Bilevel Optimization: Theory, Algorithms, and Applications in Machine Learning and Foundation Models

Mingyi Hong

Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis

DeepLearn 2026 Tutorial
(3 sessions $\times$ 1.5 hours)

Tutorial Roadmap: Three Sessions

Session 1: Foundations

What and why bilevel; Stackelberg games; a gallery of ML applications; the implicit gradient (hypergradient) and the implicit function theorem.

Session 2: Theory & Algorithms

Reformulations (KKT, value-function, penalty) and the three hypergradient approaches; deterministic, stochastic, and first-order algorithms; a frontier beyond strong convexity.

Session 3: Applications

Classical ML (meta-learning, adversarial training, pruning, dataset selection, IRM); foundation models (RLHF/alignment, inverse RL); frontiers.

Session 1: Outline

1. Define & situate. What is bilevel optimization (leader–follower)? Its Stackelberg origin and history.
2. Motivate. Bilevel is *everywhere* in ML: motivating applications *with results*, and a gallery of formulations (HPO, meta-learning, NAS, data reweighting, IRM).
3. Compute. A toy bilevel (by hand and in PyTorch); min–max / hierarchy and optimality as special cases; the implicit gradient (hypergradient) via the implicit function theorem.

Roadmap

Three beats: *define* $\rightarrow$ *motivate* $\rightarrow$ *compute the implicit gradient.* (Session 2: theory & algorithms.)

What is Bilevel Optimization?

A bilevel program nests one optimization inside another:

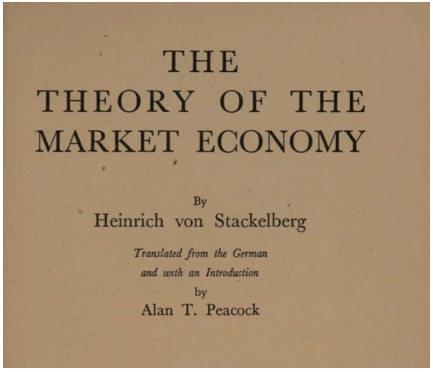
$$\min_{x,\,y} \underbrace{f(x,y)}_{\text{upper level}} \qquad \text{s.t.} \qquad y \in \underbrace{\arg\min_{y' \in \mathcal{C}} g(x,y')}_{\text{lower level}}, \quad x \in \mathcal{U}.$$

- Upper level: the variable x (leader's decision) and objective f.
- Lower level: the variable y (follower's response) and objective g.
- Benefit: models a decision *hierarchy* across two coupled objectives.
- Difficulty: the coupling through $y^*(x)$ makes optimization hard.

Stackelberg Games: The Origin of Bilevel

Bilevel is rooted in a Stackelberg game [von Stackelberg, 1934, 1952]: a *sequential* game with commitment:

- Leader (upper level) *moves first* and commits to x.
- Follower (lower level) *observes* x, then best-responds $y^*(x) \in \arg\min_y g(x, y)$.
- The leader optimizes *anticipating* the reaction, i.e., *through* the follower's response curve $y^*(\cdot)$.
- Origin: von Stackelberg's 1934 *duopoly*, a market leader that commits before rivals.
- Unlike Nash (simultaneous play), *order* and *commitment* matter: a first-mover structure.
- Not a single shared objective (Obj 1 $\neq$ Obj 2); the bilevel solution *is* the Stackelberg equilibrium.



THE
THEORY OF THE
MARKET ECONOMY

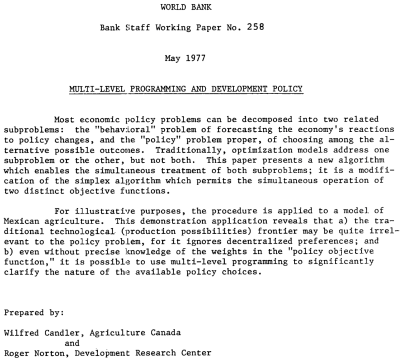
By
Heinrich von Stackelberg

Translated from the German
and with an Introduction
by
Alan T. Peacock

Leader commits, follower reacts; leader optimizes through $y^*(\cdot)$.

Bilevel: A Problem with a Long History

- 1934 / 1952: H. von Stackelberg formalizes the leader–follower game in economics [von Stackelberg, 1934, 1952].
- 1973: Bracken & McGill give the modern optimization form, "programs with optimization problems in the constraints," for *military resource allocation* [Bracken and McGill, 1973].
- 1977: the terms *"bilevel" / "multilevel" programming* are coined in a World Bank report [Candler and Norton, 1977].
- 1980s →: transportation, economics, power systems, signal processing, and now machine learning.



WORLD BANK

Bank Staff Working Paper No. 258

May 1977

MULTI-LEVEL PROGRAMMING AND DEVELOPMENT POLICY

Most economic policy problems can be decomposed into two related subproblems: the "behavioral" problem of forecasting the economy's reactions to policy changes, and the "policy" problem proper, of choosing among the alternative possible outcomes. Traditionally, optimization models address one subproblem or the other, but not both. This paper presents a new algorithm which enables the simultaneous treatment of both subproblems; it is a modification of the simplex algorithm which permits the simultaneous operation of two distinct objective functions.

For illustrative purposes, the procedure is applied to a model of Mexican agriculture. This demonstration application reveals that a) the traditional technological (production possibilities) frontier may be quite irrelevant to the policy problem, for it ignores decentralized preferences; and b) even without precise knowledge of the weights in the "policy objective function," it is possible to use multi-level programming to significantly clarify the nature of the available policy choices.

Prepared by:

Wilfred Candler, Agriculture Canada
and
Roger Norton, Development Research Center

Nested multilevel structure.

Checkpoint: We Have Defined Bilevel. Now, Why Care?

✓ What we've covered

- Bilevel = a leader–follower (Stackelberg) program: the upper level optimizes *through* the follower's response $y^*(x)$.
- Its game-theoretic origin & history: Stackelberg (1934) → Bracken–McGill (1973) → today.

⇒ Coming up next

- Why bilevel is everywhere in ML: motivating applications shown *with results*.
- A gallery of formulations: HPO, meta-learning, NAS, data reweighting, IRM.

Why Should We Care? Bilevel Is Already Everywhere

Recent AI/ML papers built directly on a bilevel formulation:

- Fast adversarial training [ICML'22]
- Advancing model pruning [NeurIPS'22]
- DARTS: architecture search [ICLR'19]
- Meta-learning w/ implicit grad. [NeurIPS'19]
- Probabilistic coreset selection [ICML'22]
- Clean-label data poisoning [NeurIPS'20]
- What is missing in IRM [ICLR'23]
- RLHF / alignment of LLMs [NeurIPS'22+]

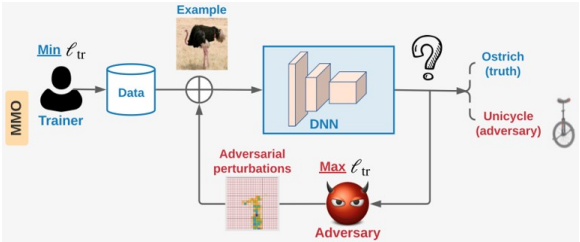
Survey: [Zhang et al., 2024] (arXiv:2308.00788) | classical: [Colson et al., 2007].

Motivating App 1: Adversarial Training (Robust AI)

- A tiny perturbation flips "ostrich" $\rightarrow$ "unicycle."
- Min–max defense (Madry'17): trainer vs. adversary *share* one loss ℓ_{tr},

$$\min_{\theta} \max_{\|\delta\| \le \epsilon} \ell_{\text{tr}}(\theta, \delta).$$

- Limitation: trainer and attacker forced to use the *same* objective.



Adversarial example + min–max view of AT.

Adversarial Training: Min–Max → Bilevel

Bilevel view [Zhang et al., 2022b]: let the attacker have its *own* objective ℓ_{atk},

$$\min_{\theta} \ \mathbb{E}_x\big[\ell_{\text{tr}}(\theta, \delta^*(\theta))\big] \quad \text{s.t.} \quad \delta^*(\theta) = \arg\min_{\delta \in \mathcal{C}} \ \ell_{\text{atk}}(\theta, \delta).$$

- A *minor* modeling change (MMO → BLO) significantly changes the optimization foundations.
- Min–max is the *special case* $\ell_{\text{atk}} = -\ell_{\text{tr}}$.
- Enables attack-agnostic, scalable robust training (Fast-BAT, Session 2).

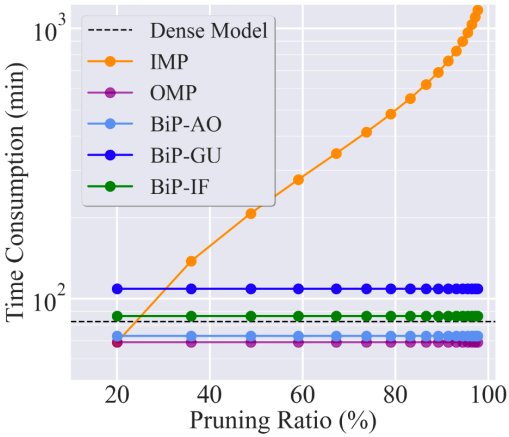
Motivating App 2: Model Pruning (Efficient AI)

Prune weights but keep accuracy. Bilevel formulation [Zhang et al., 2022a]:

$$\min_{m} \ \ell_{\text{prune}}\big(m \odot \theta^*(m)\big)$$

$$\text{s.t.} \ \ \theta^*(m) = \arg\min_{\theta} \ell_{\text{tr}}(m \odot \theta) + \tfrac{\gamma}{2}\|\theta\|^2.$$

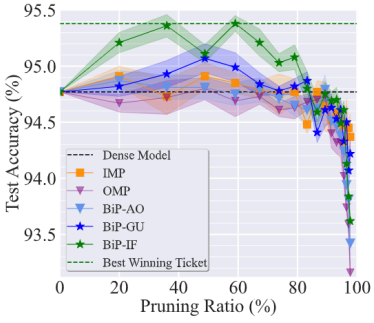
- Upper: pruning mask m. Lower: mask-fixed retraining.
- SOTA before: iterative magnitude pruning (IMP), accurate but costly (re-train T times).



IMP cost explodes with sparsity; bilevel pruning (BiP) is sparsity-agnostic.

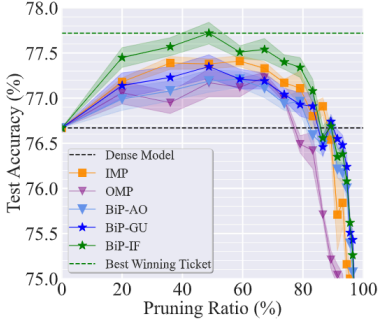
Initial Result: Bilevel Pruning (BiP) Engages

- BiP-IF matches/beats the best "winning ticket" across sparsities.
- Accounting for the coupling (implicit gradient) *matters*: BiP-IF/GU > BiP-AO.
- And it stays cheap (previous slide).



(a) CIFAR-10

CIFAR-10



(b) CIFAR-100

CIFAR-100

Takeaway

Takeaway: a clean bilevel formulation + implicit gradient = better *and* faster.

A Holistic View of Bilevel in AI/ML

Robust / Generalizable AI

- Adversarial training, attack generation
- Invariant risk minimization (IRM)

Automated AI

- Hyperparameter optimization
- Neural architecture search (DARTS)

Efficient AI

- Model pruning, dataset condensation
- Coreset / data selection

Meta / Data

- Model-agnostic meta-learning (MAML)
- Data poisoning, reweighting

One template, four "frontiers" of trustworthy/efficient AI.

Example Gallery I: HPO, Meta-Learning, NAS

Hyperparameter optimization [Franceschi et al., 2018]

$\min_\lambda \mathcal{L}_{\text{val}}(w^*(\lambda))$ s.t. $w^*(\lambda) = \arg\min_w \mathcal{L}_{\text{tr}}(w, \lambda)$ (val picks λ, train fits w)

Meta-learning / MAML [Finn et al., 2017]

$\min_x \sum_\tau \mathcal{L}_\tau^{\text{val}}(y_\tau^*(x))$ s.t. $y_\tau^*(x) = \arg\min_y \mathcal{L}_\tau^{\text{tr}}(y, x)$ (shared init x)

Neural architecture search / DARTS [Liu et al., 2019]

$\min_\alpha \mathcal{L}_{\text{val}}(\alpha, w^*(\alpha))$ s.t. $w^*(\alpha) = \arg\min_w \mathcal{L}_{\text{tr}}(\alpha, w)$ (architecture α)

Common shape

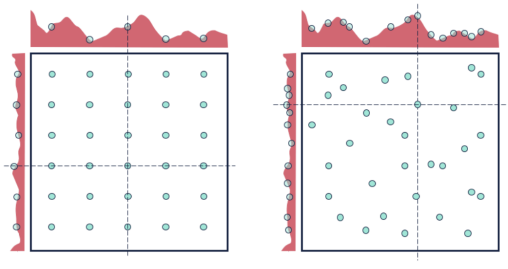
All share one shape: *validation* chooses structure, *training* fits weights.

Application 1: Hyperparameter Optimization (HPO)

Hyperparameters λ (regularizers, learning rate, data weights) are judged on validation data; the weights w are fit on training data:

$$\min_{\lambda} \; \mathcal{L}_{\text{val}}\big(w^*(\lambda)\big) \quad \text{s.t.} \quad w^*(\lambda) = \arg\min_{w} \; \mathcal{L}_{\text{tr}}(w, \lambda).$$

- Grid/random search scales *exponentially* in $\#\lambda$: hopeless beyond a few knobs.
- Bilevel view: treat λ as an upper-level variable and follow the *hypergradient* $\nabla_\lambda \mathcal{L}_{\text{val}}$.



Validation loss over the hyperparameter landscape.

HPO: From a Few Knobs to Millions

Few hyperparameters

Regularization weights, learning rate, momentum: classical tuning; search still (barely) works.

Many hyperparameters [Maclaurin et al., 2015, Franceschi et al., 2018]

One weight *per training example* (reliability), per-layer ℓ_2, augmentation strengths: *millions* of λ.

- The hypergradient $\nabla_\lambda \mathcal{L}_{\text{val}} = -\nabla^2_{\lambda w}\mathcal{L}_{\text{tr}}\,[\nabla^2_{ww}\mathcal{L}_{\text{tr}}]^{-1}\nabla_w \mathcal{L}_{\text{val}}$ is *exactly* the implicit gradient, computed by unrolling or IFT (Session 2).
- Gradient-based HPO scales where search cannot: data reweighting/cleaning, learned augmentation, per-parameter regularization.

Why bilevel

Why bilevel: turns intractable search into a differentiable optimization.

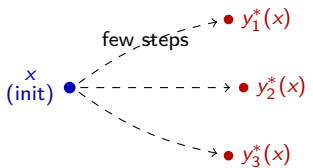
Application 2: Meta-Learning (MAML)

Learning to learn: adapt to a *new* task from only a few examples (few-shot).

- MAML [Finn et al., 2017] learns a shared initialization x from which a few gradient steps solve any new task.
- Bilevel formulation over tasks τ:

$$\min_{x} \sum_{\tau} \mathcal{L}_{\tau}^{\text{val}}\big(y_{\tau}^{*}(x)\big) \quad \text{s.t.} \quad y_{\tau}^{*}(x) = \arg\min_{y} \; \mathcal{L}_{\tau}^{\text{tr}}(y, x).$$

- Upper: the initialization (shared meta-knowledge). Lower: per-task fine-tuning from x.



One initialization x; a few gradient steps *adapt* it to each task's solution $y_{\tau}^{*}(x)$. Meta-learning tunes x so *every* adaptation succeeds.

Meta-Learning: Why Bilevel

- The initialization x is judged by its *post-adaptation* validation loss, so the hypergradient must flow through the fine-tuning steps $y_\tau^*(x)$.
- This is exactly the implicit gradient $\mathrm{d}y_\tau^*/\mathrm{d}x$: it measures how the initialization x shapes each task's solution.

Solvers are a choice (Session 2)

FO-MAML, full unrolling (MAML), sign-based (Sign-MAML), and implicit function (iMAML) all solve the *same* bilevel problem.

Uses: few-shot vision, fast adaptation, personalization, robotics.

Big picture

Same template: the tasks play the role of "data."

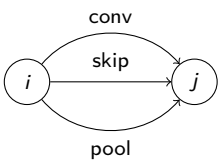
Application 3: Neural Architecture Search (NAS)

Automate network design (which operations, layers, and connections) instead of hand-crafting it [Liu et al., 2019].

- Upper: choose an architecture α (judged on validation).
- Lower: train weights w for that architecture (on training data).

$$\min_{\alpha} \ \mathcal{L}_{\text{val}}\big(w^*(\alpha)\big) \ \text{ s.t. } \ w^*(\alpha) = \arg\min_{w} \ \mathcal{L}_{\text{tr}}(w, \alpha).$$

- Difficulty: the architecture space is *discrete* and astronomically large: naive search costs thousands of GPU-days.



Candidate operations on one edge of a cell.

DARTS: Differentiable Architecture Search [Liu et al., 2019]

Key idea: relax the discrete architecture choice into a continuous mixture, so the architecture parameters α become differentiable. Each edge of a cell mixes candidate operations by a softmax:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})}\, o(x).$$

With continuous α, architecture search becomes a smooth bilevel problem:

$$\min_{\alpha} \; \mathcal{L}_{\text{val}}\big(w^*(\alpha)\big) \quad \text{s.t.} \quad w^*(\alpha) = \arg\min_{w} \; \mathcal{L}_{\text{tr}}(w, \alpha).$$

- Upper: architecture weights α (validation). Lower: network weights w (training).
- The hypergradient $\nabla_\alpha \mathcal{L}_{\text{val}}$ comes from one-step unrolling (Session 2); after search, discretize by keeping the strongest operation $\arg\max_o \alpha_o$.
- Payoff: architecture search from thousands of GPU-days to a few.

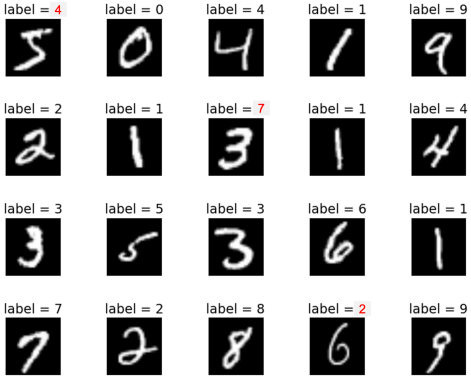
Application 4: Learning to Reweight Data

Real training sets are noisy: mislabeled, corrupted, imbalanced, even poisoned. Plain ERM trusts *every* sample equally.

- Give each example i a weight $\lambda_i \geq 0$; a small *trusted* validation set decides which samples to trust.
- Bilevel formulation [Ren et al., 2018]:

$$\min_{\lambda \geq 0} \mathcal{L}_{\text{val}}\big(w^*(\lambda)\big) \quad \text{s.t.} \quad w^*(\lambda) = \arg\min_{w} \sum_i \lambda_i \, \ell_i(w).$$

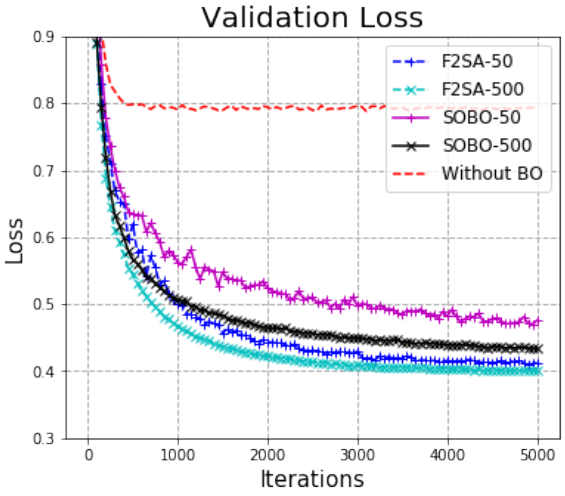
- Lower: fit the model on *weighted* data.
  Upper: choose weights that generalize.



Red labels are wrong. Reweighting learns to *down-weight* the corrupted examples, without being told which they are.

Data Reweighting: One Template, Many Tasks

- The validation set *supervises* the weights: the implicit gradient $\nabla_{\lambda_i} \mathcal{L}_{\text{val}}$ tells us whether up-weighting example i helps or hurts generalization.
- Down-weight noisy/poisoned; up-weight informative, hard examples.
- One template, many names:
  - coreset / data selection ($\lambda_i \in \{0, 1\}$, keep k),
  - dataset distillation & condensation,
  - (adversarially) data poisoning.



30% corrupted labels: bilevel reweighting (F^2SA/SOBO)

Application 5: Invariant Risk Minimization (IRM)

The problem: models exploit spurious correlations that do not transfer.

- Colored-MNIST: in training, *color* happens to predict the digit; at test time the color–label link flips.
- Plain ERM latches onto color (easy but spurious) and fails at test.
- Data arrive in several environments $e = 1, \ldots, E$, each with a *different* spurious correlation.
- Goal: a representation that is invariant across environments, forced to use the *causal* feature (shape), not color.

In training, color tracks the label; at test the colors are swapped. A color-reliant (ERM) model breaks; a *shape*-reliant model survives.

IRM: The Invariance Principle and IRMv1

Invariance principle [Arjovsky et al., 2019]

Learn a representation θ so that *one* predictor ϕ is simultaneously optimal in *every* environment. Then $\phi \circ \theta$ can only rely on features that are stable across environments (the causal ones).

First attempt: IRMv1 [Arjovsky et al., 2019]. Enforce the principle with a gradient-norm penalty, using a fixed scalar "dummy" classifier $w = 1.0$ on top of θ:

$$\min_{\theta} \sum_{e=1}^{E} R_e(\theta) \; + \; \lambda \left\| \nabla_{w|w=1.0} R_e(w \cdot \theta) \right\|^2,$$

where R_e is the risk in environment e; the penalty asks $w = 1.0$ to be *optimal in every environment at once*.

- Limitation: this single-level penalty is only a loose surrogate, and can fail in multi-environment training [Zhang et al., 2024].

IRM as a Consensus-Constrained Bilevel

Bilevel reformulation [Zhang et al., 2024]: give each environment its own predictor ϕ_i, tied by a consensus constraint $\phi_1 = \cdots = \phi_E$:

$$\min_{\theta} \sum_{i=1}^{E} \ell_i(\phi_i^* \circ \theta) \quad \text{s.t.} \quad \{\phi_i^*(\theta)\} = \arg\min_{\phi_1=\cdots=\phi_E} \sum_{i=1}^{E} \ell_i(\phi_i \circ \theta).$$

- Upper: the shared representation θ. Lower: per-environment predictors forced to a consensus.
- *Why bilevel:* it splits a non-separable problem into per-environment fits plus a consensus that enforces invariance.
- Avoids IRMv1's loose penalty; solved by gradient unrolling (Session 2).

IRM: Bilevel Closes the Gap

- ERM: high accuracy gap across environments: it is *not* invariant.
- All IRM variants beat ERM; IRM-BLO (consensus-constrained bilevel) is best on *both* metrics.
- Turns a non-separable problem into decomposable per-environment fits + consensus.

Method	BLO (Solver)	Colored-MNIST		Colored-FashionMNIST	
		Avg. Acc.	Acc. Gap	Avg. Acc.	Acc. Gap
ERM	N/A	49.19±1.89	90.72±2.08	49.77±1.71	88.62±2.49
IRM-v1	N/A	68.33±0.31	2.04±0.05	68.76±0.31	1.45±0.09
IRM-Game	N/A	67.73±0.24	1.67±0.14	67.49±0.32	1.82±0.13
IRM-BLO	GU	**69.47**±0.24	**1.04**±0.07	**69.43**±0.21	**1.14**±0.11

Colored-MNIST / Colored-FashionMNIST: avg. acc. (higher better) & acc. gap (lower better).

Stackelberg Application: Security Games

A defender commits to a (randomized) patrol; an attacker observes it and strikes the least-protected target.

$$\max_{x \in \mathcal{X}} \; U_d\big(x, a^*(x)\big) \quad \text{s.t.} \quad a^*(x) \in \arg\max_{a \in \mathcal{A}} \; U_a(x, a).$$

- Upper (leader): the defender's coverage x over targets.
- Lower (follower): the attacker's best response $a^*(x)$ to that coverage.
- Real deployments: airport patrols (ARMOR, LAX), federal air marshals, coast-guard patrols, and anti-poaching (PAWS).

Why bilevel

Commit first, then anticipate the attacker's response: exactly the leader–follower structure.

Stackelberg Application: Pricing & Toll Setting

A leader sets prices or tolls; many self-interested users then settle into an equilibrium in response.

$$\max_{\tau \geq 0} \; R\big(\tau, f^*(\tau)\big) \quad \text{s.t.} \quad f^*(\tau) = \text{user equilibrium under tolls } \tau.$$

- Upper (leader): tolls / prices τ, set to maximize revenue or reduce congestion.
- Lower (followers): users' selfish routing or consumption (a Wardrop / Nash equilibrium).
- Uses: congestion pricing, electricity and cloud pricing, revenue management.

Why bilevel

The leader optimizes over an *equilibrium* of many followers, a classic bilevel (MPEC) problem.

Checkpoint: Motivated. Now Make It Concrete

✓ **What we've covered**

- Bilevel is pervasive: robust, efficient, automated, and generalizable AI.
- Five ML formulations, one template: the upper level *shapes* the lower-level problem.

⇒ **Coming up next**

- A toy bilevel solved by hand *and* in PyTorch.
- Special cases (min–max, hierarchy) and optimality concepts.
- The three pillars and the implicit gradient (hypergradient) via the implicit function theorem.

Working Example: A Toy Bilevel We Can Solve by Hand

$$\min_{x\in\mathbb{R}} \; F(x) = f(x, y^*(x)), \quad f(x,y) = \tfrac{1}{2}(x-3)^2 + \tfrac{1}{2}(y-1)^2,$$

$$y^*(x) = \arg\min_{y} \; g(x,y), \qquad g(x,y) = \tfrac{1}{2}(y-x)^2.$$

- Lower level: $\nabla_y g = y - x = 0 \Rightarrow y^*(x) = x$ (closed form here, rarely so in practice).
- Substitute: $F(x) = \tfrac{1}{2}(x-3)^2 + \tfrac{1}{2}(x-1)^2$, so $\nabla F(x) = (x-3) + (x-1) = 2x - 4$.
- Optimum: $x^\star = 2$, $y^\star = 2$. Naive alternating min would instead drive $y \to x$ then $x \to 3$, converging to the *wrong* point $x = 3$ if it ignores $\frac{dy^*}{dx}$.

Punchline

This $\frac{dy^*}{dx} = 1$ term is exactly the implicit gradient we develop next.

Working Example: The Same Problem in PyTorch

- **Working example:** differentiate through the (solved) lower level via autograd.

```
import torch
x = torch.tensor(3.0, requires_grad=True) # upper-level variable

def lower_solution(x): # y*(x) = argmin_y 0.5*(y-x)^2
    y = x.detach().clone().requires_grad_(True)
    for _ in range(200): # solve LL by gradient descent
        g = 0.5*(y - x)**2
        gy, = torch.autograd.grad(g, y, create_graph=True)
        y = y - 0.5*gy # keep graph -> y* depends on x
    return y

for _ in range(100): # upper-level gradient descent
    y_star = lower_solution(x)
    F = 0.5*(x-3)**2 + 0.5*(y_star-1)**2 # upper objective f(x, y*(x))
    (gx,) = torch.autograd.grad(F, x) # hypergradient dF/dx (incl. dy*/dx)
    with torch.no_grad(): x -= 0.3*gx
print(x.item()) # -> 2.0 (correct bilevel optimum)
```

Intuition

Min–Max and Hierarchical Optimization as Special Cases

Min–max = bilevel with adversarial lower level

$$\min_x \max_y f(x,y) \;\equiv\; \min_x f(x, y^*(x)), \quad y^*(x) \in \arg\min_y \{-f(x,y)\}.$$

A shared, fully *opposed* objective; bilevel allows the levels to differ.

Multi-level / hierarchical

Nest more than two levels (leader $\rightarrow$ middle $\rightarrow$ follower); RL actor–critic and hierarchical planning are instances.

Key point

Bilevel is the strictly more expressive model: decoupled upper/lower goals.

Optimality Concepts: The Optimistic Case

When the lower solution set $S(x) = \arg\min_y g(x, y)$ is *not a singleton*, which $y^*(x)$ does the leader assume?

Optimistic (cooperative)

$$\min_x \min_{y \in S(x)} f(x, y)$$

Ties among lower-level optima are broken *for* the leader.

- When: we *control* the follower, so it can return the optimum most favorable to us.
- Examples: hyperparameter optimization and meta-learning (the inner solver is ours); a cooperative operator who can steer which equilibrium is reached.

Optimality Concepts: The Pessimistic Case

Pessimistic (adversarial)

$$\min_{x} \max_{y \in S(x)} f(x, y)$$

Ties among lower-level optima are broken *against* the leader.

- When: we need *worst-case* guarantees; the follower may respond adversarially.
- Examples: security games (the attacker picks the worst response for the defender), adversarial training, and network interdiction.

Good news

Most ML assumes a unique $y^*(x)$ (lower-level strong convexity), so the two cases *coincide*. Existence and continuity of $y^*(\cdot)$ is *well-posedness*.

The Simplest Bilevel Problem

$$\min_x \; f(x, y^*(x)), \qquad y^*(x) = \arg\min_y \; g(x, y),$$

with f strongly convex in x and g strongly convex in y (no constraints).

- Tempting: alternating minimization (fix x, solve y; fix y, solve x).
- But the upper level must account for how y^* *moves with* x: as the toy example showed, ignoring it gives the wrong answer.

Key point

The correction term is the implicit gradient.

The Implicit Gradient (Hypergradient): A Fingerprint

Let $F(x) := f(x, y^*(x))$. By the chain rule,

$$\nabla F(x) = \nabla_x f(x, y^*) + \left[\nabla y^*(x)\right]^\top \nabla_y f(x, y^*),$$

the implicit gradient (IG) carrying signal from the lower level back to the upper level.

- The IG is the *fingerprint* of every bilevel algorithm.
- Need $\nabla y^*(x)$, but $y^*(x)$ generally has *no closed form*.

The Implicit Function Theorem

How can we differentiate $y^*(x)$ when we cannot even write it down?

Implicit function theorem (general form)

Let $\Phi(x, y) = 0$ with Φ continuously differentiable, and suppose the y-Jacobian $\partial_y \Phi$ is invertible at a solution (x_0, y_0). Then near x_0 the equation defines y as a unique, differentiable function $y(x)$, with

$$\frac{\partial y}{\partial x} = -\left[\partial_y \Phi\right]^{-1} \partial_x \Phi.$$

- Intuition: if the equation is non-degenerate in y (invertible y-Jacobian), its solution set is locally a *smooth graph* $y = y(x)$: we can solve for y, and it moves smoothly as x varies.
- No closed form for $y(x)$ is needed; the theorem still hands us its derivative.

Deriving $\nabla y^*(x)$ via the Implicit Function Theorem

Apply the theorem to the lower-level stationarity condition $\Phi(x, y) := \nabla_y g(x, y) = 0$. Its y-Jacobian is the Hessian $\nabla^2_{yy} g$, which is invertible by lower-level strong convexity, so $y^*(x)$ is differentiable. Differentiating $\nabla_y g(x, y^*(x)) = 0$ in x:

$$\nabla^2_{yx} g + \nabla^2_{yy} g \, \nabla y^*(x) = 0 \quad \Longrightarrow \quad \nabla y^*(x) = -\left[\nabla^2_{yy} g\right]^{-1} \nabla^2_{yx} g.$$

Hence the hypergradient

$$\nabla F(x) = \nabla_x f - \nabla^2_{xy} g \boxed{\left[\nabla^2_{yy} g\right]^{-1} \nabla_y f}.$$

Bottom line

The boxed inverse-Hessian–vector product is the computational crux.

Hypergradient Theorem (Clean Statement & Proof)

Theorem (Implicit gradient)

Suppose $g(x,\cdot)$ is twice differentiable and $\nabla^2_{yy} g(x, y^*(x)) \succ 0$ (invertible) at $y^*(x) = \arg\min_y g(x,y)$, and f is differentiable. Then $F(x) = f(x, y^*(x))$ is differentiable with

$$\nabla F(x) = \nabla_x f - \nabla^2_{xy} g \left[\nabla^2_{yy} g\right]^{-1} \nabla_y f.$$

Proof (sketch).

Optimality gives $\nabla_y g(x, y^*(x)) = 0$ for all x. Since $\nabla^2_{yy} g$ is invertible, the implicit function theorem makes $y^*(\cdot)$ C^1 with $\nabla y^* = -[\nabla^2_{yy} g]^{-1} \nabla^2_{yx} g$. Substitute into the chain rule $\nabla F = \nabla_x f + [\nabla y^*]^\top \nabla_y f$. □

Working Example: Hypergradient by Implicit Differentiation

- **Working example:** form the IG *without* unrolling: solve one linear system instead.

```
import torch
# inverse-Hessian-vector product v = [d^2_yy g]^{-1} (d_y f) via conjugate grad.
def hypergrad(f, g, x, y_star):
    dyf, = torch.autograd.grad(f, y_star, retain_graph=True)
    dyg, = torch.autograd.grad(g, y_star, create_graph=True) # for HVP
    def Hvp(v): # Hessian-vector product
        return torch.autograd.grad(dyg, y_star, grad_outputs=v, retain_graph=True)[0]
    v = conjugate_gradient(Hvp, dyf) # solve H v = dyf (no explicit inverse)
    dxf, = torch.autograd.grad(f, x, retain_graph=True)
    cross = torch.autograd.grad(dyg, x, grad_outputs=v)[0] # d^2_xy g . v
    return dxf - cross # = dF/dx
```

In practice

Only Hessian-vector products (one backward pass each): never form or invert H (Session 2).

Session 1 Recap → Session 2 Preview

✓ What we've covered

- **What bilevel is:** a leader–follower (Stackelberg) program, with a long history.
- **Where it appears:** a gallery of ML formulations (HPO, meta-learning, NAS, data reweighting, IRM), each shown with results.
- **The implicit gradient** (hypergradient): derived via the implicit function theorem, computed through an inverse-Hessian–vector product.

⇒ Coming up next

- **Session 2 — Theory:** re-stating the problem, then reformulations, KKT conditions, and value-function / penalty approaches.
- The three ways to compute hypergradients (implicit function, unrolling, value function), problem classes and assumptions, and the resulting algorithms.

Session 2: Outline

1. Theory & reformulations. The bilevel problem restated; three ways to compute the hypergradient (implicit function, gradient unrolling, value function); KKT / variational-inequality reformulations; penalty methods; problem classes, assumptions, and why bilevel is hard.
2. Algorithms. Deterministic and stochastic implicit-function methods (BA, AID-BiO, TTSA, ...); making the inverse Hessian cheap; inexact hypergradients; fully first-order methods; and a frontier beyond lower-level strong convexity.

Roadmap

Two parts: *theory* $\rightarrow$ *algorithms*. (Applications are Session 3.)

The Bilevel Problem

We study the bilevel program

$$\min_{x\in\mathbb{R}^{d_x}} F(x) := f\big(x, y^*(x)\big) \qquad \text{s.t.} \qquad y^*(x) \in \arg\min_{y\in\mathbb{R}^{d_y}} g(x,y).$$

- x: upper-level (leader) variable; f: upper objective.
- y: lower-level (follower) variable; g: lower objective.
- $y^*(x)$: lower-level solution map; $F(x) := f(x, y^*(x))$: the hyper-objective.

- Goal: descend F using the hypergradient $\nabla F(x)$.
- Recall (Session 1): $\nabla F = \nabla_x f - \nabla^2_{xy} g\,[\nabla^2_{yy} g]^{-1} \nabla_y f$.
- Every method below either *computes* or *avoids* this hypergradient.

Notation

Upper x / lower y; solution map $y^*(x)$; hyper-objective $F(x) = f(x, y^*(x))$.

Three Ways to Compute the Hypergradient

The coupling $y^*(x)$ has no closed form. There are three main approaches:

1. Implicit function (IF)

Differentiate the lower-level optimality condition $\nabla_y g(x, y^*) = 0$. Exact, but needs an *inverse-Hessian–vector product* (Session 1; lower-level constraints via KKT next).

2. Gradient unrolling (GU)

Backpropagate through the lower-level optimization *trajectory*. No inverse Hessian, but the cost grows with the number of unrolled steps.

3. Value function (VF) & penalty

Replace the lower level by a value-function constraint, giving a *single-level* problem. Handles constrained and non-unique lower levels.

Recall: The Unconstrained Implicit Function Approach

With no lower-level constraints, the follower's optimum satisfies stationarity,

$$\nabla_y g\big(x, y^*(x)\big) = 0.$$

Differentiating in x (implicit function theorem) gives the implicit gradient, hence the hypergradient:

$$\frac{\mathrm{d}y^*}{\mathrm{d}x} = -\big[\nabla^2_{yy} g\big]^{-1} \nabla^2_{yx} g, \qquad \nabla F = \nabla_x f - \nabla^2_{xy} g \big[\nabla^2_{yy} g\big]^{-1} \nabla_y f.$$

- Needs only the inverse Hessian $[\nabla^2_{yy} g]^{-1}$, invertible by lower-level strong convexity.
- Question: what if the lower level is constrained, so $\nabla_y g \neq 0$ at the optimum?

Constrained Lower Level: KKT Replaces Stationarity

Now the follower is constrained:

$$y^*(x) = \arg\min_y \; g(x, y) \quad \text{s.t.} \quad Ay \leq b.$$

- At a boundary optimum, $\nabla_y g \neq 0$: the gradient is balanced by constraint forces.
- Its KKT conditions (Lagrangian $\mathcal{L} = g(x, y) + \lambda^\top (Ay - b)$):

$$\underbrace{\nabla_y g(x, y^*) + A^\top \lambda^* = 0}_{\text{stationarity}}, \quad \underbrace{Ay^* \leq b, \quad \lambda^* \geq 0}_{\text{feasibility}}, \quad \underbrace{\lambda_i^* \, (Ay^* - b)_i = 0}_{\text{complementary slackness}} .$$

Intuition

Complementary slackness: each constraint is either inactive ($\lambda_i^* = 0$) or active ($(Ay^*)_i = b_i$).

When Is $y^*(x)$ Differentiable? Two Conditions

To differentiate through the KKT system, it must behave like the unconstrained $\nabla_y g = 0$. Two conditions play the role of the "invertible Hessian":

1. Constraint qualification (LICQ)

The active constraint rows A_0 are *linearly independent*.

$\Rightarrow$ the multipliers λ^* are unique, and the linearized system is solvable.

2. Strict complementarity

Every active constraint has a *strictly positive* multiplier ($\lambda_i^* > 0$, not 0).

$\Rightarrow$ the active set is stable: small changes in x do not switch constraints on/off, so $y^*(x)$ is smooth.

Otherwise

If a constraint is *weakly* active ($\lambda_i^* = 0$ while active), the active set can change and $y^*(x)$ is only *directionally* differentiable.

Differentiating the KKT System

Keep the active constraints $A_0 y^* = b_0$ (those with $\lambda_i^* > 0$). Differentiating stationarity and activity in x gives a linear system:

$$\begin{bmatrix} \nabla^2_{yy} g & A_0^\top \\ A_0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{d}y^*/\mathrm{d}x \\ \mathrm{d}\lambda_0^*/\mathrm{d}x \end{bmatrix} = - \begin{bmatrix} \nabla^2_{yx} g \\ 0 \end{bmatrix}.$$

- A saddle-point (KKT) linear system; solving it gives $\frac{\mathrm{d}y^*}{\mathrm{d}x}$ via a Schur complement of $\nabla^2_{yy} g$ restricted to the active set.
- Plug into the chain rule: $\nabla F = \nabla_x f + \left(\frac{\mathrm{d}y^*}{\mathrm{d}x}\right)^\top \nabla_y f$.

Same recipe

Constrained IF is still "differentiate the optimality conditions," now the KKT system rather than $\nabla_y g = 0$.

Special Case: Box Constraints

For box constraints $\ell \leq y \leq u$ (as in adversarial attacks), the active set is just the clamped coordinates:

- Active (clamped): the coordinate sits *exactly at a box boundary*, $y_i^* = \ell_i$ or $y_i^* = u_i$. A small change in x does not push it off that boundary, so it stays *fixed* ("pinned"): hence $\frac{dy_i^*}{dx} = 0$.
- Inactive (interior): $\ell_i < y_i^* < u_i$, the constraint is *slack*, so the coordinate moves freely with x, exactly as in the unconstrained case.

Result

$\frac{dy^*}{dx}$ is a coordinate mask: zero on clamped coordinates, the ordinary (unconstrained) sensitivity on interior ones. No linear solve needed.

Box Constraints: A Concrete Example

Suppose $y^* \in \mathbb{R}^3$ with $y^* = (\, u_1,\ y_2^*,\ \ell_3\,)$: coordinates 1 and 3 are clamped at their bounds, coordinate 2 is interior. Then

$$\frac{\mathrm{d}y^*}{\mathrm{d}x} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{keep interior only}} \left.\frac{\mathrm{d}y^*}{\mathrm{d}x}\right|_{\text{unconstrained}}.$$

- Only the interior coordinate y_2^* passes sensitivity to x; the clamped coordinates contribute *nothing*.
- So the hypergradient is: mask out the clamped coordinates, then apply the usual chain rule on the interior block.

Why it matters

This mask is what makes Fast-BAT (adversarial training) cheap: a closed-form constrained hypergradient. (Application in Part C.)

Gradient Unrolling: Differentiate the LL Trajectory

Question: can we get $\nabla y^*(x)$ directly by the chain rule along the optimization *path*, avoiding the inverse Hessian?

- Run K LL steps $y_0 \to y_1 \to \cdots \to y_K \approx y^*(x)$ (with y_0 independent of x).
- Backprop through the path: with $y_{k+1} = y_k - \beta \nabla_y g(x, y_k)$,

$$\frac{dy_K}{dx} = \sum_{i=1}^{K} \Big(\prod_{j=i+1}^{K} \frac{\partial y_j}{\partial y_{j-1}} \Big) \frac{\partial y_i}{\partial x}, \qquad \frac{\partial y_{k+1}}{\partial y_k} = I - \beta \nabla^2_{yy} g.$$

- No inverse needed; differentiable end-to-end.
- Cost: time *and* memory grow with K (store the whole path): a bias/cost trade-off.

Unrolling: Forward vs. Backward Mode

Write the per-step Jacobians of the lower-level step $y_{k+1} = y_k - \beta \nabla_y g(x, y_k)$:

$$B_k := \frac{\partial y_{k+1}}{\partial y_k} = I - \beta \nabla^2_{yy} g(x, y_k), \qquad A_k := \frac{\partial y_{k+1}}{\partial x} = -\beta \nabla^2_{yx} g(x, y_k).$$

Then $Z_k := \frac{\mathrm{d}y_k}{\mathrm{d}x}$ satisfies $Z_{k+1} = B_k Z_k + A_k$, and $\nabla F = \nabla_x f + Z_K^\top \nabla_y f$.

Forward mode [Franceschi et al., 2018]

Carry Z_k *forward* with the trajectory:
$Z_0 = 0,\ Z_{k+1} = B_k Z_k + A_k$.

Memory independent of K; cost scales with $\dim(x)$.

Best when x is low-dimensional.

Backward mode (reverse / BPTT)

Store $y_0, \ldots, y_K$, then backpropagate an adjoint: $\alpha_K = \nabla_y f,\ \alpha_k = B_k^\top \alpha_{k+1}$, accumulating ∇F.

Cost independent of $\dim(x)$; memory $O(K \dim(y))$.

Best when x is high-dimensional (the autodiff default).

Truncated and Sign-Based Unrolling

Truncated unrolling [Shaban et al., 2019]

Backpropagate through only the *last* τ steps (freeze B_k, A_k for $k < K - \tau$).

Memory drops to $O(\tau \dim(y))$; the hypergradient bias decays *geometrically* in τ (the lower-level map is a contraction). Trades memory for bias.

Sign-based unrolling (Sign-MAML)

Use sign-GD at the lower level: $y_{k+1} = y_k - \beta \operatorname{sign}\big(\nabla_y g(x, y_k)\big)$.

Since $\operatorname{sign}(\cdot)$ has *zero derivative* almost everywhere, $B_k = A_k = 0$, so $\frac{\mathrm{d}y_K}{\mathrm{d}x} = 0$ and ∇F collapses to the first-order term $\nabla_x f$.

Intuition

Sign-based unrolling is exactly why *Sign-MAML* costs like a first-order method: it drops the coupling.

Working Example: A K-Step Unrolled Inner Loop

- **Working example:** differentiable K-step LL solve: `autograd` returns the unrolled hypergradient.

```
import torch
# Upper var x; lower obj g(x,y); upper obj f(x,y). Unroll K LL GD steps.
def unrolled_hypergrad(x, g_fn, f_fn, y0, K=10, beta=0.1):
    y = y0.clone().requires_grad_(True)
    for _ in range(K): # differentiable LL trajectory
        g = g_fn(x, y)
        gy, = torch.autograd.grad(g, y, create_graph=True) # keep graph -> dy_K/dx
        y = y - beta * gy
    F = f_fn(x, y) # f(x, y_K(x)) ~ F(x)
    gx, = torch.autograd.grad(F, x) # chain rule through the K steps
    return gx, y.detach()
# More steps K -> smaller bias, but K x the memory/compute (store the path).
```

In practice

`create_graph=True` keeps the graph through each LL step so the implicit term flows back to x.

IF vs. GU: When to Use Which

Implicit Function (IF)

Pros: agnostic to the LL optimizer; theoretically grounded.
Cons: needs the Hessian inverse; IG derivation is non-trivial.

Gradient Unrolling (GU)

Pros: no Hessian inverse; trivially differentiable.
Cons: depends on the LL optimizer/path; cost blows up for large K.

Key point

Both assume an *unconstrained, single-solution* lower level. For constrained / non-unique LL $\Rightarrow$ value-function methods next.

Value-Function Reformulation

For *non-singleton* / general LL, replace it by a value-function constraint:

$$\min_{x,\, y \in \mathcal{C}} \; f(x,y) \quad \text{s.t.} \quad g(x,y) \le v(x) := \min_{y' \in \mathcal{C}} g(x,y').$$

- Any LL solution satisfies $g(x,y) \le v(x)$ with equality $\Rightarrow$ *equivalent* single-level constrained problem.
- Handles constrained *and* non-unique lower levels, where IF/GU break down.
- Catch: the constraint $g(x,y) - v(x) \le 0$ is *non-smooth* and not strictly feasible $\Rightarrow$ standard regularity (constraint qualifications) fail.

Making the Value Function Usable: Smoothing & Feasibility

Smoothed, strictly-feasible surrogate

$$v_a(x) = \min_{y \in \mathcal{C}} \; g(x,y) + \tfrac{a_1}{2}\|y\|^2 + a_2, \qquad a_1, a_2 > 0.$$

- a_1: the inner problem becomes strongly convex, so v_a is differentiable.
- $a_2 > 0$: makes it strictly feasible. The raw constraint $g(x,y) \le v(x)$ can hold only with *equality* (since $v(x) = \min_y g$), so it is always tight: no strictly feasible point exists and Slater's condition fails. Adding a_2 *lifts the bound*, so the inner minimizer satisfies $g \le v_a - a_2 < v_a$, a strict margin.

Result

The relaxed problem is smooth and strictly feasible, so it yields to interior-point or primal-dual methods (next).

Solving the Smoothed VF Problem: BVFIM & PDBO

BVFIM: Bilevel Value-Function Interior-point Method [Liu et al., 2021]

Enforce the margin $v_a(x) - g(x,y) \geq 0$ with a log-barrier:

$$\min_y \; f(x,y) + \tfrac{p_1}{2}\|y\|^2 - p_2 \ln\big(v_a(x) - g(x,y)\big).$$

The barrier blows up at the boundary, keeping y strictly feasible; the objective is smooth and solved by gradient descent.

PDBO: Primal-Dual Bilevel Optimization [Sow et al., 2022]

Attach a multiplier $\lambda \geq 0$ to the value-function constraint and solve the saddle problem:

$$\min_{x,y} \max_{\lambda \geq 0} \; f(x,y) + \lambda\big(g(x,y) - v_a(x)\big).$$

Handles convex lower levels with *multiple* minimizers, where the implicit-function approach breaks down.

Penalty-Based Bilevel Gradient Descent [Shen and Chen, 2023]

General recipe: fold the lower level into the upper objective as a penalty:

$$\min_{x,\, y \in \mathcal{C}} \; f(x,y) + \gamma\, P(x,y).$$

LL-stationarity penalty

$P = \|\nabla_y g(x,y)\|^2$

for unconstrained LL.

Value-function penalty

$P = g(x,y) - v(x)$

for general / constrained LL.

- Avoids Hessian inverses; works under weaker assumptions (constrained, non-strongly-convex, non-unique LL).
- Cost: the penalty introduces nonconvex/nonsmooth structure; analysis is harder, and most results are deterministic.

KKT-Based Reformulation

Replace the lower problem by its first-order (KKT) conditions as constraints:

$$\min_{x,y,\nu} f(x,y) \quad \text{s.t.} \quad \nabla_y g(x,y) = 0 \;\; (+ \text{ LL multipliers } \nu \text{ if constrained}).$$

- Turns bilevel into a *single-level* constrained program.
- Catch: the feasible set is nonconvex; standard constraint qualifications (LICQ/MFCQ) *generically fail* at the LL stationarity manifold.
- Valid when LL is convex (KKT = global optimality); otherwise only *necessary*.

Connections to Variational Inequalities & Equilibria

- LL optimality $\nabla_y g(x, y) = 0$ generalizes, under constraints $y \in \mathcal{C}$, to a variational inequality:

$$\langle \nabla_y g(x, y^*),\, y - y^* \rangle \geq 0 \quad \forall\, y \in \mathcal{C}.$$

- Bilevel then = optimization *over the solution set of a VI* (an MPEC / MPCC).
- Leader–follower = a Stackelberg equilibrium; symmetric play = Nash. Links bilevel to game theory and equilibrium programming.
- This viewpoint extends bilevel to non-unique, constrained, or game-structured lower levels.

Three Problem Classes

Basic-BLO (unconstrained, unique LL)

$\min_x f(x, y^*(x))$ s.t. $y^*(x) = \arg\min_{y \in \mathbb{R}^n} g(x, y)$: the IF theory above applies directly.

Cons-BLO (constrained, unique LL)

$\min_x f(x, y^*(x))$ s.t. $y^*(x) = \arg\min_{y \in \mathcal{C}} g(x, y)$: IG via KKT differentiation (below).

NS-BLO (non-singleton LL)

$\min_{x,\, y \in S(x)} f(x, y)$ s.t. $S(x) = \arg\min_{y \in \mathcal{C}} g(x, y)$: needs value-function / penalty methods.

Rule of thumb

Increasing generality $\Rightarrow$ increasing algorithmic difficulty.

Standard Assumptions & Performance Metrics

Standard assumptions (Basic-BLO)

- ∇f Lipschitz & bounded.
- g twice diff.; $g(x, \cdot)$ strongly convex.
- $\nabla^2_{yy} g, \nabla^2_{xy} g$ Lipschitz & bounded.

Second-order conditions appear because the IG itself is second-order.

Metrics

- ϵ-stationarity: $\|\nabla F(x)\|^2 \leq \epsilon$ (det.) or $\mathbb{E}\|\nabla F(x)\|^2 \leq \epsilon$ (stoch.).
- Sample complexity $G(f, \epsilon)$, $G(g, \epsilon)$: #stochastic gradient/Hessian evaluations to reach it.

So what?

These let us *compare* the algorithms ahead on equal footing.

Checkpoint: The Problem Is Set Up. What Comes Next?

✓ What we've covered

- Three ways to compute the hypergradient: implicit function, gradient unrolling, value function.
- Reformulations (KKT, variational inequalities, penalty); problem classes, standard assumptions, and metrics.

⇒ Coming up next

- First, why the implicit gradient is genuinely hard to compute at scale.
- Then the algorithms: deterministic and stochastic implicit-function methods, cheap inverse-Hessian tricks, inexact hypergradients, and fully first-order methods, with a MAML case study.

Why the Implicit Gradient Is Hard

- Inverse of a Hessian $[\nabla^2_{yy} g]^{-1}$: prohibitive at deep-learning scale.
- Relies on assumptions: lower-level stationarity, a unique LL solution, and a twice-differentiable, invertible LL objective.
- Constraints at the lower level break stationarity: IG needs more careful (KKT/VF) treatment.
- Stochasticity: $y^*(x)$ is never solved exactly $\Rightarrow$ *biased* hypergradients.

Looking ahead

The algorithms ahead approximate or avoid the inverse Hessian.

Design Choices for Bilevel Algorithms

Every bilevel algorithm makes three decisions:

1. Deterministic vs. stochastic

Full gradients/Hessians, or *noisy* estimates from minibatches (ML default).

2. Single- vs. double-loop

Double: #lower-level steps grows with target accuracy ϵ. *Single:* a constant $O(1)$ number of lower-level steps per upper step.

3. Exact vs. approximate implicit gradient

Solve the linear system exactly, or approximate the inverse-Hessian product.

Big picture

These three axes organize the IF algorithms in this session.

The Simplest IF Algorithm: Bilevel Approximation (BA)

"Gradient descent on the implicit gradient" [Ghadimi and Wang, 2018]. At each upper step t:

BA iteration

1. Solve the lower level (approximately) by GD: $y \approx y^*(x_t)$.
2. Form the IG with the *exact* formula $\nabla F = \nabla_x f - \nabla^2_{xy} g\,[\nabla^2_{yy} g]^{-1} \nabla_y f$.
3. One descent step: $x_{t+1} = x_t - \alpha\, \nabla F(x_t)$.

- A double loop: an inner solver re-solves the lower level every upper step.
- Sample complexity: $G(f, \epsilon) = O(\epsilon^{-1})$, $G(g, \epsilon) = O(\epsilon^{-5/4})$.

So what?

Can we avoid the explicit inverse and improve the rate?

Approximate Implicit Differentiation: AID-BiO

Idea [Ji et al., 2021]: never form $[\nabla^2_{yy} g]^{-1}$; *solve a linear system* instead.

The problem AID-BiO solves (inverse-Hessian–vector product)

$$\boxed{\nabla^2_{yy} g\, z = \nabla_y f} \quad \Longleftrightarrow \quad \min_z \ \tfrac{1}{2} z^\top \nabla^2_{yy} g\, z - (\nabla_y f)^\top z,$$

then $z \approx [\nabla^2_{yy} g]^{-1} \nabla_y f$, and $\nabla F = \nabla_x f - \nabla^2_{xy} g\, z$.

1. A fixed number of LL GD steps: $y_{k+1} = y_k - \beta \nabla_y g(x_t, y_k)$.
2. Solve the boxed system by N steps of conjugate gradient (CG) (Hessian-vector products only, no matrix stored).
3. One upper step on the approximate hypergradient.

Complexity

$G(f, \epsilon) = O(\epsilon^{-1})$, $G(g, \epsilon) = O(\epsilon^{-1})$: both linear. The *stochastic* variant (stocBiO) swaps CG for a Neumann-series estimator.

The Inverse-Hessian Bottleneck, and Four Fixes

The crux is the inverse-Hessian–gradient product $H^{-1}g$ with $H = \nabla^2_{yy} g$; at deep-learning scale, forming/inverting H is prohibitive.

Conjugate gradient (CG)

Solve $Hz = g$ as a QP; only HVPs, no inverse.

Hessian-free $H \approx I$

Reasonable for ReLU nets (+ small ℓ_2); used by Fast-BAT & BiP pruning.

WoodFisher

Empirical-Fisher curvature $H \approx \frac{1}{B}\sum_i g_i g_i^\top + \gamma I$; rank-1 recurrence + Woodbury for the inverse.

Neumann series

$H^{-1} = \eta \sum_k (I - \eta H)^k$: a truncated sum of HVPs.

Trade-off

All trade IG *accuracy* against per-step *cost*.

Neumann-Series Approximation of the Inverse [Lorraine et al., 2020]

For $H = \nabla^2_{yy} g$ scaled so $\|I - \eta H\| < 1$,

$$H^{-1} = \eta \sum_{k=0}^{\infty} (I - \eta H)^k \;\approx\; \eta \sum_{k=0}^{N} (I - \eta H)^k.$$

- Apply to a vector with only Hessian–vector products, never forming H or H^{-1}.
- Truncating at N terms gives a geometrically small bias $\sim (1 - \eta \mu_g)^N$.
- Scaled implicit differentiation to millions of hyperparameters.
- CG solves the same system $Hz = \nabla_y f$ at comparable cost (often fewer iterations).

Working Example: Neumann Inverse-Hessian–Vector Product

- **Working example:** compute $[\nabla^2_{yy} g]^{-1} v$ with a truncated Neumann series: HVPs only.

```
import torch
# Approximates H^{-1} v where H = d^2_yy g(x, y), via Neumann series.
def neumann_ihvp(g, y, v, eta=0.1, N=10):
    dyg, = torch.autograd.grad(g, y, create_graph=True) # for HVP
    def Hvp(u): # u -> H u (one backward)
        return torch.autograd.grad(dyg, y, grad_outputs=u, retain_graph=True)[0]
    p = v.clone() # running term (I - eta H)^k v
    s = v.clone() # running sum
    for _ in range(N):
        p = p - eta * Hvp(p) # p <- (I - eta H) p
        s = s + p # accumulate
    return eta * s # ~= H^{-1} v
# Hypergradient: dF = d_x f - d^2_xy g . (H^{-1} d_y f)
```

In practice

Plug the returned vector into the cross term $\nabla^2_{xy} g \cdot (\cdot)$ to finish the IG (cf. Session 1's CG snippet).

Stochastic Bilevel: From Double-Loop to Single-Loop

In ML we only have minibatch gradients/Hessians $\Rightarrow$ stochastic IF methods. A progression:

- BSA [Ghadimi and Wang, 2018]: stochastic BA. Double-loop: an inner SGD solves the LL accurately each step; Hessian inverse via stochastic Neumann.
- TTSA [Hong et al., 2023]: single-loop, *two-timescale*. One LL SGD step per upper step, with $\alpha \ll \beta$ (lower level moves faster). Suited to online / sequential data.
- STABLE [Chen et al., 2022]: single-timescale. A *corrected* LL update (correction $\propto x_{t+1} - x_t$) keeps y tracking $y^*(x)$; momentum on Hessian/Jacobian.
- SUSTAIN [Khanduri et al., 2021]: single-loop, double-momentum (variance reduction on both levels) $\Rightarrow$ near-optimal rate.

Two-Timescale Stochastic Approximation (TTSA) [Hong et al., 2023]

A single-loop method: one lower step and one upper step per iteration, at *two* stepsizes.

TTSA recursion

$$y_{k+1} = y_k - \beta_k \nabla_y g(x_k, y_k; \zeta_{k+1}) \quad \text{(lower: fast timescale)}$$
$$x_{k+1} = x_k - \alpha_k \widehat{\nabla F}(x_k, y_k) \quad \text{(upper: slow timescale)}$$

- $\widehat{\nabla F}$: a stochastic hypergradient using a sampled Neumann Hessian-inverse estimate ($\mathcal{O}(\log \frac{1}{\epsilon})$ samples per step).
- Two timescales: $\alpha_k/\beta_k \to 0$, so the *fast* lower level tracks $y^*(x_k)$ while the *slow* upper level descends F.
- Only $\mathcal{O}(1)$ lower-level samples per step: no inner solve, SGD-like, easy to implement.

Why it matters

First single-loop bilevel method with finite-time guarantees ($\mathcal{O}(\epsilon^{-5/2})$ for a nonconvex upper level); specializes to two-timescale actor-critic in RL.

TTSA: Why the Hypergradient Is Hard to Estimate

The lower-level gradient $G = \nabla_y g$ is easy to sample unbiasedly. The hypergradient is not:

$$\nabla F = \nabla_x f - \nabla^2_{xy} g \underbrace{[\nabla^2_{yy} g]^{-1}}_{\text{cannot sample directly}} \nabla_y f.$$

- Plugging in one stochastic Hessian is biased: $\mathbb{E}\big[[\nabla^2_{yy} g(\cdot;\zeta)]^{-1}\big] \neq [\nabla^2_{yy} g]^{-1}$ (inverse of a mean $\neq$ mean of inverses).
- Fix: write $H := \nabla^2_{yy} g$; strong convexity gives $0 \prec \mu_g I \preceq H \preceq L_g I$. Expand as a Neumann series:

$$H^{-1} = \tfrac{1}{L_g} \sum_{k=0}^{\infty} \big(I - \tfrac{1}{L_g} H\big)^k.$$

- Each term is a *product of Hessians*, so it can be sampled.

TTSA: The Hypergradient Estimator $\widehat{\nabla F} = h_f$ [Hong et al., 2023]

Subroutine (input $\mathtt{t}$)

1. Draw $p \in \{0, \dots, \mathtt{t}-1\}$ uniformly at random (a randomized truncation of the series).
2. Form the estimate

$$h_f = \nabla_x f(\cdot;\xi) - \nabla^2_{xy} g(\cdot;\zeta_0)\Big[\tfrac{\mathtt{t}}{L_g}\textstyle\prod_{i=1}^{p}\big(I - \tfrac{c_h}{L_g}\nabla^2_{yy} g(\cdot;\xi_i)\big)\Big]\nabla_y f(\cdot;\xi).$$

- Uses only Hessian-vector products; draws at most $\mathtt{t}+1$ samples per step.
- Guarantee: bias $\|\nabla F - \mathbb{E}[h_f]\| = \mathcal{O}\big((1-\mu_g/L_g)^{\mathtt{t}}\big)$ (geometric in $\mathtt{t}$); variance $\mathcal{O}(\sigma^2)$.
- So $\mathtt{t} = \mathcal{O}(\log \frac{1}{\epsilon})$ makes the bias $\mathcal{O}(\epsilon)$: a controllable-bias stochastic gradient.

TTSA: Standing Assumptions

Upper level f

$\nabla_y f$ Lipschitz and bounded; $\nabla_x f$ Lipschitz in y.

Lower level g (the key structural assumption)

- $g(x, \cdot)$ is strongly convex in y: the solution $y^*(x)$ is unique and $\nabla^2_{yy} g \succ 0$.
- $\nabla^2_{xy} g$, $\nabla^2_{yy} g$ are Lipschitz; $\nabla^2_{xy} g$ is bounded.

Consequence

$y^*(\cdot)$ is Lipschitz and the hyper-objective $\ell(x) = f(x, y^*(x))$ is smooth, so the hypergradient is well-defined.

TTSA: Why Two Timescales?

The upper step uses $\widehat{\nabla F}(x_k, y_k)$, but $y_k \neq y^*(x_k)$. This bias is controlled by tracking:

$$\Delta_y^k = \mathbb{E}\big[\|y_k - y^*(x_{k-1})\|^2\big] \qquad \text{(lower-level tracking error)}.$$

- If the upper level moves slowly ($\alpha_k/\beta_k \to 0$), the target $y^*(x_k)$ drifts slowly, so the fast lower level keeps up and $\Delta_y^k \to 0$.
- Then the hypergradient bias vanishes, and the upper update behaves like SGD on $\ell(x)$.
- Progress is measured by the optimality gap $\|x_k - x^*\|^2$ (convex) or stationarity $\|\nabla \ell(x_k)\|^2$ (nonconvex).

TTSA: Convergence Rates [Hong et al., 2023]

With two-timescale stepsizes, after K iterations the tracking error and the upper-level measure both decay:

Upper $\ell(x)$	stepsizes (α_k, β_k)	outer rate	inner rate
strongly convex	$(k^{-1}, k^{-2/3})$	$O(K^{-2/3})$	$O(K^{-2/3})$
convex	$(K^{-3/4}, K^{-1/2})$	$O(K^{-1/4})$	$O(K^{-1/2})$
nonconvex	$(K^{-3/5}, K^{-2/5})$	$O(K^{-2/5})$	$O(K^{-2/5})$

- In every case $\alpha_k \ll \beta_k$: the upper level is slower than the lower.
- Nonconvex $O(K^{-2/5})$ on $\|\nabla \ell\|^2$ gives sample complexity $\mathcal{O}(\epsilon^{-5/2})$.

Stochastic IF Methods: Loop Structure & Complexity

Method	Loop structure	$G(f,\epsilon)$ / $G(g,\epsilon)$
BSA [Ghadimi and Wang, 2018]	double-loop	ϵ^{-2} / ϵ^{-3}
TTSA [Hong et al., 2023]	single-loop, two-timescale	$\epsilon^{-5/2}$ / $\epsilon^{-5/2}$
STABLE [Chen et al., 2022]	single-loop, single-timescale	ϵ^{-2} / ϵ^{-2}
SUSTAIN [Khanduri et al., 2021]	single-loop, double-momentum	$\epsilon^{-3/2}$ / $\epsilon^{-3/2}$
SABA [Dagréou et al., 2022]	single-loop, SAGA-type (VR)	$\epsilon^{-3/2}$-type
F^{2}SA [Kwon et al., 2023]	single-loop, Hessian-free	ϵ^{-2}, no Hessian

- TTSA trades a *worse* f-rate for a *better* g-rate vs. BSA, but is single-loop.
- Momentum / variance reduction (SUSTAIN, SABA) push toward the single-level optimum.

The Convergence-Rate Landscape

Sample complexity to reach an ϵ-stationary point ($\|\nabla F\|^2 \leq \epsilon$):

Regime	Representative rate	Method
Deterministic, LL-SC	$O(\epsilon^{-1})$	AID-BiO [Ji et al., 2021]
Stochastic, double-loop	$O(\epsilon^{-2})$	BSA [Ghadimi and Wang, 2018]
Stochastic, single-loop	$O(\epsilon^{-2})$	STABLE [Chen et al., 2022]
+ variance reduction	$O(\epsilon^{-3/2})$	SUSTAIN/SABA [Khanduri et al., 2021, Dagréou et al.,

- Bilevel SGD now matches single-level SGD: ϵ^{-2}, or $\epsilon^{-3/2}$ with VR, and the inner solve no longer dominates.
- All rates assume LL strong convexity; relaxing it is an active frontier (end of session).

A Descent Lemma Under *Inexact* Hypergradients

Two errors enter every practical IG: (i) inexact $y \approx y^*(x)$; (ii) truncated inverse-Hessian. Treat the net effect as a *bias* b.

Lemma (Descent with a biased hypergradient)

Let F be L-smooth, step $\alpha \le 1/L$, and let $\widehat{\nabla} F$ satisfy $\|\widehat{\nabla} F(x) - \nabla F(x)\| \le b$. Then $x^+ = x - \alpha \widehat{\nabla} F(x)$ obeys

$$F(x^+) \le F(x) - \tfrac{\alpha}{2}\|\nabla F(x)\|^2 + \tfrac{\alpha}{2} b^2.$$

Proof (sketch).

L-smoothness gives $F(x^+) \le F(x) + \langle \nabla F, x^+ - x\rangle + \frac{L}{2}\|x^+ - x\|^2$. Substitute $x^+ - x = -\alpha \widehat{\nabla} F$, write $\widehat{\nabla} F = \nabla F + e$ with $\|e\| \le b$, use $\alpha \le 1/L$ and $\langle \nabla F, e\rangle \le \frac{1}{2}\|\nabla F\|^2 + \frac{1}{2} b^2$. □

Takeaway

Drive $b \to 0$ (more inner / Neumann / CG steps) $\Rightarrow$ standard (S)GD analysis $\Rightarrow$ stationarity.

Truncation, Warm-Starting, and Bias Control

- Warm-starting: reuse the previous y as the next inner init; as x moves slowly, far fewer inner steps reach a target bias [Arbel and Mairal, 2022].
- Truncated unrolling / inversion [Shaban et al., 2019]: keep only the last few inner steps; bias decays *geometrically* in the number kept.
- Budgeting: per the descent lemma, set $b^2 = O(\|\nabla F\|^2)$ so the bias never dominates the descent term: inner work scales with how close to stationarity you are.

Bottom line

Net effect: hypergradient error becomes a *controllable* bias term in the rate.

Checkpoint: Can We Avoid the Hessian Entirely?

✓ What we've covered

- Implicit-function methods (BA, AID-BiO, TTSA, ...) all form or approximate the inverse-Hessian–vector product in the hypergradient.
- Cheap-Hessian tricks (CG, Neumann, Hessian-free) and inexact hypergradients *reduce* the second-order cost, but do not *remove* it.

⇒ Coming up next

- Fully first-order / Hessian-free methods: estimate the hypergradient using *only gradients*, with no Hessian, Jacobian, or inverse.
- Essential at foundation-model scale, where Hessian-vector products are impractical (Session 3).

Recall: Penalty & Value-Function Reformulation

From Part A: using the value function $v(x) = \min_y g(x, y)$, the bilevel problem becomes a single-level penalty:

$$\min_{x,y} \; f(x, y) + \gamma\big(g(x, y) - v(x)\big), \qquad \gamma > 0 \text{ large.}$$

- The penalty $g(x, y) - v(x) \geq 0$ vanishes exactly when $y = y^*(x) = \arg\min_y g(x, y)$.
- As $\gamma \to \infty$, the penalized solution approaches the true bilevel solution, recovering $F(x) = f(x, y^*(x))$.
- Question: can we compute ∇F from this penalty using *only gradients*, with no Hessian?

Next

F^{2}SA answers yes: a fully first-order hypergradient built from the value-function difference.

Fully First-Order: The F^{2}SA Idea [Kwon et al., 2023]

Goal: minimize the hyper-objective $F(x) := f\big(x, y^*(x)\big)$ using *only gradients*, where $y^*(x) = \arg\min_y g(x, y)$.

- Introduce a penalized lower solution (γ large): $\hat{y}_\gamma(x) = \arg\min_y \left[g(x, y) + \frac{1}{\gamma} f(x, y)\right] \to y^*$.
- With value functions $v(x) = \min_y g(x, y)$ and $v_\gamma(x) = \min_y [g + \frac{1}{\gamma} f]$, one has the identity

$$F(x) = \lim_{\gamma \to \infty} \gamma\big(v_\gamma(x) - v(x)\big).$$

- Differentiating (Danskin, so *no* inner derivative is needed):

$$\boxed{\nabla F(x) \approx \nabla_x f(x, \hat{y}_\gamma) + \gamma\big[\nabla_x g(x, \hat{y}_\gamma) - \nabla_x g(x, y^*)\big]}$$

- Only gradients $\nabla_x f, \nabla_x g$ at two points: no Hessian, Jacobian, or inverse.

F^{2}SA: The Algorithm and Its Cost

Track the two lower-level solutions by SGD; update x by the gradient difference:

F^{2}SA updates

$$y_{k+1} = y_k - \beta \nabla_y g(x_k, y_k) \qquad \text{(track } y^*\text{)}$$

$$z_{k+1} = z_k - \beta \nabla_y \left[g + \tfrac{1}{\gamma} f \right](x_k, z_k) \qquad \text{(track } \hat{y}_\gamma\text{)}$$

$$x_{k+1} = x_k - \alpha \Big(\nabla_x f(x_k, z_k) + \gamma \left[\nabla_x g(x_k, z_k) - \nabla_x g(x_k, y_k) \right] \Big)$$

- Every step is a first-order oracle call; no second-order object is ever formed.
- Complexity: a polynomial first-order-oracle rate; avoiding second-order info costs a higher power of $1/\epsilon$ (and a γ variance factor), which later works reduce.

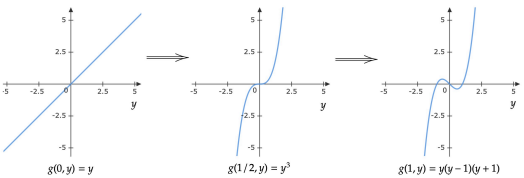
Why it matters

Simpler updates, friendly at foundation-model scale (Session 3); the price is penalty tuning (γ) and higher variance.

Counter-Example: Why Non-Convex Lower Levels Break

Drop lower-level strong convexity and the hyper-objective can fail to be well-behaved:

- The solution set $S(x) = \arg\min_y g(x, y)$ can change discontinuously as x varies (a bifurcation): minimizers appear or disappear.
- Then $\ell(x) = f(x, y^*(x))$ is discontinuous, and practically not even computable.
- At a bifurcation, $\nabla^2_{yy} g$ degenerates, so the implicit gradient blows up and $\|\hat{y} - y^*(x)\|$ cannot be controlled.



The lower-level optimal set jumps as x crosses a threshold.

Frontier: Removing Lower-Level Strong Convexity

Every rate above assumed LL strong convexity. Without it, $F(x)$ can be discontinuous (the LL solution set jumps as x moves).

- Relaxed conditions: replace strong convexity by the *Polyak–Łojasiewicz (PL)* condition; penalty/VF methods still reach $\widetilde{O}(\epsilon^{-2})$-type rates [Shen and Chen, 2023].
- LP / constrained lower levels (e.g. $y^*(x) \in \arg\min g$ over a polytope): a log-barrier smoothing $g(x,y) - t\sum_i \log(-h_i(x,y))$ restores a differentiable surrogate.
- Fully first-order methods (F^2SA) extend most naturally to these regimes [Kwon et al., 2023].
- Survey of the modern landscape: [Yang et al., 2025].

Big picture

These are exactly the tools we need for foundation models (Session 3).

Session 2: Takeaways

- One fingerprint: every method forms or avoids the inverse-Hessian product in ∇F.
- IF (CG/Neumann/Hessian-free) is optimizer-agnostic; GU differentiates the LL path (cost grows with K); VF/penalty handle constrained/non-unique LL.
- Rates have caught up: stochastic bilevel now matches single-level SGD: $O(\epsilon^{-2})$, and $O(\epsilon^{-3/2})$ with variance reduction.
- Inexact hypergradients are fine: a controllable bias b enters the descent lemma; drive $b \to 0$ via inner/Neumann/CG steps and warm-starting.
- Next (Session 3): *applications* of this template, from classical ML (meta-learning, adversarial training, pruning, dataset selection, IRM) to foundation models (RLHF, alignment, inverse RL).

Session 3: Outline

1. Classical ML applications: meta-learning (MAML), adversarial training (Fast-BAT), model pruning (BiP), dataset selection, and invariant risk minimization (IRM).
2. Foundation models: the alignment problem and RLHF; alignment as bilevel; DPO; inverse RL as bilevel; maximum-likelihood IRL and a single-loop algorithm with guarantees.
3. Results & frontiers: MuJoCo and LLM alignment; AIHF (joint reward+policy); offline IRL and reward transfer; bilevel across the LLM stack; open problems.

Roadmap

One template, applied: from classical ML to foundation models.

Bilevel in Classical Machine Learning

✓ **What we've covered**

- Theory: three ways to compute the hypergradient, reformulations, problem classes.
- Algorithms: deterministic and stochastic IF, cheap-Hessian tricks, and fully first-order methods.

⇒ **Coming up next**

- The same bilevel template, applied: meta-learning (MAML), adversarial training (Fast-BAT), model pruning (BiP), dataset selection, and invariant risk minimization.
- Each shown as a formulation, a solver choice, and results; then foundation models follow.

Case Study: MAML as Bilevel

Meta-learning a good initialization [Finn et al., 2017]: upper = validation of the fine-tuned model; lower = per-task fine-tuning from the shared init x:

$$\min_x \ \frac{1}{N}\sum_{i=1}^{N} \ell_i\big(y_i^*(x); \mathcal{D}_i^{\text{val}}\big), \quad y_i^*(x) = \arg\min_y \ell_i(y; \mathcal{D}_i^{\text{tr}}, x).$$

With a one-step inner solve $y_i^*(x) = x - \beta\nabla\ell_i(x; \mathcal{D}_i^{\text{tr}})$, the hypergradient is

$$\nabla F(x) = \sum_{i=1}^{N} \big(I - \beta\,\nabla^2\ell_i(x)\big)\,\nabla\ell_i\big(y_i^*(x); \mathcal{D}_i^{\text{val}}\big).$$

Key point

The Hessian term $I - \beta\nabla^2\ell_i$ is exactly what the solvers below *keep*, *drop*, or *invert*.

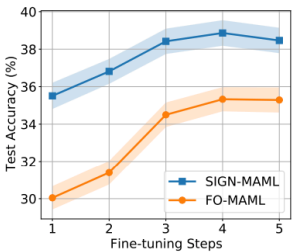
MAML Solvers: What Each One Does

Same bilevel problem; the solvers differ in how they treat the Hessian term $I - \beta\nabla^2\ell_i$:

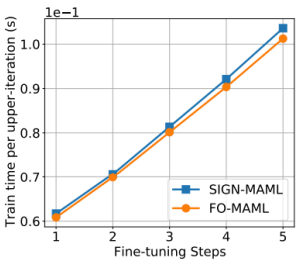
- MAML [Finn et al., 2017] (keep it, full unrolling, second-order): $x \leftarrow x - \alpha \sum_i \left(I - \beta\nabla^2\ell_i(x)\right)\nabla\ell_i(y_i^*; \text{val})$.
- FO-MAML [Nichol et al., 2018] (drop it, $I - \beta\nabla^2\ell_i \approx I$): $x \leftarrow x - \alpha \sum_i \nabla\ell_i(y_i^*; \text{val})$.
- Sign-MAML [Fan et al., 2021] (sign inner step): $y_i^* = x - \beta\,\mathrm{sign}(\nabla\ell_i(x))$; sign has zero derivative, so $\frac{dy_i^*}{dx} = I$, giving a first-order update.
- iMAML [Rajeswaran et al., 2019] (invert it, proximal inner $\ell_i + \frac{\lambda}{2}\|y - x\|^2$): IG $= \left(I + \frac{1}{\lambda}\nabla^2\ell_i(y_i^*)\right)^{-1}\nabla\ell_i(y_i^*; \text{val})$ via CG; optimizer-agnostic.

MAML Results: Sign-MAML Closes the Gap Cheaply

- Full MAML/iMAML are most accurate but much slower (true unrolling / inverse).
- Among first-order methods, Sign-MAML beats FO-MAML in accuracy at *near-identical* per-iteration time.
- More fine-tuning steps help, but cost grows.



(a) Accuracy 10-way 2-shot



(b) Time 10-way 2-shot

10-way 2-shot: (a) test accuracy and (b) train time per upper-iteration vs. fine-tuning steps.

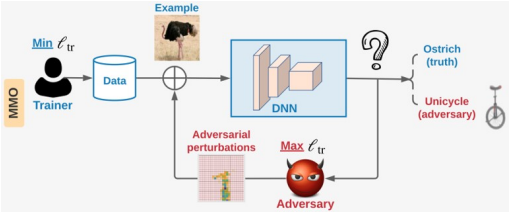
Adversarial Training as a Constrained Bilevel Problem

Give the attacker its own objective [Zhang et al., 2022b]:

$$\min_{\theta} \ \mathbb{E}_x\big[\ell_{\text{tr}}(\theta, \delta^*(\theta))\big]$$

$$\text{s.t.} \quad \delta^*(\theta) = \arg\min_{\delta \in \mathcal{C}} \ \ell_{\text{atk}}(\theta, \delta), \quad \mathcal{C} = \{\|\delta\|_\infty \leq \epsilon\}.$$

- Box-constrained LL (special case of linear) + strongly convex.
- Min–max AT is the special case $\ell_{\text{atk}} = -\ell_{\text{tr}}$.
- Goal: attack-agnostic, scalable robust training.



Adversarial example + min–max view of AT.

Fast-BAT: Lower-Level Linearization $\Rightarrow$ Closed Forms

Key trick: linearize ℓ_{atk} at z with quadratic regularization,

$$\ell_{\text{atk}}(\theta, \delta) \approx \langle \nabla_\delta \ell_{\text{atk}}(\theta, z),\, \delta - z \rangle + \tfrac{\lambda}{2} \|\delta - z\|^2.$$

- Closed-form attack (one-step PGD): $\delta^*(\theta) = \text{Proj}_{\mathcal{C}}(u), \quad u := z - \frac{1}{\lambda} \nabla_\delta \ell_{\text{atk}}(\theta, z).$
- Closed-form implicit gradient (KKT differentiation; the quadratic reg makes $\nabla^2_{\delta\delta} \to \lambda I$):

$$\boxed{\frac{d\delta^*}{d\theta} = -\tfrac{1}{\lambda}\, H \nabla^2_{\delta\theta} \ell_{\text{atk}}(\theta, z)}, \qquad H = \text{diag}\big(\mathbb{1}[\,|u_i| < \epsilon\,]\big).$$

- H is the active-set mask: 1 on *interior* (unclamped) coordinates, 0 on those clamped to a box face $\pm\epsilon$ (an "indicator over active box faces").
- Fast-BAT $\supset$ Fast-AT: replacing the linearization by its *sign* kills the IG ($\frac{d\,\text{sign}}{d\cdot} = 0$) and recovers Fast-AT [Wong'20].

The Fast-BAT Algorithm vs. Traditional AT

Every method alternates an inner attack on δ with an outer update on θ:

PGD-AT [Madry et al., 2018]: min–max, K inner steps

Inner ($K\times$): $\delta \leftarrow \mathrm{Proj}_{\mathcal{C}}\big(\delta + \eta\,\mathrm{sign}(\nabla_\delta \ell_{\mathrm{tr}}(\theta,\delta))\big)$; Outer: $\theta \leftarrow \theta - \alpha\,\nabla_\theta \ell_{\mathrm{tr}}(\theta,\delta)$.

Fast-AT [Wong et al., 2020]: one step, *no* implicit gradient

Inner ($1\times$): $\delta = \mathrm{Proj}_{\mathcal{C}}\big(\eta\,\mathrm{sign}(\nabla_\delta \ell_{\mathrm{tr}}(\theta,0))\big)$; Outer: $\theta \leftarrow \theta - \alpha\,\nabla_\theta \ell_{\mathrm{tr}}(\theta,\delta)$.

Fast-BAT [Zhang et al., 2022b]: one step + implicit gradient

Inner ($1\times$, closed form): $\delta^* = \mathrm{Proj}_{\mathcal{C}}\big(z - \frac{1}{\lambda}\nabla_\delta \ell_{\mathrm{atk}}(\theta,z)\big)$;

Outer: $\theta \leftarrow \theta - \alpha\Big(\nabla_\theta \ell_{\mathrm{tr}}(\theta,\delta^*) + \big[\frac{d\delta^*}{d\theta}\big]^\top \nabla_\delta \ell_{\mathrm{tr}}(\theta,\delta^*)\Big)$.

Key difference

Only Fast-BAT keeps the implicit gradient $\frac{d\delta^*}{d\theta}$ (closed form, box active-set mask), at one-step cost.

Fast-BAT: Robust *and* Fast

- One IG-aware SGD step on θ + one closed-form PGD step on δ.
- Deterministic IF approach over the constrained LL; sample complexity $O(\epsilon^{-1})$.
- Empirically: best robust accuracy among fast-AT baselines (Fast-AT, Fast-AT-GA, BackSmooth) at comparable runtime.

Takeaways

- AT = lower-level constrained BLO.
- Linearization $\Rightarrow$ closed-form attack *and* IG.
- Fast-AT is the sign-linearized special case.

Code: `github.com/OPTML-Group/Fast-BAT`

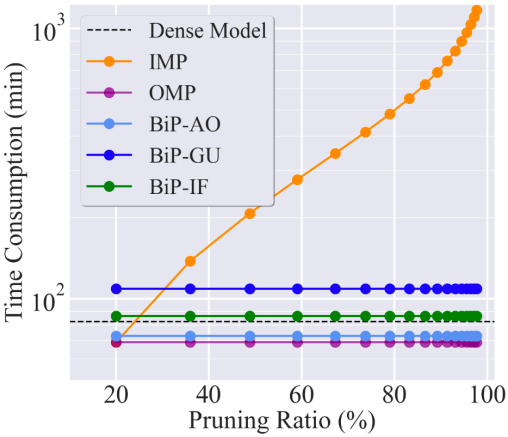
Model Pruning as Bilevel: Bilinear Structure

Prune but keep accuracy [Zhang et al., 2022a]:

$$\min_{m} \ \ell_{\text{prune}}\big(m \odot \theta^*(m)\big)$$

$$\text{s.t.} \ \ \theta^*(m) = \arg\min_{\theta} \ \ell_{\text{tr}}(m \odot \theta) + \tfrac{\gamma}{2}\|\theta\|^2.$$

- Upper: pruning mask m. Lower: mask-fixed retraining.
- Bilinear variables $m \odot \theta$; mismatched prune/retrain objectives.
- SOTA before: iterative magnitude pruning (IMP), accurate but re-trains T times.



IMP time explodes with sparsity; BiP is sparsity-agnostic.

BiP: Why the Hessian-Free IG Collapses to a Diagonal

Lower level $g(m,\theta) = \ell_{\text{tr}}(m \odot \theta) + \frac{\gamma}{2}\|\theta\|^2$, with bilinear coupling $m \odot \theta$. Stationarity, componentwise ($z = m \odot \theta$):

$$(\nabla_\theta g)_i = m_i \left(\nabla \ell_{\text{tr}}(z)\right)_i + \gamma\, \theta_i = 0.$$

Why the cross term is a diagonal of first-order gradients

Differentiate in m_j:

$$\frac{\partial (\nabla_\theta g)_i}{\partial m_j} = \underbrace{\delta_{ij}\left(\nabla \ell_{\text{tr}}(z)\right)_i}_{\text{diagonal (explicit } m_i)} + \underbrace{m_i\, \theta_j \left(\nabla^2 \ell_{\text{tr}}(z)\right)_{ij}}_{\approx 0 \text{ (Hessian-free)}}.$$

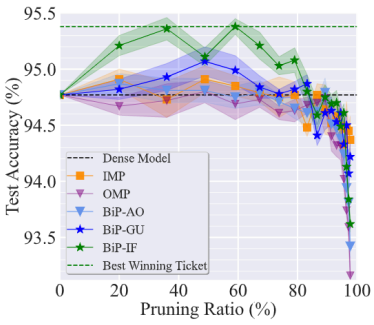
So $\nabla^2_{m\theta} g \approx \operatorname{diag}\left(\nabla \ell_{\text{tr}}(z)\right)$ and $\left[\nabla^2_{\theta\theta} g\right]^{-1} \approx \frac{1}{\gamma} I$, giving

$$\frac{d\theta^{*\top}}{dm} = -\nabla^2_{m\theta} g \left[\nabla^2_{\theta\theta} g\right]^{-1} \approx -\frac{1}{\gamma} \operatorname{diag}\left(\nabla \ell_{\text{tr}}(m \odot \theta^*)\right).$$

Key point

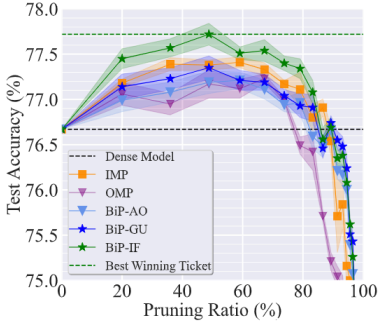
BiP Results: Better Accuracy, Sparsity-Agnostic Time

- BiP-IF matches/beats the best "winning ticket" across sparsities.
- Accounting for the coupling (IG) matters: BiP-IF/GU > BiP-AO (ignores IG).
- Cost is flat in sparsity (prev. slide), unlike IMP.



(a) CIFAR-10

CIFAR-10



(b) CIFAR-100

CIFAR-100

Bottom line

Clean bilevel formulation + implicit gradient = better *and* faster.

Code: `github.com/OPTML-Group/BiP`

Dataset / Coreset Selection as Bilevel

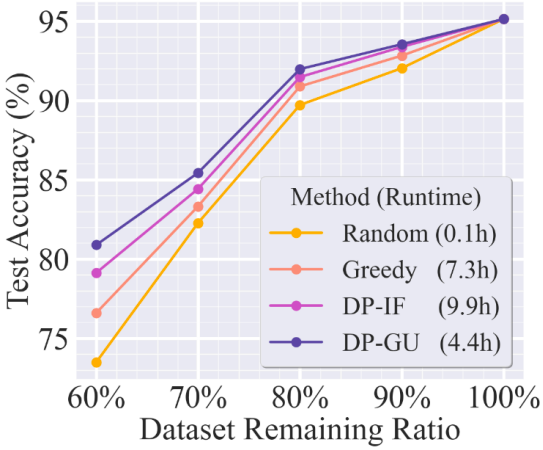
Weight (select) data above, fit a model below:

$$\min_{w \in \Omega} \frac{1}{N} \sum_{i=1}^{N} \ell_i(\theta^*(w))$$

$$\text{s.t. } \theta^*(w) = \arg\min_{\theta} \sum_{i=1}^{N} w_i \ell_i(\theta),$$

$\Omega = \{w \in [0,1]^N : \|w\|_0 \leq k\}$ (keep k samples).

- Solvers: DP-IF (WoodFisher inverse), DP-GU (10-step unroll + prox on w).
- Both bilevel solvers beat Greedy / Random.



CIFAR-10: test acc. vs. remaining ratio. DP-GU is best *and* cheapest (4.4h vs. DP-IF 9.9h); DP-IF/Greedy pay for the Hessian inverse.

Invariant Risk Minimization as Consensus-Constrained Bilevel

Learn a representation θ invariant across E environments [Zhang et al., 2024]. Decouple per-environment predictors ϕ_i, tied by a consensus constraint:

$$\min_{\theta} \sum_{i=1}^{E} \left[\ell_i(\phi_i^*(\theta) \circ \theta) + \gamma \|\nabla_{\phi_i} \ell_i(\phi_i^*(\theta) \circ \theta)\|^2 \right]$$

$$\text{s.t. } \{\phi_i^*(\theta)\} = \arg\min_{\{\phi_i\} \in \mathcal{C}} \sum_{i=1}^{E} \ell_i(\phi_i \circ \theta), \quad \mathcal{C} = \{\phi_1 = \cdots = \phi_E\}.$$

- GU + consensus projection ($\phi_i^* = \frac{1}{E} \sum_i \phi_i^{(K)}$) gives a closed-form unrolled LL solution.
- Turns a non-separable, non-singleton problem into *decomposable* per-environment fits + consensus.

IRM Results: Bilevel Closes the Gap

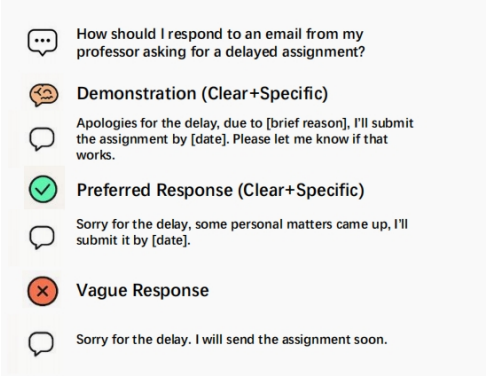
- ERM: high accuracy gap across environments (not invariant).
- All IRM variants beat ERM; IRM-BLO (consensus-constrained bilevel) is best on both metrics.
- Best avg. accuracy *and* smallest accuracy gap.

Method	BLO (Solver)	Colored-MNIST		Colored-FashionMNIST	
		Avg. Acc.	Acc. Gap	Avg. Acc.	Acc. Gap
ERM	N/A	49.19±1.89	90.72±2.08	49.77±1.71	88.62±2.49
IRM-v1	N/A	68.33±0.31	2.04±0.05	68.76±0.31	1.45±0.09
IRM-Game	N/A	67.73±0.24	1.67±0.14	67.49±0.32	1.82±0.13
IRM-BLO	GU	**69.47**±0.24	**1.04**±0.07	**69.43**±0.21	**1.14**±0.11

Colored-MNIST / Colored-FashionMNIST: avg. acc. (higher) & acc. gap (lower better).

The LLM Alignment Problem

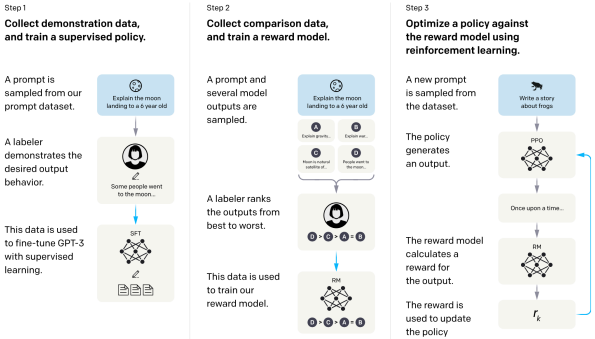
- Goal: make an LLM produce text that conforms to human preferences and values [Ouyang et al., 2022].
- Data is fixed and offline: human demonstrations $\mathcal{D}_d$ and pairwise preferences $\mathcal{D}_p$.
- A special case of "improving AI through experience": learn a reward and a policy from human data.
- This is naturally a bilevel problem, and the structure pays off.
- Good news: structure $\Rightarrow$ *efficient* hypergradients and improved alignment.



Demonstration vs. preferred vs. vague responses.

The RLHF Pipeline [Ouyang et al., 2022]

Three steps: SFT $(\mathcal{D}_d) \to$ reward model $(\mathcal{D}_p) \to$ RL policy opt. (PPO).



A *sequential* pipeline: we will see it is one bilevel program.

RLHF: The Three Objectives

SFT: $\min_{\pi} \; -\mathbb{E}_{(s,a)\sim\mathcal{D}_d}\big[\log \pi(a \mid s)\big]$ (clone the expert)

RM: $\min_{\theta} \; -\mathbb{E}_{(s,a_w,a_l)\sim\mathcal{D}_p}\big[\log \sigma\big(r(s,a_w;\theta) - r(s,a_l;\theta)\big)\big]$ (Bradley–Terry)

Policy: $\max_{\pi} \; \mathbb{E}_{s,\,a\sim\pi(\cdot|s)}\Big[\, r(s,a;\theta) - \beta \log \frac{\pi(a|s)}{\pi_{\mathrm{SFT}}(a|s)} \Big]$ (KL-regularized RL)

- The RM uses the Bradley–Terry model: $P(a_w \succ a_l) = \sigma\big(r_w - r_l\big)$.
- The policy step is KL-regularized toward π_{SFT} with coefficient β, to stay close to a fluent base.

Alignment Is a Bilevel Problem

Reward parameter θ (upper) sits over the RL-optimal policy $\pi^*(\theta)$ (lower):

$$\min_{\theta} \ \ell_{\text{pref}}\big(\theta,\ \pi^*(\theta)\big) \quad \text{s.t.} \quad \pi^*(\theta) \in \arg\max_{\pi} \ \mathbb{E}_{\pi}\big[r(s,a;\theta)\big] - \beta\,\mathrm{KL}\big(\pi \,\|\, \pi_{\mathrm{SFT}}\big).$$

- Upper: choose the reward so its induced policy matches human preferences/demonstrations.
- Lower: the RL policy that is optimal under the current reward.
- Exactly the two-timescale leader–follower structure of Sessions 1–2: reward leads, policy follows.
- Task 1 (policy opt.) and Task 2 (reward estimation) folded into one bilevel program.

DPO: Collapsing the Bilevel [Rafailov et al., 2023]

The KL-regularized lower level has a closed-form optimal policy:

$$\pi^*(a \mid s) = \tfrac{1}{Z(s)}\, \pi_{\mathrm{SFT}}(a \mid s)\, \exp\big(\tfrac{1}{\beta} r(s,a;\theta)\big) \implies r(s,a;\theta) = \beta \log \tfrac{\pi^*(a|s)}{\pi_{\mathrm{SFT}}(a|s)} + \underbrace{\beta \log Z(s)}_{\text{const in } a}.$$

- Substitute this implicit reward into the Bradley–Terry loss $\Rightarrow$ a single-level objective in π: no separate RM, no RL loop ("your LM is secretly a reward model").

Caveats: why the *general* bilevel does not collapse

- Needs the *exact* KL-regularized optimum π^*; an SGD-trained finite network only approximates it.
- Works only for the Bradley–Terry pairwise loss, where the intractable $\log Z(s)$ *cancels* between the two responses; demonstrations or other losses do not cancel.
- The reward is identified only up to the per-prompt constant $\log Z(s)$: DPO yields no standalone, transferable reward model.
- Purely offline on the preference set: prone to distribution shift / reward hacking off-distribution.

Working Example: The DPO Loss in PyTorch

- **Working example:** the bilevel collapse becomes ≈ ten lines: log-prob ratios + `logsigmoid`.

```
import torch.nn.functional as F
def dpo_loss(policy, ref, s, a_w, a_l, beta=0.1):
    # log-probabilities of chosen / rejected answers under policy & frozen ref
    lp_w = policy.logprob(a_w, s); lp_l = policy.logprob(a_l, s)
    lr_w = ref.logprob(a_w, s); lr_l = ref.logprob(a_l, s) # pi_SFT
    # implicit rewards r = beta * log(pi/pi_SFT) (DPO reparameterization)
    rew_w = beta * (lp_w - lr_w)
    rew_l = beta * (lp_l - lr_l)
    # Bradley-Terry: maximize sigma(r_w - r_l) -> minimize -logsigmoid
    return -F.logsigmoid(rew_w - rew_l).mean()
```

Key point

The reward never appears explicitly: it is *absorbed* into the policy's log-ratio.

Preference Learning & Reward Hacking

- Bradley–Terry maps pairwise comparisons to a reward: $P(a_w \succ a_l) = \sigma(r_w - r_l)$.
- Rewards are identifiable only *up to* a shift/scale: many rewards explain the same preferences ($\Rightarrow$ an ill-posed upper level).
- Reward hacking: the policy exploits errors in an imperfect learned reward, drifting from intent.
- Bilevel remedies: regularize the reward (KL, demonstrations); learn reward + policy *jointly* rather than in a brittle sequence (AIHF, later).

BLO for Alignment

- **Task 1 policy optimization**: For a given reward $r(s, a; \theta)$, find the best policy π_θ
- **Task 2 reward estimation**: Find the reward $r(s, a; \theta)$, whose optimal policy matches the expert policy
- The Maximum Likelihood principle: The actions generated by the desired policy should be **most likely** if the ground truth is the expert policy π^E.
- Use BLO to integrate them into a single problem ([Zeng-Li-Garcia-H. 22][1] [Zeng-Garcia-H. 24]) [2]

[1] S. Zeng, C. Li, A. Garcia, & M. Hong. Maximum-likelihood inverse reinforcement learning with finite-time guarantees. **NeurIPS**, 2022.

[2] S. Zeng, M. Hong, A. Garcia, Structural estimation of markov decision processes in high-dimensional state space with finite-time guarantees, **Operations Research**, 2024.

BLO for Alignment

- We consider the following formulation:

$$\max_{\theta} \; L(\theta) := \mathbb{E}_{\tau^{\mathrm{E}} \sim \pi^{\mathrm{E}}} \left[\log \pi_\theta^\star(a \mid s) \right] \qquad \text{(ML-BLO)}$$

$$s.t. \;\; \pi_\theta^\star := \arg\max_{\pi} \; \mathbb{E}_{\tau \sim \pi} \left[r(s, a; \theta) + \mathcal{H}(\pi(\cdot \mid s)) \right]$$

- $\mathcal{H}(\pi(\cdot|s))$ is either a entropy regularizer or KL regularizer
- $\pi_\theta^\star$ denotes the optimal policy when the reward parameter is θ
- **Challenge:** Is the lower-level problem even convex? How to compute the gradient $\nabla L(\theta)$? Hessian computation? Online expert interaction?

Resolve the Challenge: Online to Offline Data

- To avoid online interaction with the expert $\tau \sim \pi^E$, observe the following

$$L(\theta) := \mathbb{E}_{\tau \sim \pi^E}\Big[r(s_t, a_t; \theta)\Big] - \mathbb{E}_{s_0 \sim \rho}\Big[V_\theta(s_0)\Big]$$

  where $V_\theta(s_0)$ denotes the value function for a given reward $r(\cdot; \theta)$.
- Consider the following surrogate loss

$$\widehat{L}(\theta; \mathcal{D}) := \mathbb{E}_{\tau^E \sim \mathcal{D}}\Big[r(s_t, a_t; \theta)\Big] - \mathbb{E}_{s_0 \sim \rho}\Big[V_\theta(s_0)\Big]$$

  where $\mathcal{D}$ is a set of offline demonstration data
- It turns out that we have the following:

$$|L(\theta) - \widehat{L}(\theta; \mathcal{D})| \leq \frac{C_r}{1-\gamma}\sqrt{\frac{\ln(2/\delta)}{2|\mathcal{D}|}}, \text{ with probability } 1-\delta$$

  where C_r is a bound on the reward function.

Resolve the Challenge: Property of the problem

- The policy optimization problem is generally non-convex
- The optimal policy $\pi_\theta^\star$ is unique under each reward function $r(\cdot, \cdot; \theta)$:

$$\pi_\theta^\star(a|s) = \frac{\exp Q_\theta(s, a)}{\sum_{\tilde{a} \in \mathcal{A}} \exp Q_\theta(s, \tilde{a})}$$

where Q_θ is the soft Q-function under the optimal policy $\pi_\theta^\star$

$$Q_\theta(s, a) := r(s, a; \theta) + \mathbb{E}_{(s', a') \sim \pi_\theta^\star} \left[r(s', a'; \theta) + \mathcal{H}(\pi_\theta^\star(\cdot|s')) \right]$$

- Need to iteratively compute π and Q to converge to optimal solutions
- But this structure will help us finding closed-form solution for gradients.

Resolve the Challenge: Property of the problem

- Taking the gradient we obtain

$$\begin{aligned}
\nabla_\theta L(\theta) :=& \mathbb{E}_{\tau^{\mathrm{E}}\sim\pi^{\mathrm{E}}}\Big[\nabla_\theta r(s_t, a_t; \theta)\Big] - \mathbb{E}_{s_0\sim\rho}\Big[\nabla_\theta V_\theta(s_0)\Big] \\
=& \mathbb{E}_{\tau^{\mathrm{E}}\sim\pi^{\mathrm{E}}}\Big[\nabla_\theta r(s_t, a_t; \theta)\Big] - \mathbb{E}_{s_0\sim\rho}\Big[\nabla_\theta \log\Big(\sum_{\tilde{a}\in\mathcal{A}} \exp Q_\theta(s_0, \tilde{a})\Big)\Big] \\
=& \mathbb{E}_{\tau^{\mathrm{E}}\sim\pi^{\mathrm{E}}}\Big[\nabla_\theta r(s_t, a_t; \theta)\Big] - \mathbb{E}_{s_0\sim\rho}\Big[\sum_{a\in\mathcal{A}} \pi_\theta(a|s_0)\nabla_\theta Q_\theta(s_0, a)\Big] \\
=& \mathbb{E}_{\tau^{\mathrm{E}}\sim\pi^{\mathrm{E}}}\Big[\nabla_\theta r(s_t, a_t; \theta)\Big] - \mathbb{E}_{\tau\sim\pi_\theta}\Big[\nabla_\theta r(s_t, a_t; \theta)\Big].
\end{aligned}$$

Where the property of the Q function is used

$$V_\theta(s) = \log\Big(\sum_{\tilde{a}\sim\mathcal{A}} \exp Q_\theta(s, \tilde{a})\Big)$$

Approximate Gradient Computation

- Replace the original loss with the surrogate loss, leverage the optimality condition, we obtain a simple gradient expression:

$$\nabla L(\theta) \approx \mathbb{E}_{\tau^{\mathrm{E}} \sim \mathcal{D}} \left[\nabla_\theta r(s, a; \theta) \right] - \mathbb{E}_{\tau \sim \pi_\theta^\star} \left[\nabla_\theta r(s, a; \theta) \right]$$

- Surprisingly simple form! No Hessian computation needed
- **Observation:** The gradient contrasts the reward obtained, by following the experts and moving away from the model's current policy π_θ
- A recent work has a derivation with a more generic loss function
- Note, the second term still depends on the optimal lower-level policy

TTSA-Type Algorithm: A single-step policy optimization

- (**Policy Optimization Step.**) Following TTSA, a single-step soft-policy iteration
- Estimate the soft-Q function $\widehat{Q}(s,a)$
- Update the policy by soft policy iteration [Cen et al 21]:

$$\pi_{k+1}(a|s) \propto \exp\left(\widehat{Q}(s,a)\right), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}.$$

- One step update; $\pi_{k+1}(a \mid s)$ will track the global optimal solution $\pi_\theta^\star(a \mid s)$

TTSA-Type Algorithm: A single-step policy optimization

- (**Reward Optimization Step.**) Recall the gradient expression:

$$\nabla L(\theta) = \mathbb{E}_{\tau^{\mathrm{E}} \sim \pi^{\mathrm{E}}} \Big[\nabla_\theta r(s_t, a_t; \theta) \Big] - \mathbb{E}_{\tau \sim \pi_\theta^\star} \Big[\nabla_\theta r(s_t, a_t; \theta) \Big].$$

Under the current policy π_{k+1}, (biased) stochastic gradient update:

$$\theta_{k+1} := \theta_k + \alpha \Big(h(\theta_k, \tau_k^{\mathrm{E}}) - h(\theta_k, \tau_k) \Big)$$

where $\tau_k^{\mathrm{E}} \sim \mathcal{D}$, $\tau_k \sim \pi_{k+1}$ and $h(\theta, \tau) := \nabla_\theta r(s_t, a_t; \theta)$.

Non-asymptotic Analysis

Theorem (Zeng, Garcia & Hong, 2022)

By choosing the stepsize $\alpha = \alpha_0 \cdot K^{-\frac{1}{2}}$, *it holds:*

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\log \pi_{k+1} - \log \pi_{\theta_k}\|_\infty\right] = \mathcal{O}(K^{-\frac{1}{2}}) + \mathcal{O}(\epsilon_{\text{app}})$$

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla L(\theta_k)\|^2\right] = \mathcal{O}(K^{-\frac{1}{2}}) + \mathcal{O}(\epsilon_{\text{app}})$$

- Under linear reward parameterization, guarantee of global optimality.

Remarks

- The above work is closely related to a line of research called imitation learning, and inverse reinforcement learning (IRL) [Ziebart et al., 2008, Ross et al., 2011, Pomerleau, 1988]
- How to better learn from expert demonstrations?
- Learning a reward function and a policy together is more generalizable than supervised learning [Ross et al., 2011]
- The above result is first non-asymptotic analysis for IRL algorithm under nonlinear reward parameterization

Application to Fine-Tuning of LLM

- In typical RLHF, SFT step learn from demonstration data using the following:

$$\min_{\pi} \ell_{\mathrm{SFT}}(\pi) := -\mathbb{E}_{\tau \in \mathcal{D}_{\mathrm{SFT}}}[\log(\pi(a \mid s))]$$

  "Clone" the behavior of the expert demonstrator

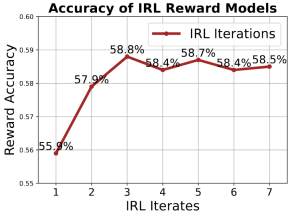
- Learn more generalizable policy? Apply (ML-BLO):

$$\min_{\theta} L_{\mathrm{SFT}} = -\mathbb{E}_{\tau \in \mathcal{D}_{\mathrm{SFT}}}[\log(\pi_\theta^\star(a \mid s))]$$
$$\text{s.t.} \quad \pi_\theta^\star = \arg\max_{\pi} \mathbb{E}_{(a,x)\sim\pi(\cdot|s)}[r(s,a;\theta) + \mathcal{H}(\pi(\cdot|s))]$$

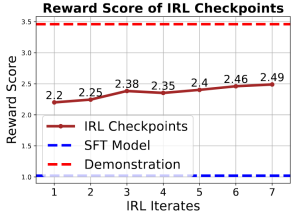
- Algorithm: Alternating between reward update (contrastive learning) and policy update (proximal policy optimization).

Numerical Results [3]

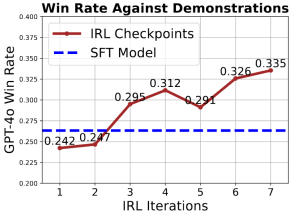
- Train LLM for text summarization tasks using the TL;DR dataset
- Initialize the model by performing SFT on a pretrained 1B parameter Pythia model
- Apply the proposed algorithm (denoted as IRL below), where the RL is done using Proximal Policy Optimization (PPO)



(a) Reward Accuracy



(b) Reward Score



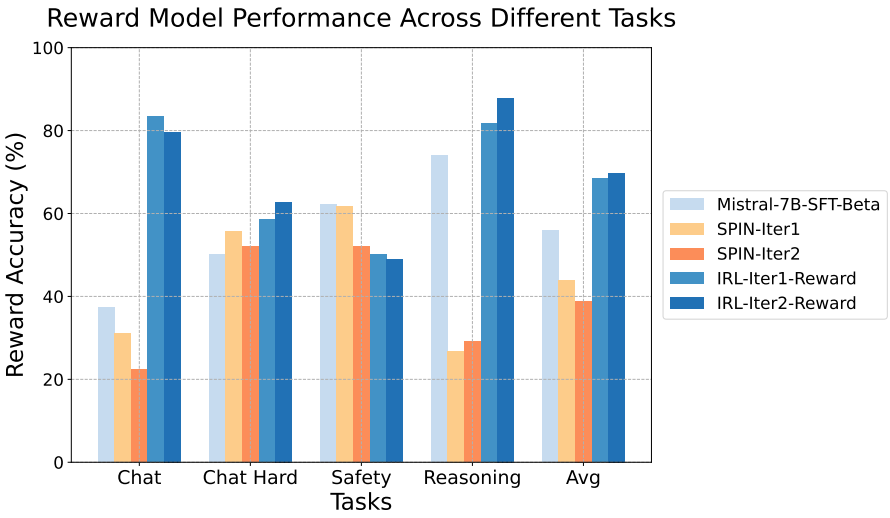
(c) Win Rate

[3] Zeng, S., Liu, Y., Rangwala, H., Karypis, G., Hong, M., & Fakoor, R. (2025). From demonstrations to rewards: Alignment without explicit human preferences.

Numerical Results

- We compare with a few SOTA methods such as SPIN [Chen et al 24] using the UltraChat dataset
- Initial policy Mistral-7b-SFT-Beta 5
- Evaluate both reward models and policy models.

Numerical Results: Reward Bench

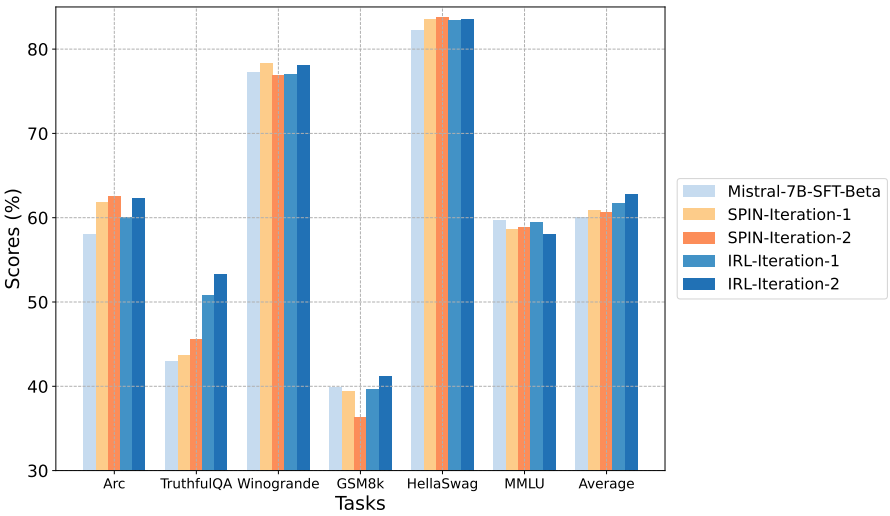


Numerical Results: MT-Bench

- Evaluate different LLM policy models using the Open LLM Leaderboard and MT-Bench

Tasks	First turn	Second turn	Average
`mistral-7b-sft-beta`	5.66	5.09	5.37
`SPIN-Iter1`	6.75	5.56	6.16
`SPIN-Iter2`	3.18	3.41	3.29
`IRL-Iter1-Policy`	6.71	5.96	6.33
`IRL-Iter2-Policy`	**7.01**	**6.19**	**6.60**

Numerical Results: OpenLLM Leaderboard



Extension

- We can further leverage this approach to integrate the three stages of RLHF (SFT, RM, RL) into a single unified stage [Li-Zeng-Li-Garcia-H. 2025] [4]

$$\begin{aligned} \min_{\theta} \quad & L_{\mathrm{SFT}}(\theta; \mathcal{D}_d) + L_{\mathrm{RM}}(\theta; \mathcal{D}_p) \\ \text{s.t.} \quad & \pi_\theta := \arg\max_{\pi} \ \mathbb{E}_{\tau\sim\pi}\left[\left(r(s,a;\theta) + \mathcal{H}(\pi(\cdot|s))\right)\right] \end{aligned}$$

- The reward and policy learning leverages all the available data
- The algorithm is similar to the two-step bilevel optimization algorithm, alternates between policy optimization and reward optimization.

[4] C Li, S Zeng, Z Liao, J Li, D Kang, A Garcia, M Hong "Learning Reward and Policy Jointly from Demonstration and Preference Improves Alignment", **ICLR (spotlight)**, 2025

Numerical Results: 1B Model for HH dataset

- Policy model: EleutherAI/pythia-1B Reward model: EleutherAI/pythia-1.4B
- Dataset: Anthropic/hh-rlhf; Evaluation: PKU-Alignment/beaver-7b-v3.0-reward

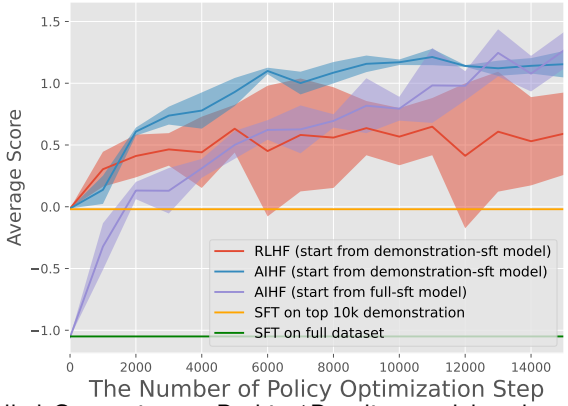


Figure: Helpfulness-controlled Generation on Pythia-1B policy models, where the reward model is trained from Pythia-1.4B models.

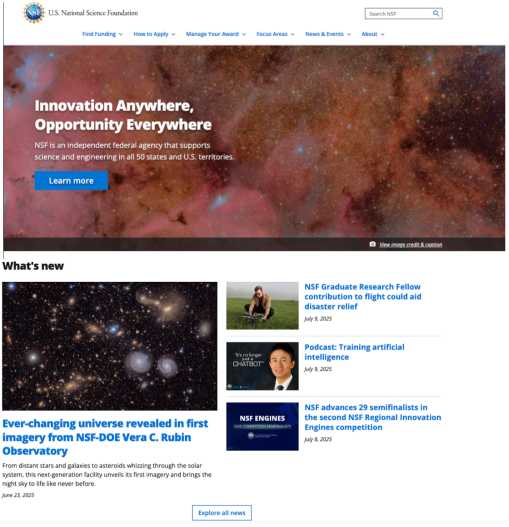
Experiment: Alignment with Demonstration & Preference

- Demonstration Dataset: UltraChat_200k
- Preference Dataset: Ultrafeedback_Binarized

Tasks	Arc Challenge	TruthfulQA MC2	Winogrande	GSM8k	HellaSwag	MMLU	Avg
mistral-7b-sft-beta	54.69%	42.96%	77.27%	39.88%	82.23%	59.72%	59.46%
zephyr-7b-beta	59.64%	55.18%	77.82%	33.51%	84.19%	59.76%	61.68%
SPIN	58.45%	43.66%	78.30%	39.50%	83.59%	58.60%	60.35%
DPO	62.80%	53.17%	**79.40%**	39.20%	85.13%	59.41%	63.19%
IPO	58.02%	48.29%	79.24%	42.91%	83.93%	60.07%	62.08%
AIHF-DPO	61.17%	**60.03%**	79.00%	39.80%	85.71%	60.02%	64.29%
Self-play AIHF	61.77%	58.29%	78.53%	**44.20%**	85.53%	58.66%	64.50%
AIHF	**63.90%**	58.38%	79.24%	40.56%	**86.23%**	**60.18%**	**64.75%**

More Experiments and Discussions

- A tutorial paper for details about applying BLO to LLM alignment [Zeng et al 25] [a]
- An NSF podcast discussing high-level ideas on how to use demonstration data to improve AI



[a] S. Zeng, L. Viano, C. Li, J. Li, V. Cevher, M. Wulfmeier, S. Ermon, A. Garcia, M. Hong, "Aligning Large Language Models with Human Feedback: Mathematical Foundations and Algorithm Design", submitted **IEEE Signal Processing Magazine**, 2025

Where Bilevel Sits in the LLM Stack

Pretraining

- Data mixing / reweighting as an (outer) bilevel objective.
- Domain weights over one giant training run.

Post-training

- RLHF / DPO / alignment: reward over policy.
- IRL: reward + policy from human data.

One unifying lens: the whole pipeline is a nested hierarchy of coupled objectives.

Pretraining: Data Mixing as Bilevel

Choose domain mixing weights λ (web/code/math/...) so the model generalizes:

$$\min_{\lambda \in \Delta} \mathcal{L}_{\text{val}}\big(w^*(\lambda)\big) \;\text{ s.t. }\; w^*(\lambda) = \arg\min_{w} \sum_{d} \lambda_d \mathcal{L}_d(w).$$

- The same data-reweighting template as Session 1, now at trillion-token scale.
- Lower-level solve = one (extremely expensive) pretraining run.
- $\Rightarrow$ demands cheap, first-order hypergradients (penalty / fully-first-order methods, e.g. F2SA [Kwon et al., 2023]): never an inverse Hessian.

Emerging: Hierarchical & Long-Horizon RL

- Hierarchical RL shares bilevel's nested, two-timescale structure: a high-level *planner* over a low-level *executor*.
- Long-horizon LLM agents (plan–execute): credit assignment *across* subgoals and *within* each segment, a leader–follower decomposition.
- Bilevel theory (two-timescale tracking, biased-gradient control) informs how to train these agents stably.
- Open: theory for multi-level (> 2) hierarchies at FM scale.

Open Problems & Frontiers

- Scalable solvers for foundation-model-scale bilevel (cheap hypergradients, no inverse Hessian).
- Theory *beyond* lower-level strong convexity: non-singleton, nonsmooth, non-convex LL [Yang et al., 2025].
- Constrained / safe alignment as bilevel (hard safety constraints on the follower).
- Finer-grained alignment: multi-turn generation, autonomous self-learning from experience.
- Benchmarks and software for bilevel at scale.

Big picture

Bilevel optimization is a vibrant area, with many open theoretical *and* practical problems.

Tutorial Summary

- Bilevel optimization = a unifying lens for hierarchical learning (leader over follower).
- **Session 1:** formulation, the implicit gradient (hypergradient), why bilevel is hard.
- **Session 2:** scalable algorithms (implicit-function / unrolling / value-function) and classical ML applications.
- **Session 3:** foundation models: alignment and inverse RL as bilevel; a Hessian-free contrastive hypergradient with finite-time guarantees; AIHF; offline IRL & reward transfer; pretraining data mixing and hierarchical frontiers.

Punchline

One template: from a toy quadratic to aligning trillion-parameter models.

Thank you!

Bilevel survey: [Zhang et al., 2024] (arXiv:2308.00788) | questions welcome.

References I

Michael Arbel and Julien Mairal. Amortized implicit differentiation for stochastic bilevel optimization. In *International Conference on Learning Representations (ICLR)*, 2022.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Jerome Bracken and James T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973.

Wilfred Candler and Roger Norton. Multi-level programming. Technical Report 20, World Bank Development Research Center, 1977.

Tianyi Chen, Yuejiao Sun, Quan Xiao, and Wotao Yin. A single-timescale method for stochastic bilevel optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

References II

Chen Fan, Parikshit Ram, and Sijia Liu. Sign-MAML: Efficient model-agnostic meta-learning by SignSGD. In *Fifth Workshop on Meta-Learning at NeurIPS*, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning (ICML)*, pages 1563–1572, 2018.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

Mingyi Hong, Hoi-To Wai, Zhaoran Wang, and Zhuoran Yang. A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic. *SIAM Journal on Optimization*, 33(1): 147–180, 2023.

Kaiyi Ji, Junjie Yang, and Yingbin Liang. Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning (ICML)*, pages 4882–4892, 2021.

References III

Prashant Khanduri, Siliang Zeng, Mingyi Hong, Hoi-To Wai, Zhaoran Wang, and Zhuoran Yang. A near-optimal algorithm for stochastic bilevel optimization via double-momentum. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Jeongyeol Kwon, Dohyun Kwon, Stephen Wright, and Robert D Nowak. A fully first-order method for stochastic bilevel optimization. In *International Conference on Machine Learning (ICML)*, pages 18083–18113, 2023.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Risheng Liu, Xuan Liu, Xiaoming Yuan, Shangzhi Zeng, and Jin Zhang. A value-function-based interior-point method for non-convex bi-level optimization. In *International Conference on Machine Learning (ICML)*, pages 6882–6892, 2021.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1540–1552, 2020.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning (ICML)*, pages 2113–2122, 2015.

References IV

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Dean A Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1988.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

References V

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning (ICML)*, pages 4334–4343, 2018.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1723–1732, 2019.

Han Shen and Tianyi Chen. On penalty-based bilevel gradient descent method. *arXiv preprint arXiv:2302.05185*, 2023.

Daouda Sow, Kaiyi Ji, Ziwei Guan, and Yingbin Liang. A constrained optimization approach to bilevel optimization with multiple inner minima. *arXiv preprint arXiv:2203.01123*, 2022.

Heinrich von Stackelberg. *Marktform und Gleichgewicht*. J. Springer, 1934.

Heinrich von Stackelberg. *The Theory of the Market Economy*. Oxford University Press, 1952.

Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations (ICLR)*, 2020.

References VI

Yan Yang, Bin Gao, and Ya-xiang Yuan. Bilevel reinforcement learning via the development of hyper-gradient without lower-level convexity. *arXiv preprint arXiv:2501.xxxxx*, 2025.

Yihua Zhang, Yuguang Yao, Parikshit Ram, Pu Zhao, Tianlong Chen, Mingyi Hong, Yanzhi Wang, and Sijia Liu. Advancing model pruning via bi-level optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

Yihua Zhang, Guanhua Zhang, Prashant Khanduri, Mingyi Hong, Shiyu Chang, and Sijia Liu. Revisiting and advancing fast adversarial training through the lens of bi-level optimization. In *International Conference on Machine Learning (ICML)*, 2022b.

Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An introduction to bilevel optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 2024. arXiv:2308.00788.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008.