireless simulation is moving from static CPU pipelines to real-time GPU pipelines.
- Differentiable ray tracing transforms propagation from analysis to optim
- Key e ur DiffeRT ecosystem.
- Modern JIT + autodiff stacks (TensorFlow, PyTorch, JAX, DrJIT) unlock scalability.
- If path tracing is not GPU-aware and differentiable, these opportunities vanish.

Traditional RT
CPU-oriented
Non-differentiable

1. Motivation



Talk Roadmap

1. Motivation
2. State of the Art
3. Current limitations and our approach
4. Results
5. Ongoing and future research

2. State of the Art

2. State of the Art

- Image method, exact and extremely fast for pure reflections.

2. State of the Art

- Image method, exact and extremely fast for pure reflections.
- Min-Path-Tracing (MPT) or Fermat-based minimization, for paths including diffraction, refraction, etc.

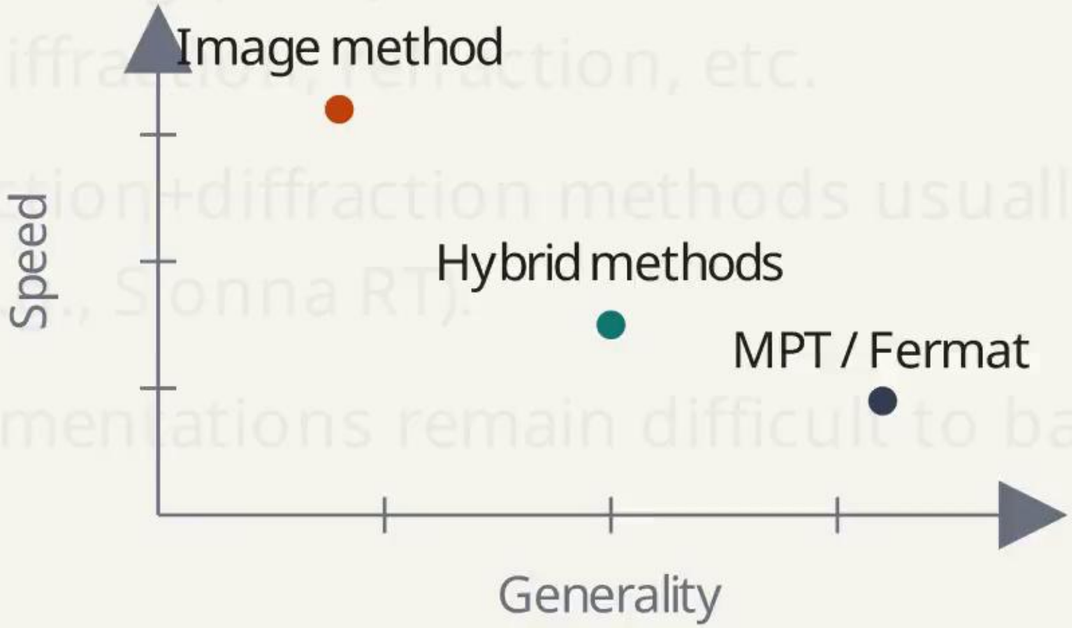
2. State of the Art

- Image method, exact and extremely fast for pure reflections.
- Min-Path-Tracing (MPT) or Fermat-based minimization, for paths including diffraction, refraction, etc.
- Mixed reflection+diffraction methods usually combine separate pipelines (e.g., Sionna RT).

2. State of the Art

- Image method, exact and extremely fast for pure reflections.
- Min-Path-Tracing (MPT) or Fermat-based minimization, for paths including diffraction, refraction, etc.
- Mixed reflection+diffraction methods usually combine separate pipelines (e.g., Sionna RT).
- Most implementations remain difficult to batch efficiently on GPUs.

2. State of the Art



3. Current Limitations and Our Approach

3. Current Limitations and Our Approach

- Limitation: branching logic depends on interaction order (R/R/D/...)

3. Current Limitations and Our Approach

- Limitation: branching logic depends on interaction order (R/R/D/...)
- Limitation: variable-size optimization makes vectorization harder.

3. Current Limitations and Our Approach

- Limitation: branching logic depends on interaction order (R/R/D/...)
- Limitation: variable-size optimization makes vectorization harder.
- Approach: single convex minimization template for arbitrary sequences.

3. Current Limitations and Our Approach

- Limitation: branching logic depends on interaction order (R/R/D/...)
- Limitation: variable-size optimization makes vectorization harder.
- Approach: single convex minimization template for arbitrary sequences.
- Approach: same tensorized shape for reflection and diffraction paths.

3. Current Limitations and Our Approach

- Limitation: branching logic depends on interaction order (R/R/D/...)
- Limitation: variable-size optimization makes vectorization harder.
- Approach: single convex minimization template for arbitrary sequences.
- Approach: same tensorized shape for reflection and diffraction paths.
- Result: cleaner GPU batching and simpler differentiable integration.

3. Methodology I: Problem Formulation

3. Methodology I: Problem Formulation

- As introduced by Carluccio G. and Albani M. (CA) et al., we formulate ray tracing as a convex optimization problem.

3. Methodology I: Problem Formulation

- As introduced by Carluccio G. and Albani M. (CA) et al., we formulate ray tracing as a convex optimization problem.
- However, we do not use the image method for intermediate reflections.

3. Methodology I: Problem Formulation

- As introduced by Carluccio G. and Albani M. (CA) et al., we formulate ray tracing as a convex optimization problem.
- However, we do not use the image method for intermediate reflections.
- Instead: unified parameterization for reflections and diffractions.

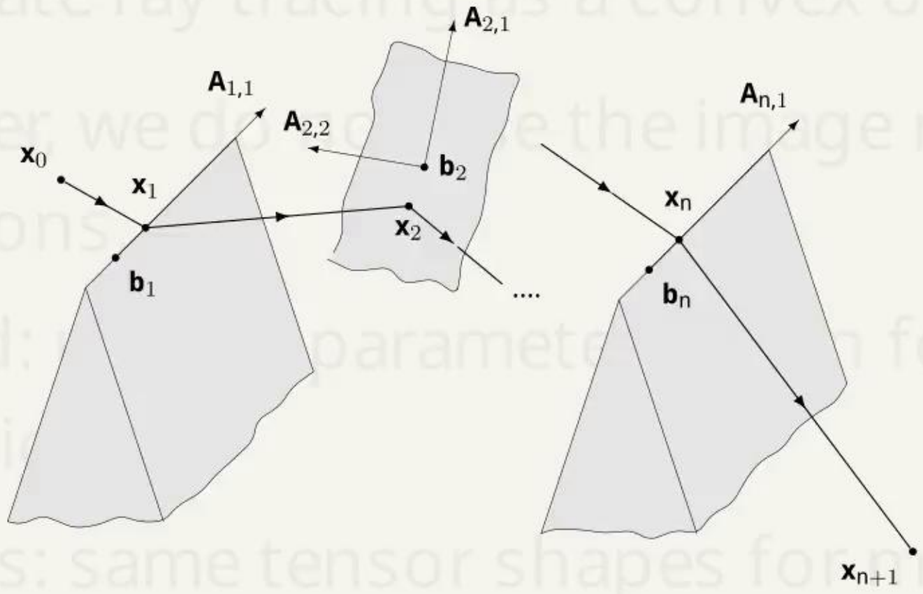
3. Methodology I: Problem Formulation

- As introduced by Carluccio G. and Albani M. (CA) et al., we formulate ray tracing as a convex optimization problem.
- However, we do not use the image method for intermediate reflections.
- Instead: unified parameterization for reflections and diffractions.
- Benefits: same tensor shapes for mixed interaction sequences.

3. Methodology I: Problem Formulation

- As introduced by Carluccio G. and Albani M. (CA) et al., we formulate ray tracing as a convex optimization problem.
- However, we do not use the image method for intermediate reflections.
- Instead: unified parameterization for reflections and diffractions.
- Benefits: same tensor shapes for mixed interaction sequences.
- Benefits: efficient GPU batching and simpler differentiable integration.

3. Methodology I: Problem Formulation



$$\mathbf{T}^* = \arg\min_{\mathbf{T}} L(\mathbf{T}; \mathbf{A}, \mathbf{B})$$

with

$$L(\mathbf{T}; \mathbf{A}, \mathbf{B}) = \sum_i \|\Delta \mathbf{x}_i\|$$

and

$$\mathbf{x}_i = \mathbf{A}_i \mathbf{t}_i + \mathbf{b}_i$$

Aparte: Handling Refraction

Aparte: Handling Refraction

- Not shown in the paper: refractive index can be included directly.

Aparte: Handling Refraction

- Not shown in the paper: refractive index can be included directly.
- Problem remains convex.

Aparte: Handling Refraction

- Not shown in the paper: refractive index can be included directly.
- Problem remains convex.

$$\min_{\mathbf{T}} \sum_{i} n_i \, \|\Delta \mathbf{x}_i\|$$

Methodology II: BFGS Solver

Methodology II: BFGS Solver

- Initialize $\mathbf{T}_0$ and $\mathbf{B}_0$ (typically identity).

Methodology II: BFGS Solver

- Initialize $\mathbf{T}_0$ and $\mathbf{B}_0$ (typically identity).
- Solve $\mathbf{B}_k\mathbf{p}_k = -\nabla_T L(T_k)$ for the descent direction.

Methodology II: BFGS Solver

- Initialize $\mathbf{T}_0$ and $\mathbf{B}_0$ (typically identity).
- Solve $\mathbf{B}_k\mathbf{p}_k = -\nabla_T L(T_k)$ for the descent direction.
- Use line-search to pick step size α_k for $T_{k+1} = T_k + \alpha_k p_k$.

Methodology II: BFGS Solver

- Initialize $\mathbf{T}_0$ and $\mathbf{B}_0$ (typically identity).
- Solve $\mathbf{B}_k\mathbf{p}_k = -\nabla_T L(T_k)$ for the descent direction.
- Use line-search to pick step size α_k for $T_{k+1} = T_k + \alpha_k p_k$.
- Set $s_k = T_{k+1} - T_k$ and $y_k = \nabla L(T_{k+1}) - \nabla L(T_k)$,
  then update $\mathbf{B}_k$ with BFGS.

Methodology II: BFGS Solver

- Initialize $\mathbf{T}_0$ and $\mathbf{B}_0$ (typically identity).
- Solve $\mathbf{B}_k\mathbf{p}_k = -\nabla_T L(T_k)$ for the descent direction.
- Use line-search to pick step size α_k for $T_{k+1} = T_k + \alpha_k p_k$.
- Set $s_k = T_{k+1} - T_k$ and $y_k = \nabla L(T_{k+1}) - \nabla L(T_k)$, then update $\mathbf{B}_k$ with BFGS.
- Run a fixed number of iterations K on GPU to avoid warp divergence and idle threads.

Methodology II: BFGS Solver

Why BFGS over mixed Newton/GD?

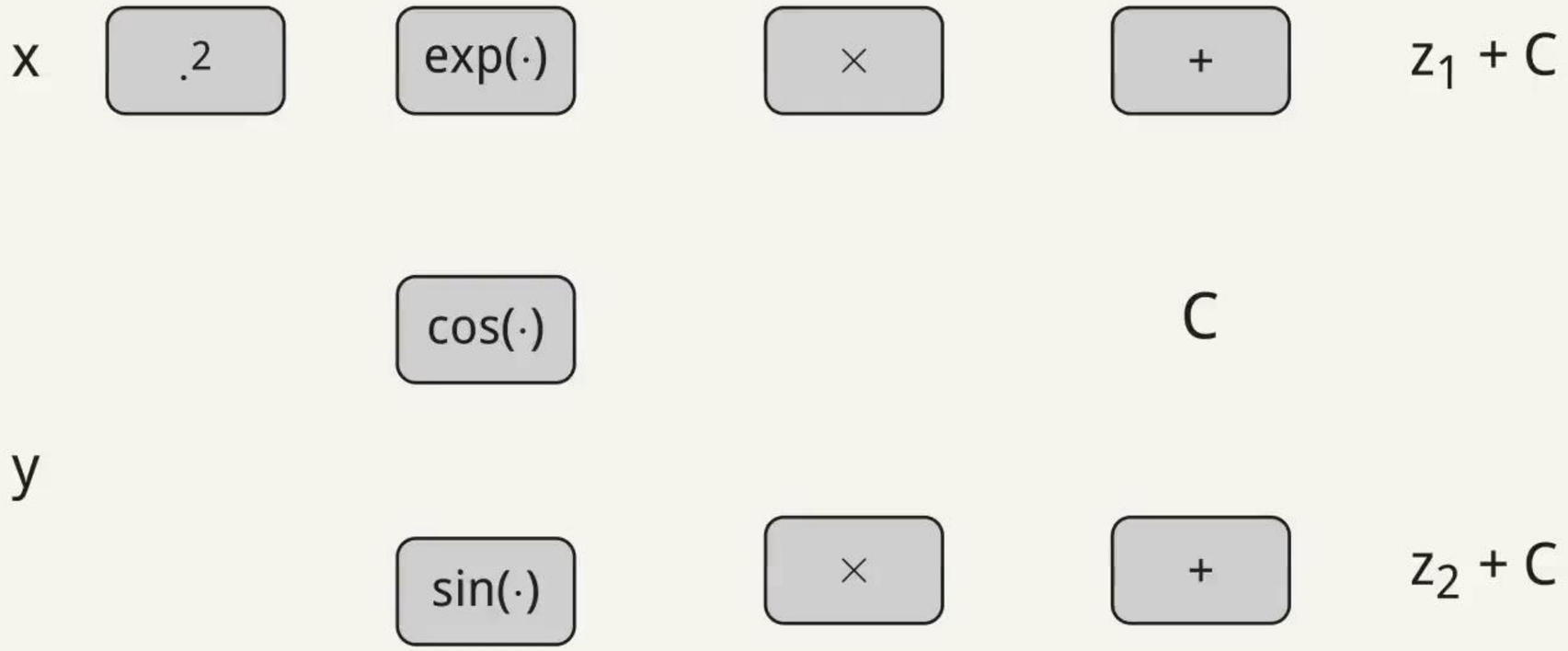
CA's Newton step is sensitive to ill-conditioned Hessians. This appears frequently with zero-padded diffraction dimensions.

BFGS avoids inverting the true Hessian and supports stronger line-search.

$$B_{k+1} = B_k + \frac{y_k y_k^\top}{y_k^\top s_k} - \frac{B_k s_k s_k^\top B_k}{s_k^\top B_k s_k}$$

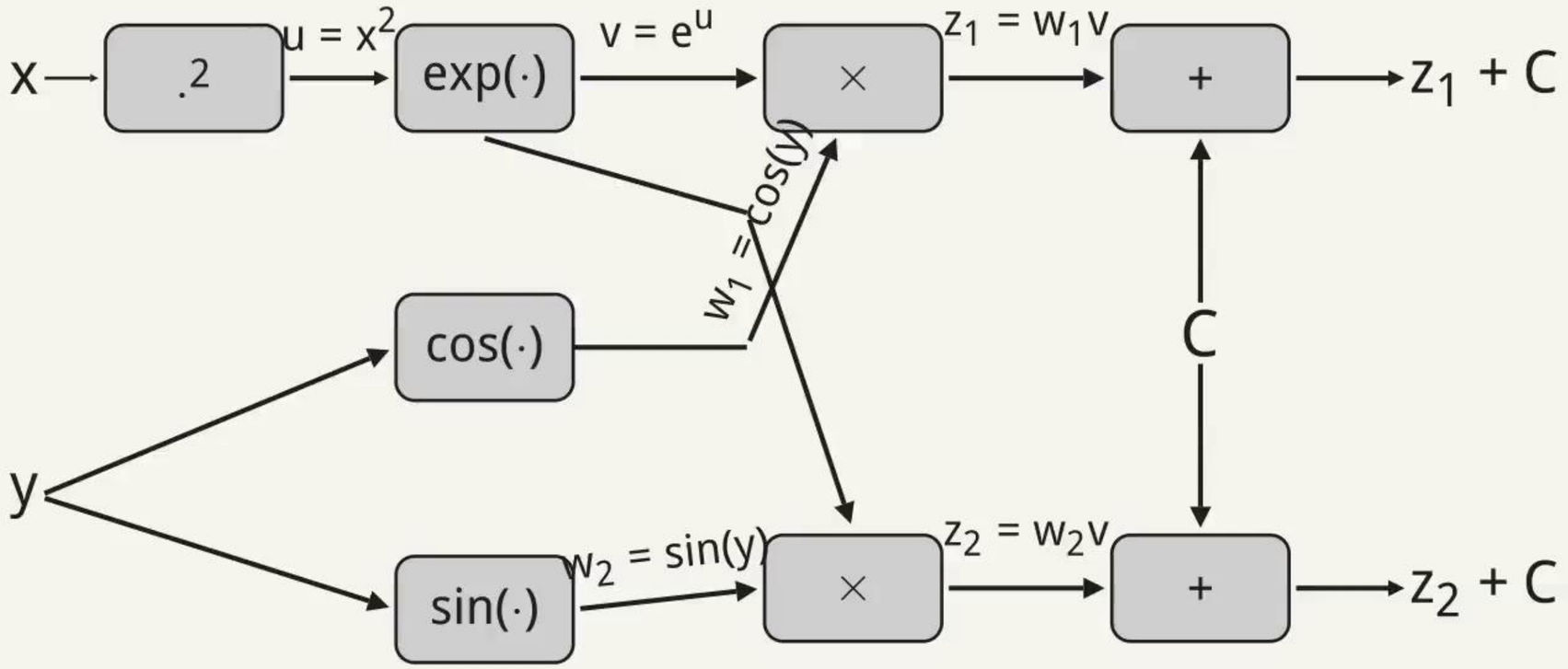
Methodology III: Reverse-Mode AD

$$f(x, y) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} \cos(y)e^{x^2} + C \\ \sin(y)e^{x^2} + C \end{bmatrix}$$



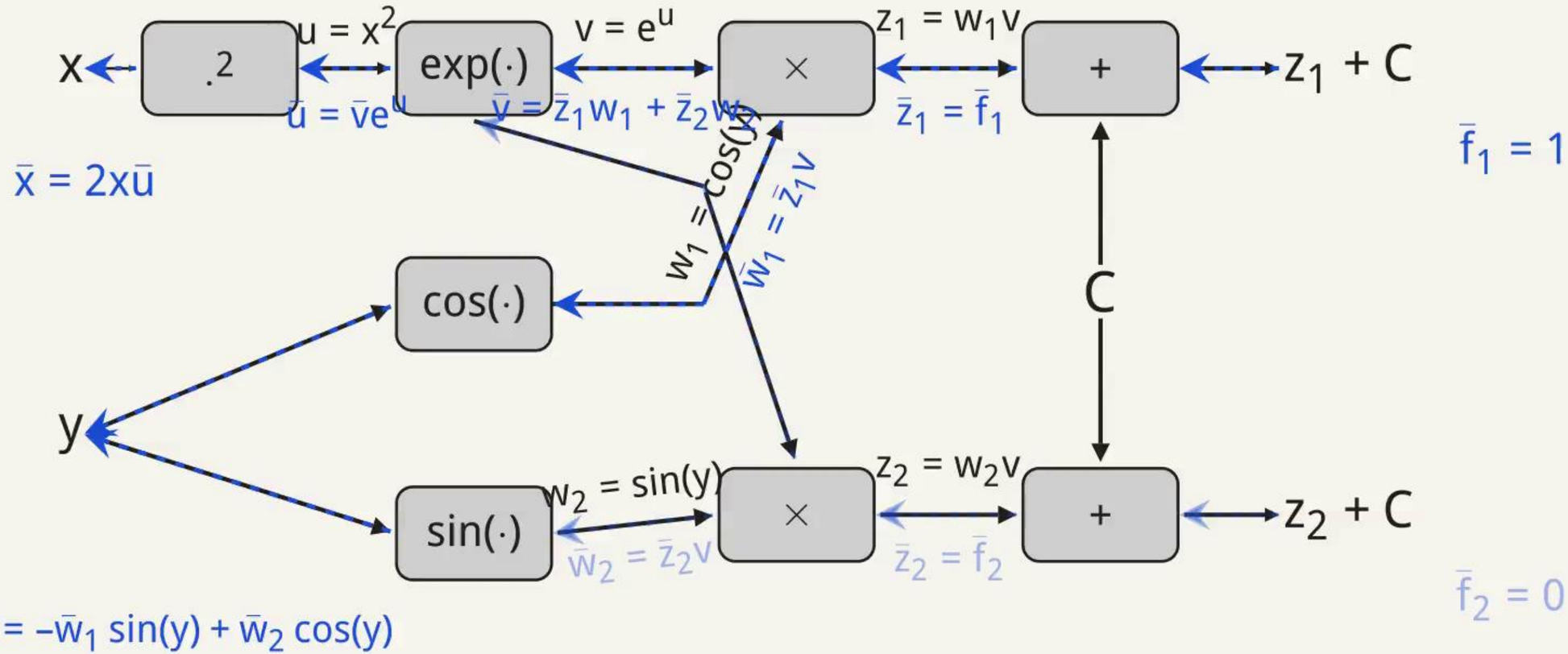
Methodology III: Reverse-Mode AD

$$f(x, y) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} \cos(y)e^{x^2} + C \\ \sin(y)e^{x^2} + C \end{bmatrix}$$



Methodology III: Reverse-Mode AD

$$f(x, y) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} \cos(y)e^{x^2} + C \\ \sin(y)e^{x^2} + C \end{bmatrix}$$



Methodology IV: Implicit Differentiation

Methodology IV: Implicit Differentiation

- Reverse-mode AD stores intermediate states from the forward pass.

Methodology IV: Implicit Differentiation

- Reverse-mode AD stores intermediate states from the forward pass.
- For K solver iterations, unrolling creates O(K) memory and O(K) backward traversal.

Methodology IV: Implicit Differentiation

- Reverse-mode AD stores intermediate states from the forward pass.
- For K solver iterations, unrolling creates O(K) memory and O(K) backward traversal.
- For batched ray-path optimization, this can dominate memory and runtime.

Methodology IV: Implicit Differentiation

- Reverse-mode AD stores intermediate states from the forward pass.
- For K solver iterations, unrolling creates O(K) memory and O(K) backward traversal.
- For batched ray-path optimization, this can dominate memory and runtime.
- Use the implicit function theorem at the converged solution instead of unrolling.

Methodology IV: Implicit Differentiation

- Reverse-mode AD stores intermediate states from the forward pass.
- For K solver iterations, unrolling creates O(K) memory and O(K) backward traversal.
- For batched ray-path optimization, this can dominate memory and runtime.
- Use the implicit function theorem at the converged solution instead of unrolling.
- Result: gradients without storing all iterative states.

Methodology IV: Implicit Differentiation

Unrolled iterative reverse-mode

memory $\propto K$

backward time $\propto K$

$$\nabla_{\mathbf{T}} L(\mathbf{T}^*; \theta) = \mathbf{0}$$

$$\frac{\partial \mathbf{T}^*}{\partial \theta} = -H^{-1} \frac{\partial}{\partial \theta} \nabla_{\mathbf{T}} L$$

4. Results: Benchmark Setup

4. Results: Benchmark Setup

- 1000 paths solved in parallel on RTX 3070 (single precision).

4. Results: Benchmark Setup

- 1000 paths solved in parallel on RTX 3070 (single precision).
- Interaction counts from n=1 to n=5.

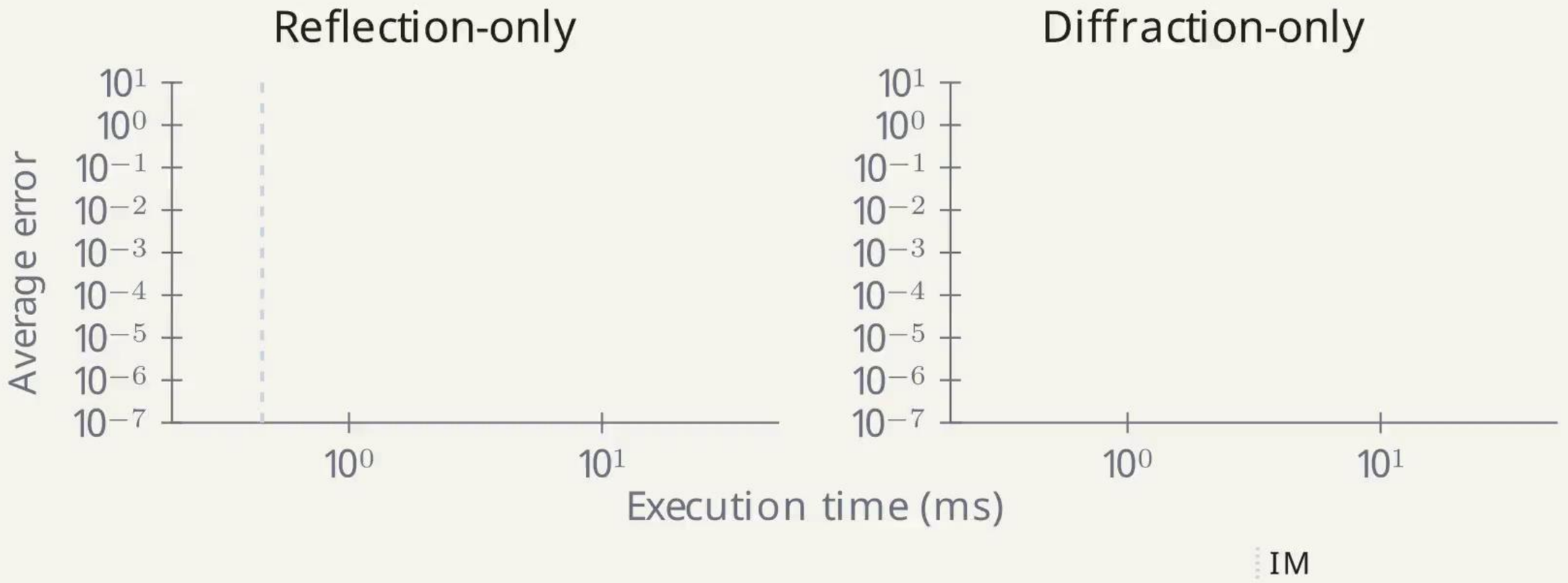
4. Results: Benchmark Setup

- 1000 paths solved in parallel on RTX 3070 (single precision).
- Interaction counts from n=1 to n=5.
- Baselines: IM, GD, CA, L-BFGS.

4. Results: Benchmark Setup

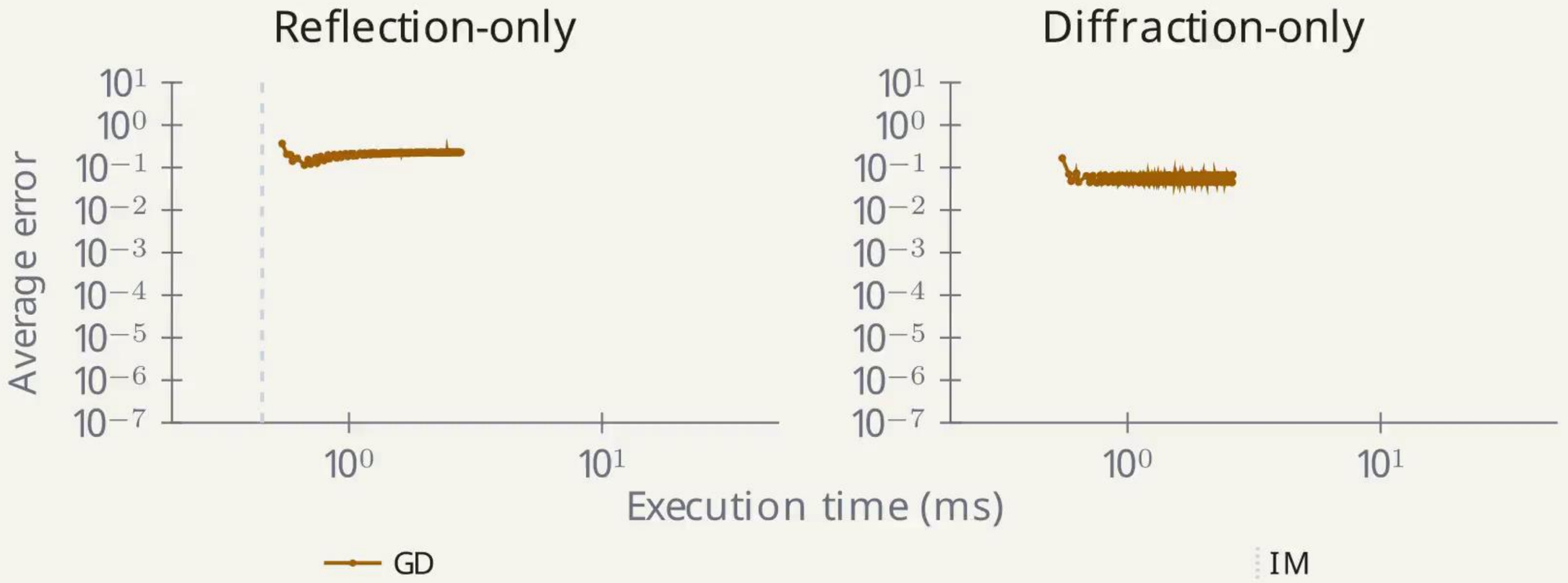
- 1000 paths solved in parallel on RTX 3070 (single precision).
- Interaction counts from n=1 to n=5.
- Baselines: IM, GD, CA, L-BFGS.
- Metrics: runtime and average interaction-point error.

Results: Accuracy vs Runtime



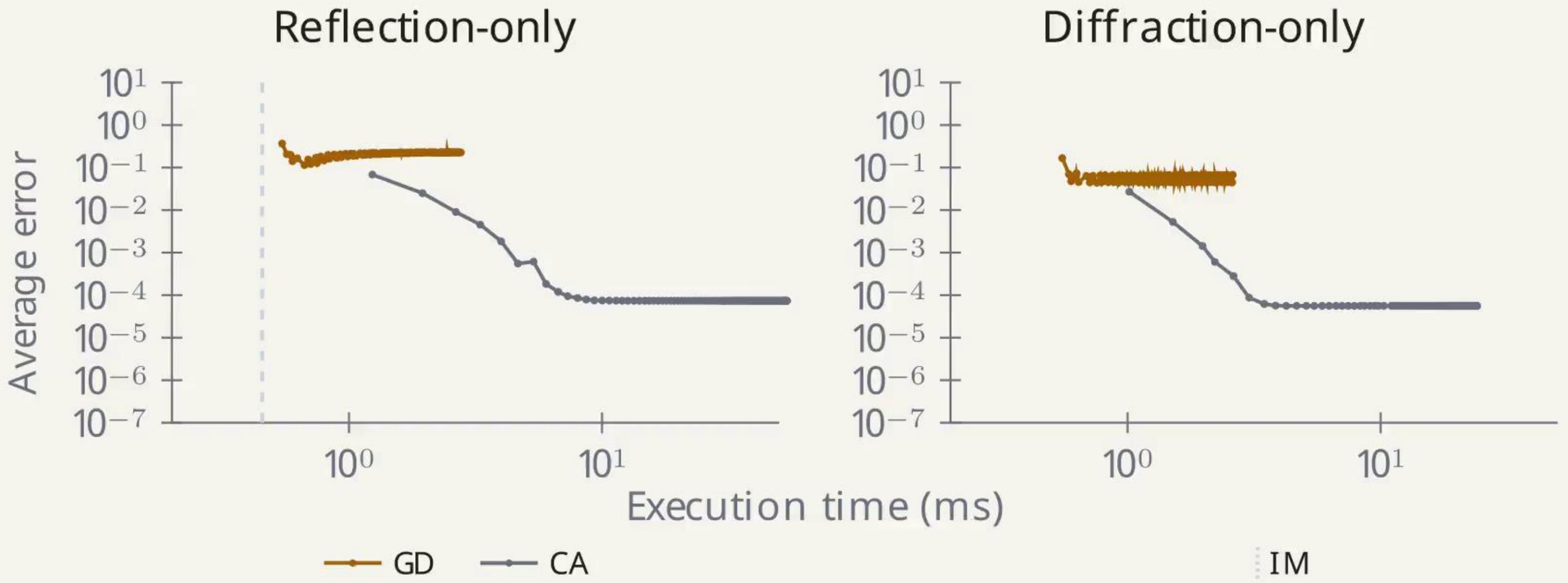
n = 1

Results: Accuracy vs Runtime



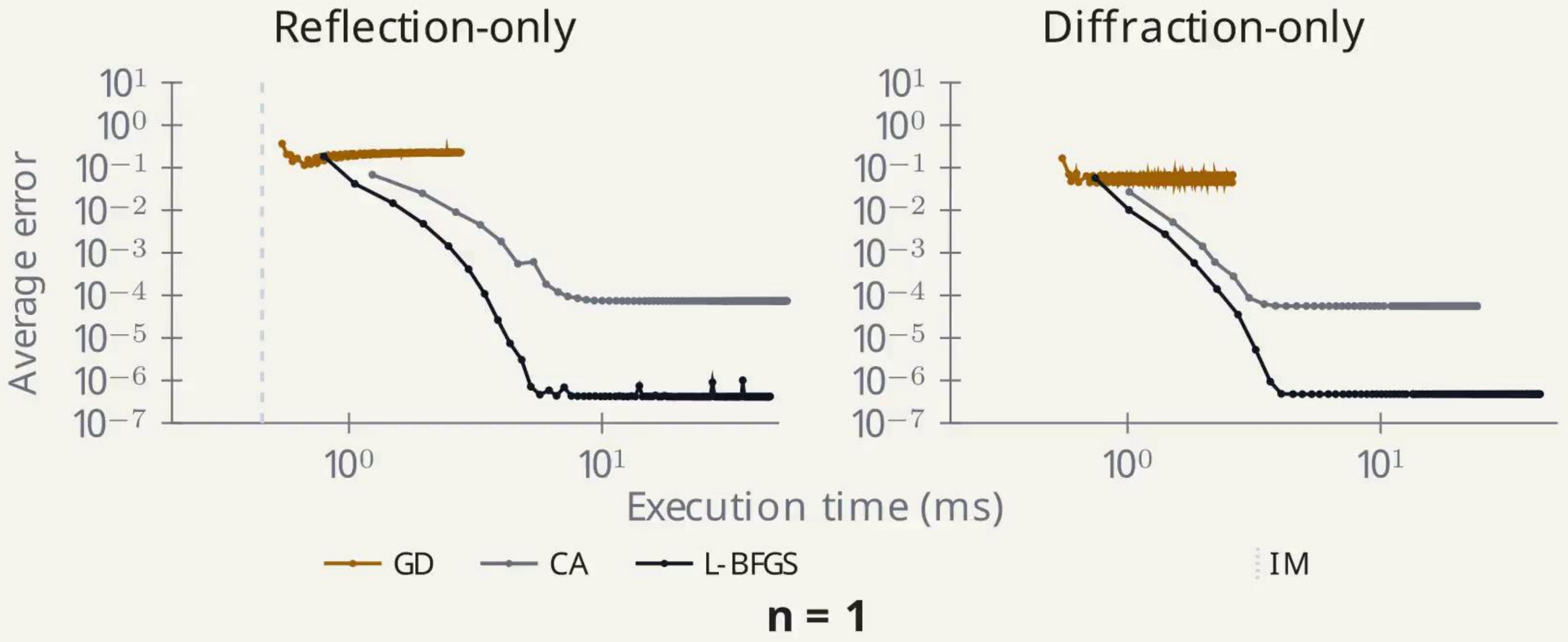
n = 1

Results: Accuracy vs Runtime



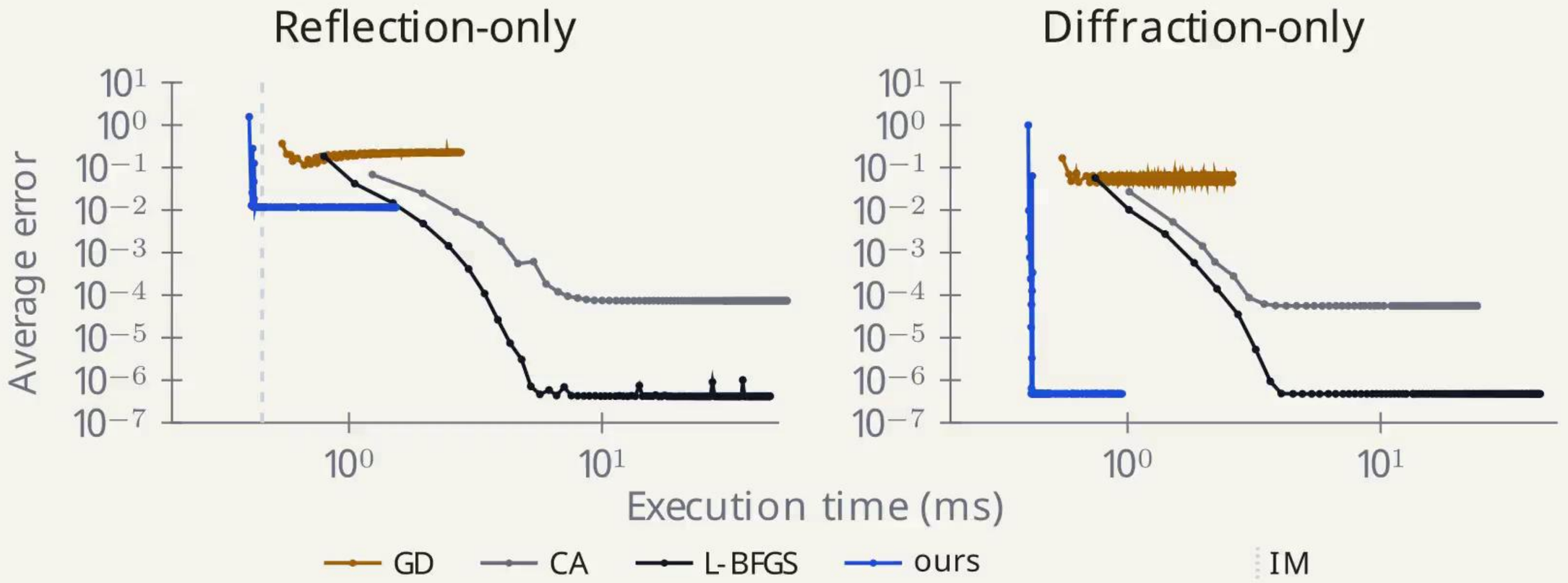
n = 1

Results: Accuracy vs Runtime



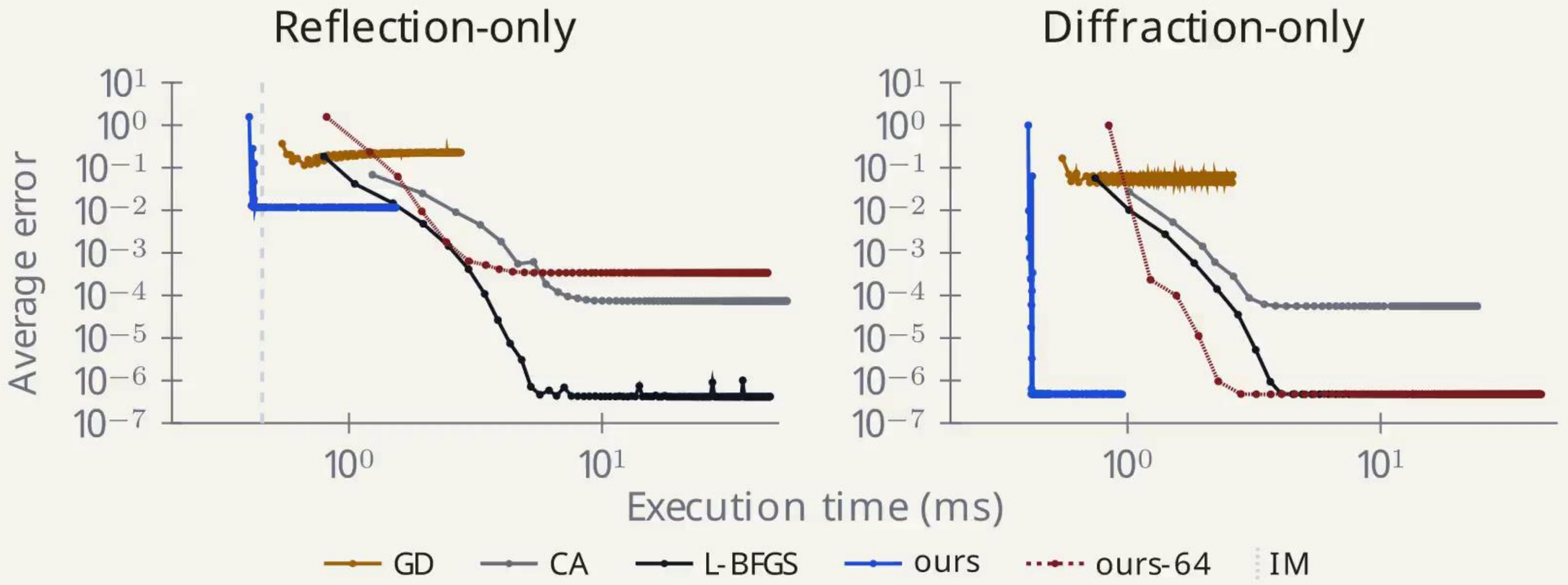
n = 1

Results: Accuracy vs Runtime



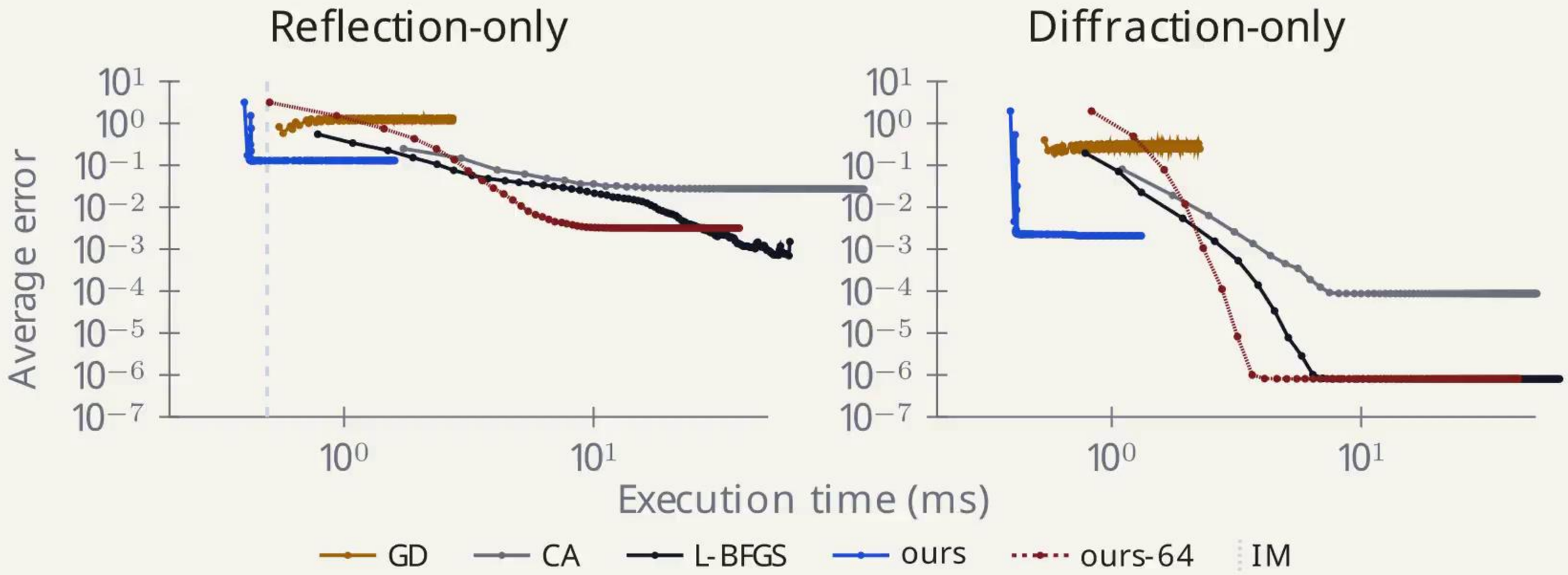
n = 1

Results: Accuracy vs Runtime



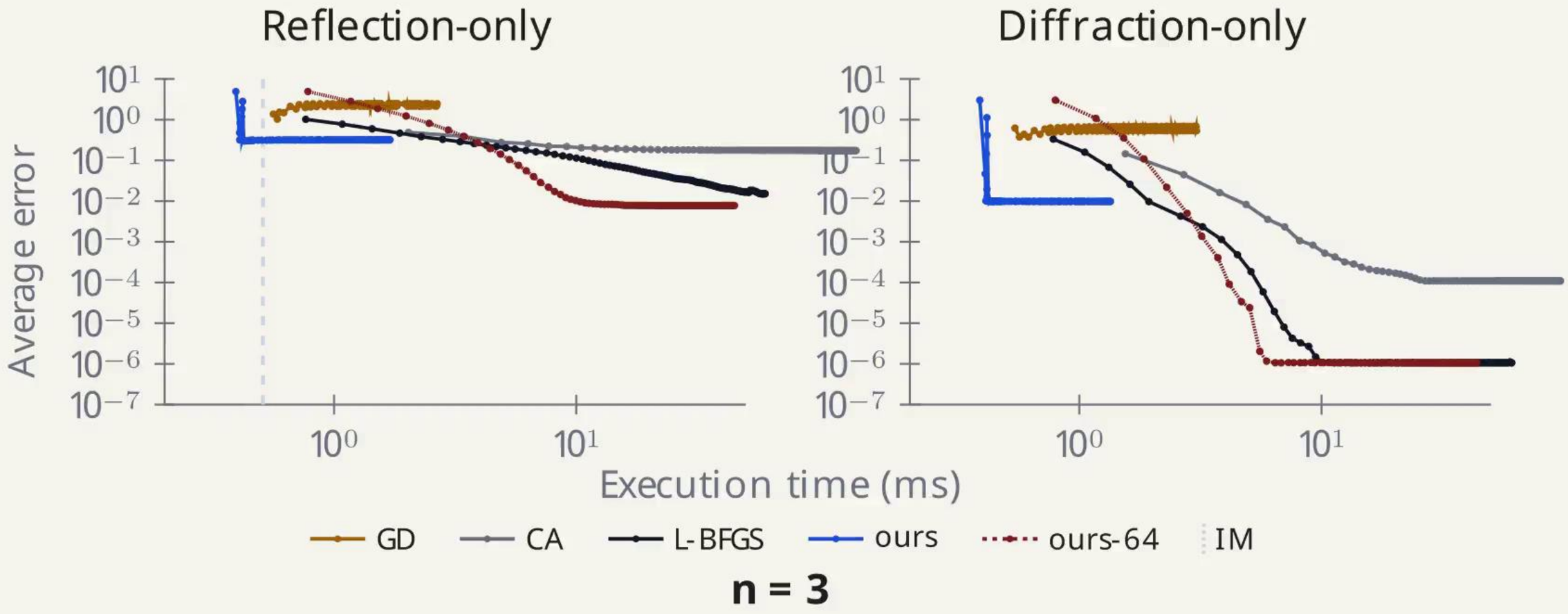
n = 1

Results: Accuracy vs Runtime



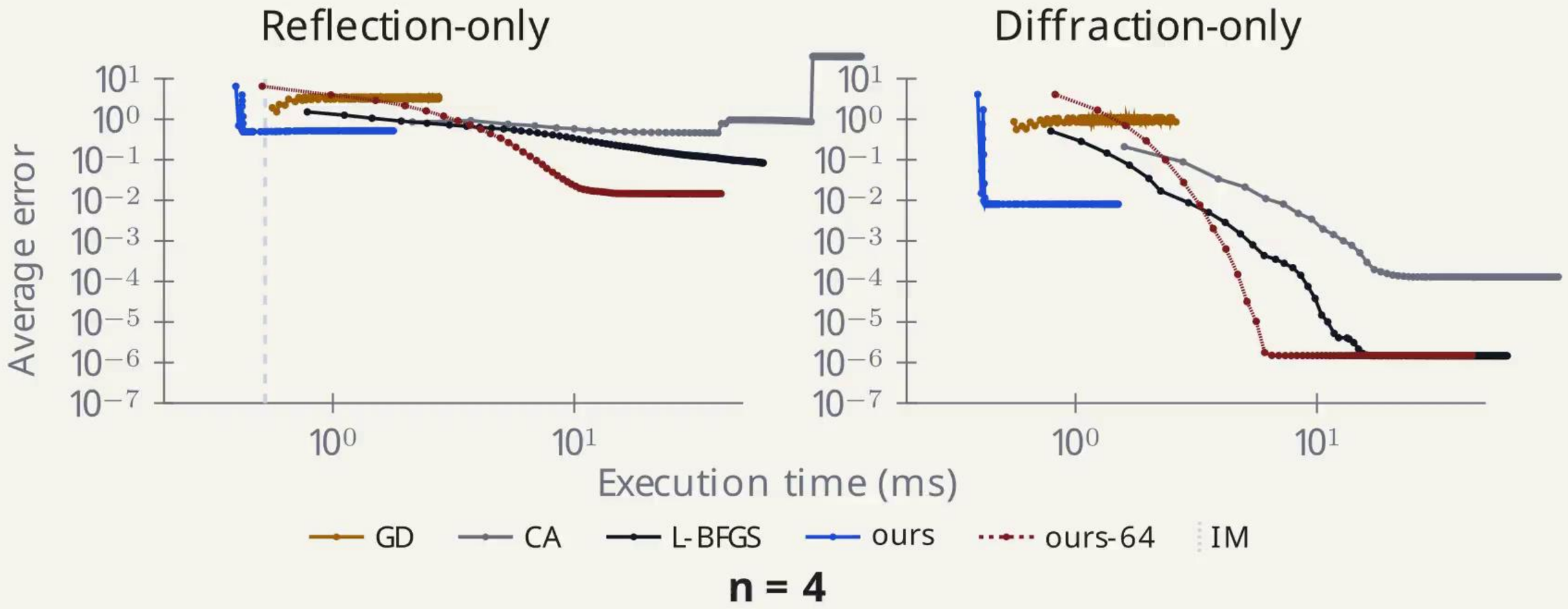
n = 2

Results: Accuracy vs Runtime



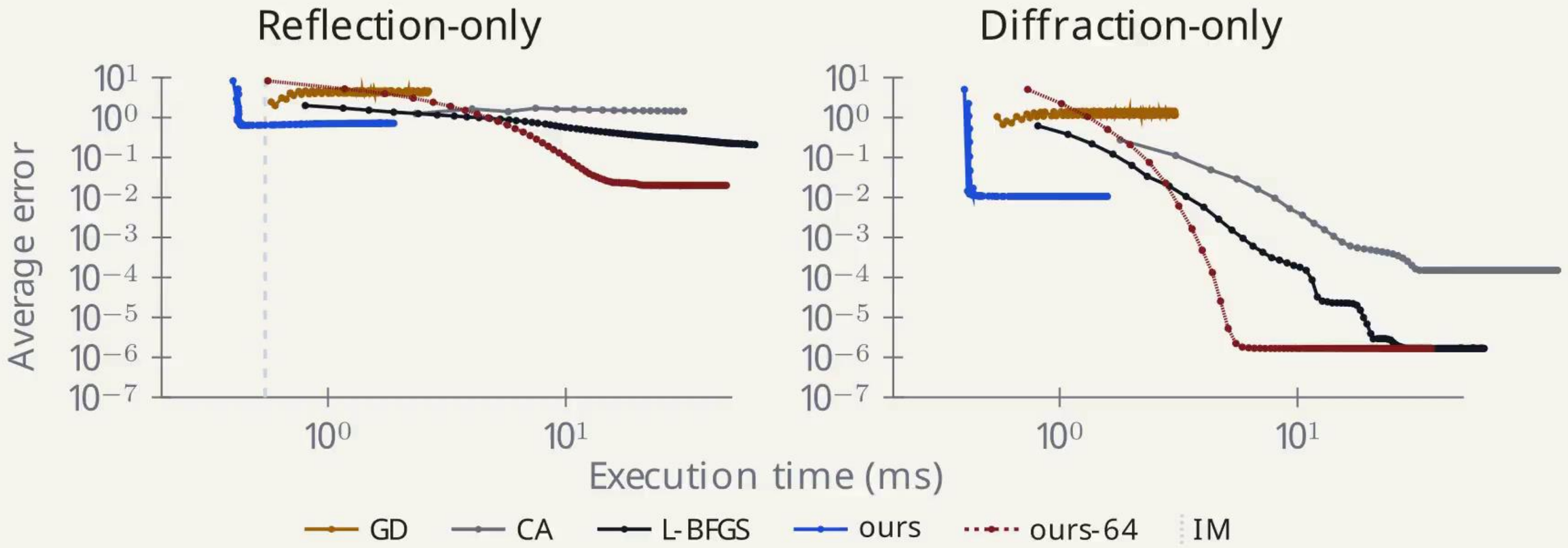
n = 3

Results: Accuracy vs Runtime



n = 4

Results: Accuracy vs Runtime



n = 5

5. Ongoing and Future Research

5. Ongoing and Future Research

- (In progress) Explore second-order cone (SOCP) formulations for better convergence.

5. Ongoing and Future Research

- (In progress) Explore second-order cone (SOCP) formulations for better convergence.
- (In progress) Port double-precision (64-bit) SOCP CPU solvers to a single-precision (32-bit) GPU implementations.

5. Ongoing and Future Research

- (In progress) Explore second-order cone (SOCP) formulations for better convergence.
- (In progress) Port double-precision (64-bit) SOCP CPU solvers to a single-precision (32-bit) GPU implementations.
- Investigate high-performance GPU solvers (open-source availability is still limited).

5. Ongoing and Future Research

- (In progress) Explore second-order cone (SOCP) formulations for better convergence.
- (In progress) Port double-precision (64-bit) SOCP CPU solvers to a single-precision (32-bit) GPU implementations.
- Investigate high-performance GPU solvers (open-source availability is still limited).
- Improve candidate initialization and line-search policies.

5. Ongoing and Future Research

- (In progress) Explore second-order cone (SOCP) formulations for better convergence.
- (In progress) Port double-precision (64-bit) SOCP CPU solvers to a single-precision (32-bit) GPU implementations.
- Investigate high-performance GPU solvers (open-source availability is still limited).
- Improve candidate initialization and line-search policies.

Better solvers exist in theory; practical open GPU implementations remain the bottleneck.

Thank you

Happy to answer questions!

DiffeRT

GitHub Implementation

Presentation with Manim Slides, an open-source tool