A Fully First-Order Layer for Differentiable Optimization

Zihao Zhao[1] Kai-Chia Mo[2] Shing-Hei Ho[1] Brandon Amos[3] Kai Wang[1]

[1]School of Computational Science and Engineering, Georgia Institute of Technology
[2]Independent Researcher
[3]Reflection AI

ICML 2026

Problem statement and assumptions

We train through a convex optimization layer:

$$\min_x F(x) = f(x, y^\star(x)), \quad y^\star(x) \in \arg\min_y g(x, y) \quad \text{s.t. } h(x, y) \leq 0,\ e(x, y) = 0.$$

Training needs

$$\nabla_x F = \nabla_x f + \left(\frac{dy^\star}{dx}\right)^\top \nabla_y f.$$

Bottleneck: compute the sensitivity term $dy^\star/dx$.

Standard assumptions

- f: smooth and Lipschitz.
- g: smooth; strongly convex in y.
- LICQ holds at the lower-level solution.
- Active constraints can be identified.

Standard approach: implicit function theorem

Implicit differentiation solves a linear system from KKT conditions:

$$\frac{d(y^\star, \lambda^\star, \nu^\star)}{dx} = -\left(\nabla_{(y,\lambda,\nu)} G\right)^{-1} \nabla_x G.$$

What works

- Exact hypergradient
- Strong for small structured problems

What breaks

- Large KKT solves
- Hessian-like information
- Memory-heavy at scale

Reformulate as bilevel optimization

View the optimization layer as the lower level:

$$\min_{x} F(x) = f(x, y^\star) \quad \text{s.t.} \quad y^\star \in \arg\min_{y:\, h(x,y)\le 0,\ e(x,y)=0} g(x, y).$$

Existing first-order bilevel idea (Kwon et al. [2])

Use a penalty-based Lagrangian proxy:

$$\mathcal{L}_\alpha(x, y) = f(x, y) + \alpha\big(g(x, y) - g(x, y^\star)\big).$$

Then $\nabla_x \mathcal{L}_\alpha$ approximates $\nabla_x F$.

Advantage: Avoid computing Hessians or solving the KKT system.

Motivation: inequalities are the bottleneck

Kornowski et al. [3], Theorem 5.3

Given accuracy $\epsilon > 0$, their linearly constrained inexact gradient oracle returns

$$\|\tilde{\nabla} F(x) - \nabla F(x)\| \leq \epsilon$$

within $\tilde{\mathcal{O}}(\epsilon^{-1})$ gradient oracle evaluations.

To reach a (δ, ϵ)-Goldstein stationary point, this leads to $\tilde{\mathcal{O}}(\delta^{-1}\epsilon^{-4})$ overall complexity, ϵ^{-1} slower than the optimal $\tilde{\mathcal{O}}(\delta^{-1}\epsilon^{-3})$ [4].

Can we keep first-order differentiation, but recover the optimal rate?

Our key idea: remove inactive constraints locally

Ghost lower level (only has equalities)

$$\min_y \tilde{g}(x,y) \quad \text{s.t. } \tilde{h}(x,y) = 0,$$

$$\tilde{g}(x,y) = g(x,y) + \langle \lambda^\star, h(x,y)\rangle + \langle \nu^\star, e(x,y)\rangle,$$

$$\tilde{h}(x,y) = \begin{bmatrix} e(x,y) \\ \nabla_x h_{\mathcal{I}}(\bar{x},y^*)(x-\bar{x}) + \nabla_y h_{\mathcal{I}}(\bar{x},y^*)(y-y^*) \end{bmatrix}.$$

Keep active constraints as equalities; drop inactive ones locally.

Active-set equivalence: $\nabla F(\bar{x}) = \nabla \tilde{F}(\bar{x})$.

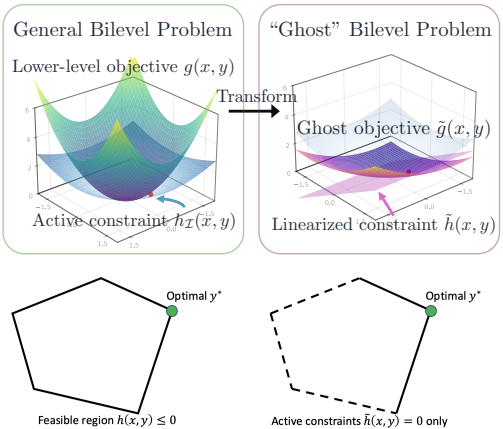

Fig. 1. Illustration of our ghost bilevel reform.

FFOLayer: backward pass **without** KKT differentiation

Forward pass:

$$y^\star = \arg\min_y g(x,y) \quad \text{s.t. } h(x,y) \le 0,\ e(x,y) = 0.$$

Backward pass solves a **perturbed ghost problem**:

$$y_\delta^\star \in \arg\min_{y:\tilde{h}(x,y)=0} \tilde{g}(x,y) + \delta f(x,y).$$

Finite-difference hypergradient

The perturbation gives a first-order estimate of $\left(\frac{dy^\star}{dx}\right)^\top \nabla_y f(x,y^\star)$:

$$v_x := \frac{1}{\delta}\Big(\nabla_x[\tilde{g}(x,y_\delta^*) + \langle \lambda_\delta^*, \tilde{h}(x,y^*)\rangle] - \nabla_x[\tilde{g}(x,y^*)]\Big).$$

Then the layer returns

$$\tilde{\nabla} F(x) = \nabla_x f(x,y^\star) + v_x.$$

What does this buy us?

Theory

Hypergradient oracle: Under previous assumptions,

$$\|\tilde{\nabla} F(x) - \nabla F(x)\| \leq \epsilon \text{ in } O(\log(1/\epsilon))$$

first-order oracle calls.

Convergence:

$$\tilde{O}(\delta^{-1}\epsilon^{-3})$$

to a (δ, ϵ)-Goldstein stationary point.

- Extends the guarantee to **general convex constraints**.
- Matches the **optimal** nonsmooth nonconvex rate.

Implementation

- Drop-in replacement for existing layers like CvxpyLayer.
- Solver-agnostic: Compatible with any advanced black-box solvers like GUROBI and MOSEK.

```
1  import cvxpy as cp
2  # define problem in CVXPY
3  x = cp.Variable(n)
4  u = cp.Parameter(n)
5  obj = cp.Minimize(0.5 * cp.sum_squares(x) + cp.sum(u
       * x))
6  constraints = [x == 0, x <= 1]
7  prob = cp.Problem(obj, constraints)
8  # layer = CvxpyLayer(prob, parameters=[u], variables
       =[x])
9  layer = FFOLayer(prob, parameters=[u], variables=[
       x])
10 x_star = layer(u_value)
```

Empirical results: same convergence behavior

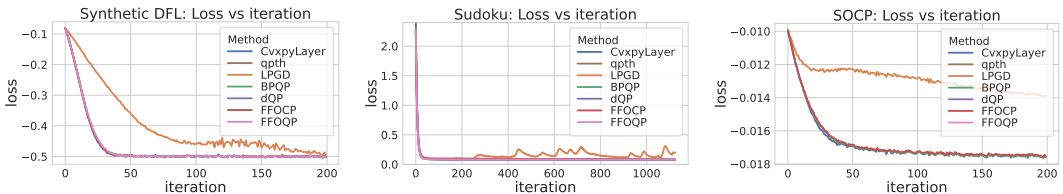

Fig. 2. Training convergence on synthetic DFL, Sudoku, and SOCP.

Takeaway: approximate first-order hypergradients track exact-gradient training behavior.

Empirical results: faster and more scalable

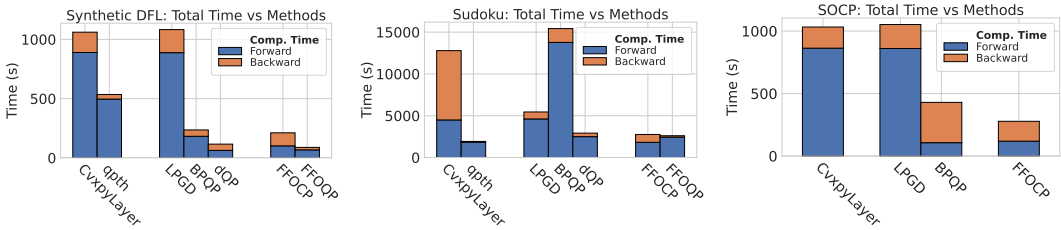

Fig. 3. Forward/backward computation cost.

Takeaway: avoid KKT factorization, reduce wall-clock and memory pressure.

Differentiable optimization can be solved by fully first-order methods!

FFOLayer: Fully first-order. Solver-agnostic. Scalable.

Try FFOLayer: https://github.com/GT-KOALA/FFOLayer!

References I

[1] Zihao Zhao, Kai-Chia Mo, Shing-Hei Ho, Brandon Amos, and Kai Wang.
A Fully First-Order Layer for Differentiable Optimization.
Proceedings of the 43rd International Conference on Machine Learning, 2026.

[2] Jeongyeol Kwon, Dohyun Kwon, Stephen Wright, Robert D Nowak.
A Fully First-Order Method for Stochastic Bilevel Optimization.
Proceedings of the 40th International Conference on Machine Learning, 2023.

[3] Guy Kornowski, Vivek Padmanabhan, Kai Wang, Gaurav Zhang, and Suvrit Sra.
First-Order Methods for Linearly Constrained Bilevel Optimization.
Advances in Neural Information Processing Systems, 2024.

[4] Ashok Cutkosky, Harsh Mehta, Francesco Orabona
Optimal Stochastic Non-smooth Non-convex Optimization through Online-to-Non-convex Conversion.
Proceedings of the 40th International Conference on Machine Learning, 2023.