FastDFL: Reusable local QP solution maps for Decision-focused learning

Anonymous authors
Paper under double-blind review

Abstract

Decision-focused learning (DFL) trains predictors through downstream optimization problems, but repeatedly solving the downstream problem and differentiating its solution make training expensive. We study DFL for strongly convex quadratic programs (QPs) with a fixed Hessian and feasible set, where predictions enter the linear term of the objective. During training, evolving predictions can revisit regions where the optimal decision is affine in the prediction and its Jacobian is constant. We propose FastDFL to turn this recurrence into reusable computation. FastDFL constructs a local formula only after the same constraint structure is observed twice, verifies its validity before reuse, and limits the cost of retrieving previously useful formulas. Whenever a formula passes the Karush–Kuhn–Tucker (KKT) validity checks, it returns the same decision and Jacobian as a fresh QP solve and differentiation. Across three benchmarks and two predictor architectures, FastDFL replaces more than half of the QP solves and derivative computations in every setting. It makes end-to-end training 1.35 to 3.47 times as fast as an otherwise identical procedure that solves each QP and differentiates its solution, while leaving decision quality nearly unchanged.

1 Introduction

Many decision-making problems follow the pattern of first predicting unknown parameters, then solving the optimization problem using those predictions. Expected returns influence portfolios, electricity prices guide storage schedules, and demand forecasts inform resource allocation (Markowitz, 1952; Donti et al., 2017; Ban & Rudin, 2019). However, minimizing a prediction metric such as mean squared error (MSE) does not necessarily minimize downstream decision costs, because errors of comparable magnitude can have substantially different operational consequences (Elmachtoub & Grigas, 2022). Decision-focused learning (DFL) addresses this mismatch by integrating the downstream optimization problem into predictor training (Wilder et al., 2019; Mandi et al., 2024). It thereby aligns the learning objective with the decisions the model is intended to support.

Nevertheless, this alignment introduces substantial computational cost. In solver-based DFL, each loss evaluation requires solving the downstream problem in the forward pass and differentiating the resulting decision for backpropagation. Revisiting a finite training set across multiple epochs repeats these optimization-layer computations many times (Amos & Kolter, 2017; Agrawal et al., 2019; Magoon et al., 2025). Accelerating this training loop requires addressing three major challenges. First, any reused computation must preserve both the decision and its derivative because both affect the model update. Second, the work required to construct, store, and retrieve reusable quantities must remain below the work ultimately saved by avoiding repeated optimization solves and differentiation. Third, fewer solver-based computations must translate into measurable end-to-end training acceleration while matching test decision quality under fair comparisons.

To address these challenges, we propose **FastDFL**. It targets strongly convex quadratic programs (QPs) in which predictions enter the linear term of the objective while the Hessian and feasible set remain fixed. Within a region of prediction space where the same constraints determine the optimum, the optimal decision is affine in the prediction, and its Jacobian is constant (Bemporad et al., 2002; Tøndel et al., 2003). On-demand reuse of such regional decision laws has also been studied in model predictive control (König & Mönnigmann, 2017). FastDFL brings this structure into the DFL training loop by constructing a formula for both the decision and its backward derivative after a constraint

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

pattern recurs, checking formulas along each sample's prediction history, and falling back to the matched solver and KKT derivative when no formula is valid.

Our contributions are:

1. We propose FastDFL, which constructs and reuses local formulas for QP decisions and prediction Jacobians when constraint structures recur during DFL training.
2. For valid formulas, we establish decision and Jacobian equivalence, bound numerical error propagation through training, and characterize solver use and formula retrieval along the prediction trajectory.
3. Across three QP benchmarks and different predictors, FastDFL preserves decision quality while achieving the lowest mean training time among the compared DFL methods.

2 Related Work

Decision-focused learning. DFL aligns prediction training with downstream optimization through differentiable optimization, decision-based objectives, and methods that reduce solver use (Kotary et al., 2021; Mandi et al., 2024; Sadana et al., 2025). Existing methods differentiate continuous relaxations (Wilder et al., 2019; Mandi & Guns, 2020) or black-box solvers (Vlastelica et al., 2020), use stochastic perturbations (Berthet et al., 2020), and optimize convex surrogates (Elmachtoub & Grigas, 2022), ranking losses (Mandi et al., 2022), directional gradients (Gupta & Huang, 2024), robust losses (Schutte et al., 2024), or task metrics (Bansal et al., 2023). Theory establishes generalization and calibration guarantees (El Balghiti et al., 2023; Liu & Grigas, 2021), while computational approaches reduce repeated solver use through solution caching (Mulamba et al., 2021), learned local losses (Shah et al., 2022), or solver-free training (Berden et al., 2025).

Differentiable optimization for QPs. OptNet differentiates QP KKT conditions (Amos & Kolter, 2017), while CVXPYLayers differentiates parametrized convex programs expressed in CVXPY (Diamond & Boyd, 2016; Agrawal et al., 2019). QP-specific methods solve a decoupled backward QP (Pan et al., 2024) or identify active constraints before reduced KKT differentiation (Magoon et al., 2025). DiffAPQP combines CVXPY canonicalization, solver updates and warm starts, and inactive constraint elimination in the backward KKT system (Xu & Teng, 2026). These layers still solve each predicted QP and compute a backward derivative at every evaluation.

Accelerating repeated QPs. QP solvers reduce work per solve through dual active-set steps (Goldfarb & Idnani, 1983), parametric active-set updates (Ferreau et al., 2014), recursive factorizations (Arnström et al., 2022), operator splitting with cached factors (Stellato et al., 2020), or proximal augmented Lagrangian updates (Bambade et al., 2025). Multiparametric QP methods represent QP families with fixed problem structure using piecewise-affine laws over polyhedral regions (Bemporad et al., 2002; Tøndel et al., 2003), supporting partial enumeration (Pannocchia et al., 2007), on-demand regional updates (König & Mönnigmann, 2017), and explicit solver generation (Arnström & Axehill, 2024; Schaller et al., 2025). Solver methods make new QPs cheaper, while regional methods can avoid a new solve by reusing an affine law within its validity region.

3 Methodology

3.1 Problem Definition

Notation. The training set contains N samples, where sample i consists of features x_i and a true QP input $a_i \in \mathbb{R}^p$. A predictor m_θ with parameters θ produces the prediction $\hat{a}_i = m_\theta(x_i)$. For any QP input a, let $z^*(a)$ denote the optimal decision. The oracle and predicted decisions are $z_i^* = z^*(a_i)$ and $\hat{z}_i = z^*(\hat{a}_i)$, respectively. For any differentiable mapping $a \mapsto v(a)$, let $J_a(v(a)) := \partial v(a)/\partial a^\top$ denote its Jacobian matrix with respect to a. In particular, $J_a(z^*(a)) \in \mathbb{R}^{n\times p}$ is the Jacobian of the optimal decision.

The strongly convex QP we focus on is given as follows:

$$z^*(a) = \arg\min_{z \in \mathbb{R}^n} \quad \frac{1}{2} z^\top P z + (q_0 + Qa)^\top z \quad \text{s.t.} \quad Az = b, \; Gz \leq h \tag{1}$$

where $P \in \mathbb{R}^{n \times n}$ satisfies $P = P^\top \succ 0$ and is the Hessian matrix of the objective function with respect to z. Positive definiteness makes the objective strongly convex and guarantees a unique optimal decision whenever the feasible set is nonempty. The coefficient of the linear term has the affine form $q(a) = q_0 + Qa$, where $q_0 \in \mathbb{R}^n$ is the fixed intercept and $Q \in \mathbb{R}^{n \times p}$ maps a to the input-dependent change in this coefficient. Let m_{eq} and m_{ineq} denote the numbers of equality and inequality constraints. The equality constraints are defined by $A \in \mathbb{R}^{m_{\text{eq}} \times n}$ and $b \in \mathbb{R}^{m_{\text{eq}}}$, while the inequality constraints are defined by $G \in \mathbb{R}^{m_{\text{ineq}} \times n}$ and $h \in \mathbb{R}^{m_{\text{ineq}}}$. Only $q(a)$ varies with the prediction, while P, A, b, G, and h remain fixed. This QP is a standard parametric QP (Ferreau et al., 2014).

Let $f_0(z; a) = \frac{1}{2} z^\top P z + q(a)^\top z$ denote the QP objective in Eq. (1). For decision evaluation, we shift this objective by the input-dependent term $\frac{1}{2} q(a)^\top P^{-1} q(a)$ and define $f(z; a) = f_0(z; a) + \frac{1}{2} q(a)^\top P^{-1} q(a)$. Completing the square gives $f(z; a) = \frac{1}{2}(z + P^{-1} q(a))^\top P (z + P^{-1} q(a))$. For a fixed input a, the added term is constant with respect to z and therefore changes neither $z^*(a)$ nor the regret difference. Because $P \succ 0$, $f(z; a)$ is nonnegative, and its value at the oracle decision is positive for every evaluated instance. The predictor is trained by minimizing the following empirical regret (Elmachtoub & Grigas, 2022):

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} [f(\hat{z}_i; a_i) - f(z_i^*; a_i)] \tag{2}$$

3.2 FastDFL Framework

FastDFL builds an on-demand library of local affine formulas that return both the QP decision and its Jacobian for recurring constraint structures. We next describe how FastDFL constructs, checks, and reuses these formulas, and how it solves and differentiates the QP when none is valid.

3.2.1 Construct and check a local formula

The following one-dimensional QP illustrates the main idea:

$$z^*(a) = \arg\min_{0 \leq z \leq 1} \left\{ \frac{1}{2} z^2 - az \right\} \tag{3}$$

The solution is $z^*(a) = 0$ for $a \leq 0$, $z^*(a) = a$ for $0 < a < 1$, and $z^*(a) = 1$ for $a \geq 1$. The lower bound binds in the first interval, no bound binds in the second, and the upper bound binds in the third. Within each interval, the decision is affine in a and its Jacobian is constant. For Eq. (1), fixing the binding constraints similarly fixes the KKT matrix and yields an affine decision (Bemporad et al., 2002).

Suppose solving the QP at a prediction $\bar{a}$ gives the decision z, the equality multipliers η, and the inequality multipliers $\lambda \geq 0$. Define the inequality slacks as $s = h - Gz \geq 0$. Constraint j binds when $s_j = 0$. The set $\mathcal{A} = \{j \mid s_j = 0\}$ contains all binding inequalities, while $\mathcal{A}_* = \{j \mid s_j > 0\}$ contains the inactive ones. FastDFL selects $\mathcal{I} \subseteq \mathcal{A}$ so that $C_{\mathcal{I}} = \begin{bmatrix} A \\ G_{\mathcal{I}} \end{bmatrix}$ has linearly independent rows. It also finds a nonnegative multiplier vector and defines $\mathcal{S} = \{j \in \mathcal{A} \mid \lambda_j > 0\}$. We write $c = (\mathcal{A}, \mathcal{I}, \mathcal{S})$. When all binding rows are independent and all binding multipliers are positive, $\mathcal{A} = \mathcal{I} = \mathcal{S}$. Appendix B.2 explains the case in which these sets differ.

For fixed c, the objective stationarity equation and the constraints indexed by $\mathcal{I}$ form the following linear system:

$$\underbrace{\begin{bmatrix} P & A^\top & G_{\mathcal{I}}^\top \\ A & 0 & 0 \\ G_{\mathcal{I}} & 0 & 0 \end{bmatrix}}_{K_{\mathcal{I}}} \underbrace{\begin{bmatrix} z \\ \eta \\ \mu_{\mathcal{I}} \end{bmatrix}}_{y_c(a)} = \underbrace{\begin{bmatrix} -q_0 \\ b \\ h_{\mathcal{I}} \end{bmatrix}}_{d_c} + \underbrace{\begin{bmatrix} -Q \\ 0 \\ 0 \end{bmatrix}}_{D_c} a \tag{4}$$

where $K_{\mathcal{I}}$ is the KKT matrix restricted to the equality constraints and the linearly independent binding inequalities. We refer to $K_{\mathcal{I}}$ as the reduced KKT matrix, following the active-set reduction used for differentiating QPs (Magoon et al., 2025). Its first block row imposes stationarity, and the next two impose the equality and independent binding constraints. The vector $y_c(a)$ contains the decision z, the equality multiplier η, and the coefficient $\mu_{\mathcal{I}}$ for the independent binding rows. The vector d_c is fixed, while $D_c a$ contains the part that varies with the prediction. The coefficient $\mu_{\mathcal{I}}$ solves the reduced linear system and need not be nonnegative. FastDFL separately obtains $\lambda_{\mathcal{S}} > 0$ from all binding rows to verify the KKT conditions. Under redundancy, $\mathcal{I}$ and $\mathcal{S}$ may therefore index different rows. Because $P \succ 0$ and $C_{\mathcal{I}}$ has linearly independent rows, $K_{\mathcal{I}}$ is nonsingular and the reduced system has a unique solution.

As only the right-hand side (RHS) of Eq. (4) changes with a, FastDFL constructs a local formula using the following two linear solves:

$$K_{\mathcal{I}} y_c(\bar{a}) = d_c + D_c \bar{a} \tag{5}$$

$$K_{\mathcal{I}} J_a(y_c(a)) = D_c \tag{6}$$

where $y_c(\bar{a})$ is the solution at the prediction $\bar{a}$, and $J_a(y_c(a)) = \partial y_c(a)/\partial a^\top$ is the Jacobian matrix at a general prediction a. Equation (5) evaluates the reduced KKT solution at $\bar{a}$. Because $K_{\mathcal{I}}$, d_c, and D_c do not depend on a, differentiating Eq. (4) with respect to a gives Eq. (6). Its kth column gives the change in $y_c(a)$ per unit change in a_k. FastDFL also differentiates stationarity on $\mathcal{S}$ to obtain $J_a(\lambda_{\mathcal{S},c}(a))$. Differentiating $s = h - Gz$ gives $J_a(s_{\mathcal{A}_*,c}(a))$. We therefore obtain the following affine formulas:

$$\begin{aligned} z_c(a) &= z_c(\bar{a}) + J_a(z_c(a))(a - \bar{a}) \\ \eta_c(a) &= \eta_c(\bar{a}) + J_a(\eta_c(a))(a - \bar{a}) \\ \lambda_{\mathcal{S},c}(a) &= \lambda_{\mathcal{S},c}(\bar{a}) + J_a(\lambda_{\mathcal{S},c}(a))(a - \bar{a}) \\ s_{\mathcal{A}_*,c}(a) &= s_{\mathcal{A}_*,c}(\bar{a}) + J_a(s_{\mathcal{A}_*,c}(a))(a - \bar{a}) \end{aligned} \tag{7}$$

where every Jacobian is constant for the given c. The local formula Φ_c consists of c, the anchor $\bar{a}$, and the anchor value and Jacobian of each quantity in Eq. (7). FastDFL stores these numerical vectors and matrices as the four affine expressions. The decision expression provides $z_c(a)$ for the forward pass and $J_a(z_c(a))$ for backpropagation. The remaining expressions verify that the formula remains valid. FastDFL stores Φ_c only after verifying the equality and stationarity equations of the full KKT system. At a new prediction a, it reuses Φ_c if $s_{\mathcal{A},c}(a) = 0$, $\lambda_{\mathcal{S},c}(a) > 0$, and $s_{\mathcal{A}_*,c}(a) > 0$. These conditions state that the same constraints remain binding, the stored multipliers remain positive, and all other constraints remain inactive. If a formula is invalid, FastDFL checks the remaining formulas and solves the QP only when none are valid.

3.2.2 Reuse local formulas during training

The formula library $\mathcal{F}$ is empty before training and contains at most one local formula for each complete binding set $\mathcal{A}$. Its formula contains the particular independent rows $\mathcal{I}$ and positive multiplier support $\mathcal{S}$ found when it was constructed. For every sample i, an ordered list $\mathcal{H}_i$ contains references to formulas in $\mathcal{F}$ that were previously reused for that sample. The most recently reused formula appears first. Thus, $\mathcal{F}$ stores the formulas, whereas $\mathcal{H}_i$ determines the order in which they are checked for sample i. For a new prediction, FastDFL checks the first formula in $\mathcal{H}_i$ and then considers older formulas in order while the following relationship holds:

$$P_t \leq (H_t + 1)\bar{\tau}_t \tag{8}$$

where P_t is the accumulated cost of checking older formulas, H_t is the number of earlier successful history searches, and $\bar{\tau}_t$ is the mean cost of solving the QP and computing its KKT derivative. The

initial budget equals one such computation. Each successful history search earns the same additional budget. The search therefore continues only while its accumulated cost is covered by observed savings.

If no stored formula is valid, FastDFL solves the QP and computes its Jacobian from the KKT conditions. This calculation also identifies $c = (\mathcal{A}, \mathcal{I}, \mathcal{S})$. The first solver result that passes the construction checks for $\mathcal{A}$ is recorded without constructing a formula. When a second solver result passes these checks for the same $\mathcal{A}$, FastDFL constructs Φ_c and stores it under $\mathcal{A}$ in $\mathcal{F}$. Delaying construction until recurrence avoids paying for a formula that may never be used. Whenever Φ_c is reused for sample i, or is constructed after solving its QP, FastDFL moves its reference to the front of $\mathcal{H}_i$. A later solver result for the same $\mathcal{A}$ may assign positive multipliers to a different subset of binding constraints. The stored formula is reused only when its own multiplier and slack conditions remain valid at the new prediction.

FastDFL also stores the latest decision and multipliers for each sample as the starting point of its next solver call. These stored values form the solver's warm state, following the reuse of primal and dual information across related QPs (Arnström et al., 2022). They affect neither formula validity nor the QP solution, yet can reduce the work of a later solve when no formula is valid. A formula evaluation also provides the decision and multipliers, allowing FastDFL to update the warm state when the solver is skipped. Algorithm 1 states these operations directly, where $\mathcal{B} \subseteq \{1, \ldots, N\}$ contains the sample indices in the current training batch, and $J_i = J_a(z^*(\hat{a}_i))$ is the Jacobian matrix used in backpropagation.

Algorithm 1 FastDFL training update

Require: current batch index set $\mathcal{B}$, predictor m_θ, formula library $\mathcal{F}$, sample histories, and solver state
for $i \in \mathcal{B}$ **do**
  Predict QP input $\hat{a}_i \leftarrow m_\theta(x_i)$
  Check the most recently used formula in $\mathcal{H}_i$, then older formulas under Eq. (8)
  if a valid formula Φ_{c_i} is found **then**
    Evaluate $(\hat{z}_i, J_i, \eta_i, \lambda_i)$ from Φ_{c_i} at $\hat{a}_i$
    Move Φ_{c_i} to the front of $\mathcal{H}_i$
  else
    Solve the QP at $\hat{a}_i$ from sample i's stored state to obtain $(\hat{z}_i, \eta_i, \lambda_i)$
    Differentiate its KKT system for J_i and identify $c_i = (\mathcal{A}_i, \mathcal{I}_i, \mathcal{S}_i)$
    if the solver result passes the construction checks for c_i **then**
      if a formula for $\mathcal{A}_i$ is already in $\mathcal{F}$ **then**
        Move the stored formula for $\mathcal{A}_i$ to the front of $\mathcal{H}_i$
      else if a previous result for $\mathcal{A}_i$ passed the construction checks **then**
        Construct Φ_{c_i} and store it under $\mathcal{A}_i$ in $\mathcal{F}$
        Move Φ_{c_i} to the front of $\mathcal{H}_i$
      else
        Record the first result for $\mathcal{A}_i$ that passes the construction checks
  Store $(\hat{z}_i, \eta_i, \lambda_i)$ as sample i's next solver state
$\mathcal{L}_\mathcal{B} \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} [f(\hat{z}_i; a_i) - f(z_i^*; a_i)]$
Update predictor parameters θ using $\nabla_\theta \mathcal{L}_\mathcal{B}$

Appendix A compares FastDFL with related QP reuse methods, and Appendix B gives two complete calculation and reuse examples.

4 Properties of FastDFL framework

This section establishes exact decision and derivative equivalence for valid formulas, then bounds numerical error when the compared computations use the same local KKT system. It also accounts for solver evaluations and formula processing along the prediction trajectory.

4.1 Precision in the forward pass and backpropagation

We first show that FastDFL preserves the decision and the derivative used in backpropagation during training. For a stored formula Φ_c with $c = (\mathcal{A}, \mathcal{I}, \mathcal{S})$, the predictions for which c remains valid are

defined as follows:

$$\mathcal{R}_c^\circ = \{a \mid s_{\mathcal{A},c}(a) = 0,\ \lambda_{\mathcal{S},c}(a) > 0,\ s_{\mathcal{A}_*,c}(a) > 0\} \tag{9}$$

where the inequalities are componentwise and the multipliers outside $\mathcal{S}$ are zero. These conditions require the constraints in $\mathcal{A}$ to remain binding, the multipliers indexed by $\mathcal{S}$ to remain positive, and all constraints in $\mathcal{A}_*$ to remain inactive. Thus, $\mathcal{R}_c^\circ$ is the exact validity region of Φ_c, and the superscript $\circ$ excludes boundary points with a zero multiplier or inactive slack. Several choices of c may be valid at the same prediction when binding constraints are redundant. Proposition 1 shows that every prediction in $\mathcal{R}_c^\circ$ produces the same decision and decision Jacobian under Φ_c as solving the QP and differentiating its KKT system. Its proof is given in Appendix C.1.

Proposition 1. *Assume that optimization problem (1) is feasible, $P \succ 0$, and c satisfies the KKT consistency conditions in Appendix B.2. For any $a \in \mathcal{R}_c^\circ$, we have $z_c(a) = z^*(a)$ and $J_a(z_c(a)) = J_a(z^*(a))$. Therefore, every differentiable decision loss $\ell(z)$ has the same gradient with respect to a under the local formula and KKT differentiation.*

Proposition 1 establishes that a valid formula preserves both quantities. We next quantify numerical deviations at a prediction where both computations identify the same binding inequalities $\mathcal{A}$ and use the same independent rows $\mathcal{I}$. Thus, both use the reduced KKT matrix $K_\mathcal{I}$. Let $y = y(a)$ and $Y = J_a(y(a))$ denote the KKT vector and its Jacobian obtained by solving and differentiating the QP. Let $\widetilde{y} = \widetilde{y}(a)$ and $\widetilde{Y} = J_a(\widetilde{y}(a))$ denote their counterparts from a formula evaluation. Their value residuals $\rho, \widetilde{\rho}$ and Jacobian residuals $\Omega, \widetilde{\Omega}$ satisfy the following systems:

$$\begin{aligned} K_\mathcal{I} y &= d_c + D_c a + \rho & K_\mathcal{I}\widetilde{y} &= d_c + D_c a + \widetilde{\rho} \\ K_\mathcal{I} Y &= D_c + \Omega & K_\mathcal{I}\widetilde{Y} &= D_c + \widetilde{\Omega} \end{aligned} \tag{10}$$

If these linear systems are solved without numerical error, all four residuals are zero. Under the shared structure above, their magnitudes and $\|K_\mathcal{I}^{-1}\|_2$ control the local decision and Jacobian differences, consistent with precision analyses of implicit differentiation (Blondel et al., 2022). Let $r \in \{0, \ldots, R-1\}$ index the R predictor updates. At training state u, let $g_r(u)$ denote the model gradient obtained by solving and differentiating the QP, and $\widetilde{g}_r(u)$ denote the gradient obtained by FastDFL. Let u_r and $\widetilde{u}_r$ contain the predictor parameters and optimizer states before update r, and let e_r satisfy $\|\widetilde{g}_r(\widetilde{u}_r) - g_r(\widetilde{u}_r)\|_2 \le e_r$. Finally, let $F_r(u)$ denote the optimizer update based on $g_r(u)$. The following Theorem 1 bounds how these per-update errors accumulate, with proof in Appendix C.2.

Theorem 1. *Assume that each formula evaluation and its matched solver evaluation identify the same binding inequalities and use the same independent rows in $K_\mathcal{I}$. Suppose the optimizer update is β_r-Lipschitz in its gradient and F_r is L_r-Lipschitz over the states being compared. If $\widetilde{u}_0 = u_0$, then after R updates we have:*

$$\|\widetilde{u}_R - u_R\|_2 \le \sum_{r=0}^{R-1} \beta_r e_r \prod_{r'=r+1}^{R-1} L_{r'} \tag{11}$$

The theorem separates the numerical deviation introduced at each update from its propagation through later optimizer updates. When the residuals vanish, FastDFL and direct KKT differentiation produce identical training trajectories. The shared-structure condition is essential because strong convexity guarantees a unique decision but not a common Jacobian across an active-set boundary (Dontchev et al., 2009).

4.2 Cumulative work along the prediction trajectory

We next bound solver use along the prediction trajectory. One evaluation occurs whenever a training sample reaches the optimization layer. FastDFL either reuses a valid formula or solves and differentiates the QP. Let N_T count the solver evaluations among the first T evaluations. In computation, FastDFL tests formula validity with the following tolerance-based admission region:

$$\mathcal{D}_c^\circ = \{a \mid \|s_{\mathcal{A},c}(a)\|_\infty < \varepsilon_b,\ \lambda_{\mathcal{S},c}(a) > \varepsilon_s,\ s_{\mathcal{A}_*,c}(a) > \varepsilon_s\} \tag{12}$$

where $\varepsilon_b, \varepsilon_s > 0$ are the binding and sign tolerances, and vector inequalities hold componentwise. This set is the numerical counterpart of $\mathcal{R}_c^\circ$ in Eq. (9).

A solver call can arise for three reasons. First, a binding set must be observed and its formula must be constructed. A solver result is a reliable observation when its decision, Jacobian, and multipliers pass the full KKT construction checks. Let $\mathcal{C}_T$ contain the distinct complete binding sets observed among the first T evaluations.

Second, a sample can lack a prior formula association or leave the region of its associated formula. Let n_T count the distinct samples among the first T evaluations. For sample i, an exit occurs when its associated formula is valid at $\hat{a}_{i,k-1}$ but invalid at the next prediction $\hat{a}_{i,k}$. Let $\mathcal{X}_T$ contain exits with resolved endpoints, and let $d_{i,k-1} > 0$ be the distance from $\hat{a}_{i,k-1}$ to the boundary of $\mathcal{D}^{\circ}_{c_i}$. The normalized prediction motion is defined as follows:

$$\mathcal{V}_T = \sum_{(i,k)\in\mathcal{X}_T} \frac{\|\hat{a}_{i,k} - \hat{a}_{i,k-1}\|_2}{d_{i,k-1}} \tag{13}$$

Every exit contributes at least one to $\mathcal{V}_T$. Third, formula reuse can miss for reasons not represented by a resolved exit. A different positive multiplier support can invalidate the stored formula, and a bounded search may not reach another valid formula. Let M_T count the remaining reliable solver evaluations after a formula for their binding set has been stored, excluding a sample's first such evaluation and resolved exits. Let U_T count unresolved evaluations whose solver results fail the full KKT construction checks. In general, neither M_T nor U_T is necessarily sublinear.

Searching older formulas also costs time. Let P_T be its cumulative cost, H_T the number of successful searches, and π_T the highest cost of one check. The following Theorem 2 bounds both solver use and search cost, with proof in Appendix C.3.

Theorem 2. *Assume a formula is stored after the same binding set passes the full KKT construction checks twice, and a formula reused for a sample is checked first at that sample's next evaluation. Then the following bounds hold:*

$$\begin{aligned} N_T &\le \min\{T, 2|\mathcal{C}_T| + n_T + \mathcal{V}_T + M_T + U_T\} \\ P_T &\le (H_T + 1) \max_{1\le t\le T} \bar{\tau}_t + \pi_T \end{aligned} \tag{14}$$

If every later reliable prediction is valid under its stored formula and that formula is tested, $M_T = 0$. The extra term is needed when a stored formula is invalid or is not reached by the bounded search. We next relate solver evaluations to optimization-layer work. Let W_T and $\bar{W}_T$ denote the cumulative optimization-layer work of FastDFL and repeated solver execution. Suppose ζ_+ bounds all formula evaluation, checking, and retrieval work per evaluation, ν_+ bounds one solver use, and B_T is cumulative formula-construction work. We then obtain:

$$W_T \le T\zeta_+ + N_T\nu_+ + B_T \tag{15}$$

If each direct QP solve and KKT derivative costs at least $\tau_- > 0$, then $\bar{W}_T \ge T\tau_-$. The W_T comparison concerns the optimization layer. Predictor and validation costs are shared by both training paths and can reduce the end-to-end speedup. Here $o(T)$ denotes a quantity that grows more slowly than T. Corollary 1 follows by combining these bounds, with proof in Appendix C.4.

Corollary 1. *If U_T, $|\mathcal{C}_T|$, n_T, $\mathcal{V}_T$, M_T, and B_T are $o(T)$, then $N_T/T \to 0$. If $\zeta_+ < \tau_-$ and $\nu_+ < \infty$, then the following asymptotic bound holds:*

$$\limsup_{T\to\infty} \frac{W_T}{\bar{W}_T} \le \frac{\zeta_+}{\tau_-} < 1 \tag{16}$$

Thus, solver use vanishes as a fraction of training only when additional formula and search misses are also sublinear. The remaining formula work is cheaper when $\zeta_+ < \tau_-$.

5 Results

5.1 Experimental setup

Benchmarks. We evaluate three public-data QP tasks: **Fama** predicts returns for constrained mean-variance portfolios (Markowitz, 1952; French, 2026), **Battery** predicts next-day electricity prices for storage dispatch (Donti et al., 2017), and **M5** predicts retail demand for budgeted replenishment (Makridakis et al., 2022). Appendix D.1 gives their formulations.

Baselines. We compare complete forward and backward paths. PO minimizes MSE without a QP layer. OSQP–KKT (Stellato et al., 2020), ProxQP–KKT (Bambade et al., 2025), and DAQP–KKT (Arnström et al., 2022) combine their solvers with KKT adjoint differentiation. OptNet (Amos & Kolter, 2017), CVXPYLayers (Agrawal et al., 2019), dQP (Magoon et al., 2025), DiffAPQP (Xu & Teng, 2026), and BPQP (Pan et al., 2024) are differentiable optimization baselines, with the latter three specialized for QPs. DAQP–KKT provides a direct comparison because FastDFL uses the same DAQP forward solve and KKT backward computation.

Metrics. We report root mean squared error (RMSE), normalized decision regret (NDR), complete training time, and computation reduction, which is the proportion of solver and derivative computations replaced by formulas. Appendix D.2 defines these metrics.

Basic settings. All methods use identical data splits across 20 seeds (42–61). The main comparison uses a linear predictor to compare optimization-layer differences. Appendix D.3 reports the predictors, hyperparameters, tolerances, and computing environment. Appendix E.4 examines the binding and sign tolerances ε_b and ε_s.

5.2 Results Analysis

Table 1 compares all methods on the three tasks under a common linear predictor, with DAQP–KKT as the matched computation without formula reuse.

Table 1: Decision quality and training time with a linear predictor

	Fama			Battery			M5		
Method	RMSE (10^{-2})	NDR (10^{-3})	Training time (s)	RMSE	NDR (10^{-3})	Training time (s)	RMSE (10^{2})	NDR	Training time (s)
PO	$4.70_{\pm<0.01}$	$3.13_{\pm0.02}$	$13.2_{\pm0.3}$	$7.42_{\pm0.10}$	$4.22_{\pm0.11}$	$21.3_{\pm0.2}$	$2.07_{\pm0.12}$	$3.80_{\pm0.56}$	$6.2_{\pm<0.1}$
OSQP–KKT	$4.99_{\pm0.04}$	$2.37_{\pm0.11}$	$54.4_{\pm0.6}$	$7.83_{\pm0.03}$	$3.14_{\pm0.04}$	$76.0_{\pm0.7}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$37.1_{\pm1.0}$
OptNet	$5.02_{\pm0.05}$	$2.42_{\pm0.09}$	$68.6_{\pm2.6}$	$7.81_{\pm0.03}$	$3.13_{\pm0.04}$	$598.9_{\pm2.7}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$229.2_{\pm1.2}$
CVXPYLayers	$4.98_{\pm0.03}$	$2.37_{\pm0.08}$	$329.1_{\pm14.0}$	$10.26_{\pm1.47}$	$3.79_{\pm0.07}$	$438.9_{\pm1.2}$	$2.14_{\pm0.01}$	$2.92_{\pm0.02}$	$219.7_{\pm0.5}$
dQP	$4.99_{\pm0.04}$	$2.37_{\pm0.11}$	$137.9_{\pm0.6}$	$7.44_{\pm0.00}$	$4.27_{\pm0.00}$	$325.1_{\pm4.0}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$194.5_{\pm1.0}$
DiffAPQP	$5.04_{\pm0.06}$	$2.40_{\pm0.12}$	$89.3_{\pm0.7}$	$7.81_{\pm0.02}$	$3.09_{\pm0.05}$	$601.3_{\pm1.8}$	$2.14_{\pm0.01}$	$2.92_{\pm0.02}$	$196.4_{\pm0.5}$
BPQP	$4.98_{\pm0.05}$	$2.40_{\pm0.10}$	$89.3_{\pm0.4}$	$7.82_{\pm0.03}$	$3.13_{\pm0.03}$	$121.0_{\pm0.7}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$147.8_{\pm0.8}$
ProxQP–KKT	$4.98_{\pm0.04}$	$2.41_{\pm0.10}$	$90.0_{\pm1.5}$	$7.82_{\pm0.04}$	$3.14_{\pm0.05}$	$122.3_{\pm1.3}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$38.3_{\pm0.3}$
DAQP–KKT	$4.99_{\pm0.04}$	$2.37_{\pm0.11}$	$35.9_{\pm0.3}$	$7.83_{\pm0.03}$	$3.14_{\pm0.04}$	$63.6_{\pm0.4}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$31.7_{\pm0.2}$
FastDFL	$4.99_{\pm0.04}$	$2.37_{\pm0.11}$	$\mathbf{24.4}_{\pm\mathbf{0.4}}$	$7.83_{\pm0.03}$	$3.14_{\pm0.04}$	$\mathbf{47.2}_{\pm\mathbf{0.5}}$	$2.15_{\pm0.01}$	$2.92_{\pm0.02}$	$\mathbf{9.15}_{\pm\mathbf{0.05}}$

FastDFL is the fastest DFL method on every task, reducing DAQP–KKT time by 25.8–71.2% and exceeding the slowest DFL baseline by over 10 times. Its RMSE and NDR match DAQP–KKT, bringing solver-based DFL closer to PO runtime without changing decision quality. Appendix E.1 confirms the acceleration with a multilayer perceptron (MLP) as a different predictor.

Figure 1 shows validation NDR at each checkpoint with 95% confidence intervals. FastDFL follows the DAQP–KKT trajectory while reaching corresponding checkpoints earlier, whereas other baselines progress at task-dependent rates.

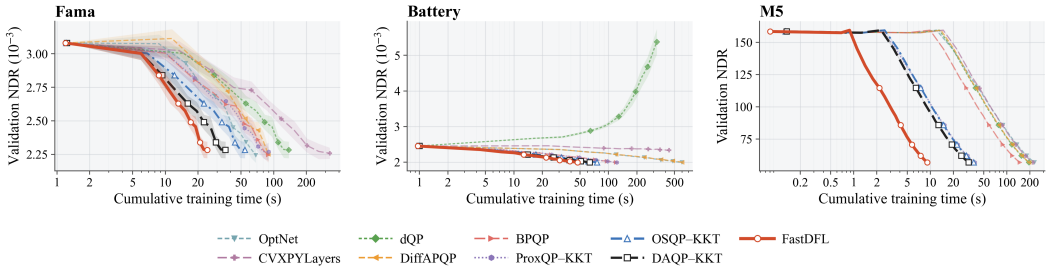

Figure 1: Validation NDR over cumulative training time

Table 2 compares formula reuse through a direct comparison with DAQP–KKT. FastDFL replaces over half of the solver and derivative computations in every setting and achieves speedups of 1.35–

3.47 times while leaving NDR nearly unchanged. Like solution caching and solver-free DFL, this result couples lower optimization cost with preserved decision quality (Mulamba et al., 2021; Berden et al., 2025). FastDFL achieves this through local reuse of matched QP outputs rather than loss approximation or replacement. Appendix E.2 compares reused decisions and Jacobians with direct computation, while Appendix E.3 extends the matched comparison to larger Battery QPs.

Table 2: Quality and computation relative to DAQP–KKT

Task	Predictor	Relative NDR difference (%)	Computation reduction (%)	Speedup (×)
Fama	Linear	$(2.17_{\pm 9.69}) \times 10^{-13}$	$65.7_{\pm 1.0}$	$1.47_{\pm 0.02}$
Fama	MLP	0	$51.5_{\pm 3.2}$	$1.39_{\pm 0.04}$
Battery	Linear	-0.0045 ± 0.0202	$55.7_{\pm 0.6}$	$1.35_{\pm 0.01}$
Battery	MLP	$(-2.43_{\pm 3.10}) \times 10^{-8}$	$55.2_{\pm 1.2}$	$1.36_{\pm 0.02}$
M5	Linear	$(3.68_{\pm 14.41}) \times 10^{-6}$	$87.2_{\pm 0.1}$	$3.47_{\pm 0.02}$
M5	MLP	$(8.69_{\pm 84.71}) \times 10^{-6}$	$75.3_{\pm 0.7}$	$1.90_{\pm 0.02}$

5.3 Ablation studies

Table 3 demonstrates FastDFL's second observation rule and budgeted history search through solver evaluations, formula constructions, search cost, and complete training time.

Table 3: Effects of formula construction and retrieval

Task	Variant	Solver evaluations	Formula constructions	Search cost (%)	Training time (s)
Fama	FastDFL	$3{,}512_{\pm 100}$	$570_{\pm 14}$	$0.4652_{\pm 0.0379}$	$\mathbf{24.38_{\pm 0.37}}$
	w/o second observation	$3{,}119_{\pm 95}$	$1{,}262_{\pm 51}$	$0.9464_{\pm 0.0560}$	$25.83_{\pm 0.49}$
	w/o history search	$3{,}989_{\pm 111}$	$570_{\pm 14}$	0	$25.05_{\pm 0.36}$
	w/o both policies	$3{,}859_{\pm 110}$	$1{,}262_{\pm 51}$	0	$26.75_{\pm 0.42}$
Battery	FastDFL	$11{,}629_{\pm 154}$	$3{,}425_{\pm 32}$	$1.5009_{\pm 0.0400}$	$\mathbf{47.19_{\pm 0.47}}$
	w/o second observation	$8{,}563_{\pm 135}$	$6{,}785_{\pm 107}$	$3.0175_{\pm 0.0662}$	$57.39_{\pm 0.88}$
	w/o history search	$15{,}159_{\pm 185}$	$3{,}428_{\pm 31}$	0	$50.53_{\pm 0.37}$
	w/o both policies	$14{,}801_{\pm 217}$	$6{,}807_{\pm 110}$	0	$64.46_{\pm 1.00}$
M5	FastDFL	$5{,}506_{\pm 57}$	$173_{\pm 3}$	$0.0032_{\pm 0.0001}$	$\mathbf{9.15_{\pm 0.05}}$
	w/o second observation	$5{,}422_{\pm 59}$	$191_{\pm 3}$	$0.0054_{\pm 0.0055}$	$9.19_{\pm 0.09}$
	w/o history search	$5{,}506_{\pm 57}$	$173_{\pm 3}$	0	$9.18_{\pm 0.05}$
	w/o both policies	$5{,}422_{\pm 60}$	$191_{\pm 3}$	0	$9.16_{\pm 0.06}$

Immediate construction reduces solver evaluations but roughly doubles formula construction on Fama and Battery without shortening training, showing that avoided calls need not justify unused formulas. Retrieving only the newest formula leaves construction nearly unchanged but misses reuse recovered by history search at a cost of at most 1.50% of training time. The need to balance formula construction against later reuse is consistent with regional MPC, which reuses an affine law within its validity region and updates it after leaving the region (König & Mönnigmann, 2017). Removing both policies combines both inefficiencies. FastDFL balances these costs by delaying construction and extending search only while observed savings cover it.

6 Discussion

FastDFL suggests that an optimization layer need not treat evaluations along a training trajectory as independent problems. When the prediction trajectory revisits a local KKT structure, the local solution map recovered from earlier solves becomes reusable computation for both subsequent decisions and their derivatives. This perspective differs from accelerating each solve in isolation or replacing optimization with a surrogate objective. It also shows that QP dimension alone does not determine the attainable benefit. Reuse is valuable only when recurrent local structures save enough solve and differentiation time to cover formula construction, evaluation, and retrieval.

These observations raise two open issues. First, formula management could estimate future savings and remove or compress formulas with low expected value under a memory budget. Second, extending reuse beyond fixed Hessians and feasible sets requires validity conditions as broader problem data vary. These questions connect the design of the optimization layer to the geometry of the training trajectory rather than treating acceleration solely as a solver problem.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 specify FastDFL, its assumptions, and its theoretical properties. Appendices B and C provide worked examples, complete construction conditions and proofs. Appendix D documents the public data sources, QP formulations, baseline implementations, experimental settings, and evaluation metrics. We also provide an anonymous supplementary material to help reproduce the results presented in this paper. The anonymous supplementary materials implement the FastDFL framework (Algorithm 1), all compared baselines with two predictors, and the experiments underlying Tables 1–3, Appendix E.1, Appendix E.3, and Appendix E.4. Its README file provides dependency requirements, public data links, and commands for reproducing the experiments on different operating systems.

AI USE STATEMENT

Large language models (LLMs) were used in the following ways. First, they assisted in identifying and summarizing prior work on reusing local solution information across QPs and differentiating through quadratic optimization layers. The authors read the cited sources, manually verified the resulting summaries and claims, and added the references themselves. Second, LLMs assisted in structuring the conditional efficiency analysis in Section 4.2, particularly in classifying solver evaluations and deriving bounds on solver use and formula retrieval work. This assistance identified and corrected an error in an initial derivation, which the authors subsequently verified. The authors checked every definition and proof. Third, LLMs assisted in coding, including dataset processing, integration of baseline implementations under common interfaces, and testing in different operating systems. The authors reviewed and tested the resulting code and verified the baseline implementations against their source papers and repositories. Finally, LLMs were used for language polishing and grammar checking after the authors drafted the initial manuscript. The authors have reviewed all AI-assisted material and take full responsibility for the manuscript, results and supplementary materials.

REFERENCES

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Daniel Arnström and Daniel Axehill. A high-performant multi-parametric quadratic programming solver. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 303–308. IEEE, 2024.

Daniel Arnström, Alberto Bemporad, and Daniel Axehill. A dual active-set solver for embedded quadratic programming using recursive $LDL^\top$ updates. *IEEE Transactions on Automatic Control*, 67(8):4362–4369, 2022.

Antoine Bambade, Fabian Schramm, Sarah El-Kazdadi, Stéphane Caron, Adrien Taylor, and Justin Carpentier. Proxqp: an efficient and versatile quadratic programming solver for real-time robotics applications and beyond. *IEEE Transactions on Robotics*, pp. 1–19, 2025.

Gah-Yi Ban and Cynthia Rudin. The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108, 2019.

Dishank Bansal, Ricky T. Q. Chen, Mustafa Mukadam, and Brandon Amos. TaskMet: Task-driven metric learning for model learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46505–46519. Curran Associates, Inc., 2023.

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

Senne Berden, Ali Mahmutoğulları, Dimos Tsouros, and Tias Guns. Solver-free decision-focused learning for linear optimization problems. In *Advances in Neural Information Processing Systems*, volume 38, pp. 158127–158145. Curran Associates, Inc., 2025.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9508–9519. Curran Associates, Inc., 2020.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-Lopez, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. In *Advances in Neural Information Processing Systems*, volume 35, pp. 5230–5242. Curran Associates, Inc., 2022.

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

Asen L Dontchev, R Tyrrell Rockafellar, and R Tyrrell Rockafellar. *Implicit functions and solution mappings: A view from variational analysis*, volume 616. Springer, 2009.

Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Othman El Balghiti, Adam N. Elmachtoub, Paul Grigas, and Ambuj Tewari. Generalization bounds in the predict-then-optimize framework. *Mathematics of Operations Research*, 48(4):2043–2065, November 2023.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1): 9–26, 2022.

Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

Kenneth R. French. 100 portfolios formed on size and book-to-market. Kenneth R. French Data Library, 2026. URL `https://mba.tuck.dartmouth.edu/pages/Faculty/ken.french/Data_Library/det_100_port_sz.html`.

Donald Goldfarb and Ashok Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33, 1983.

Vishal Gupta and Michael Huang. Decision-focused learning with directional gradients. In *Advances in Neural Information Processing Systems*, volume 37, pp. 79194–79220. Curran Associates, Inc., 2024.

Kai König and Martin Mönnigmann. Regional MPC with active set updates. *IFAC-PapersOnLine*, 50 (1):11859–11864, 2017.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pp. 4475–4482, 2021.

Heyuan Liu and Paul Grigas. Risk bounds and calibration for a smart predict-then-optimize method. In *Advances in Neural Information Processing Systems*, volume 34, pp. 22083–22094. Curran Associates, Inc., 2021.

Connor Magoon, Fengyu Yang, Noam Aigerman, and Shahar Kovalsky. Differentiation through black-box quadratic programming solvers. In *Advances in Neural Information Processing Systems*, volume 38, pp. 120073–120104, 2025.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4):1346–1364, 2022.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+ optimisation. In *Advances in Neural Information Processing Systems*, volume 33, pp. 7272–7282, 2020.

Jayanta Mandi, Vıctor Bucarey, Maxime Mulamba Ke Tchomba, and Tias Guns. Decision-focused learning: Through the lens of learning to rank. In *International Conference on Machine Learning*, pp. 14935–14947. PMLR, 2022.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, and Tias Guns. Contrastive losses and solution caching for predict-and-optimize. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pp. 2833–2840, 2021.

Brendan O'donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

Jianming Pan, Zeqi Ye, Xiao Yang, Xu Yang, Weiqing Liu, Lewen Wang, and Jiang Bian. BPQP: A differentiable convex optimization framework for efficient end-to-end learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 77468–77493, 2024.

Gabriele Pannocchia, James B Rawlings, and Stephen J Wright. Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43(5):852–860, 2007.

Utsav Sadana, Abhilash Chenreddy, Erick Delage, Alexandre Forel, Emma Frejinger, and Thibaut Vidal. A survey of contextual optimization methods for decision-making under uncertainty. *European Journal of Operational Research*, 320(2):271–289, 2025.

Maximilian Schaller, Daniel Arnström, Alberto Bemporad, and Stephen Boyd. Automatic generation of explicit quadratic programming solvers. *arXiv preprint arXiv:2506.11513*, 2025.

Noah Schutte, Krzysztof Postek, and Neil Yorke-Smith. Robust losses for decision-focused learning. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. ISBN 978-1-956792-04-1.

Sanket Shah, Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Decision-focused learning without decision-making: Learning locally optimized decision losses. In *Advances in Neural Information Processing Systems*, volume 35, pp. 1320–1332. Curran Associates, Inc., 2022.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4): 637–672, 2020.

Petter Tøndel, Tor Arne Johansen, and Alberto Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3):489–497, 2003.

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1658–1665, 2019.

Wangkun Xu and Fei Teng. Structured differentiable optimization for efficient decision-focused learning in power systems. *arXiv preprint arXiv:2608.04189*, 2026.

A Comparison with related QP reuse methods

Table 4 compares related method families by the information they retain, whether that information supports exact decision and Jacobian reuse, whether it is constructed during DFL training, and the computation required when reuse is unavailable.

Table 4: Comparison of reuse across related method families

Category	Retained information	Exact decision reuse	Exact Jacobian reuse	Constructed during DFL	When reuse is unavailable
Explicit mpQP (Bemporad et al., 2002)	Complete regional maps	✓	✓	×	Extend the parameter domain
Regional MPC (König & Mönnigmann, 2017)	Selected regional maps	✓	✓	×	Solve and add map
Differentiable QP layers (Magoon et al., 2025)	Solver state or factors	×	×	×	Solve and differentiate
Solution caching (Mulamba et al., 2021)	Cached decisions	×	×	✓	Scheduled solver call
Solver-free DFL (Berden et al., 2025)	Surrogate loss	×	×	×	Not applicable
FastDFL (Ours)	**Local decision and Jacobian formula**	✓	✓	✓	**QP solve and KKT differentiation**

B Examples and complete formula conditions

B.1 Examples

We consider two examples to analyze the proposed FastDFL framework.

Example 1. Consider the following one-dimensional problem:

$$z^*(a) = \arg\min_{0\le z\le 1} \left\{ \frac{1}{2}z^2 - az \right\} \tag{17}$$

Let L and U denote the lower and upper bounds. Their slacks are $s_L = z$ and $s_U = 1 - z$, and their multipliers are $\lambda_L \ge 0$ and $\lambda_U \ge 0$. The KKT conditions are:

$$\begin{aligned} z - a - \lambda_L + \lambda_U &= 0 \\ z &\ge 0 \\ 1 - z &\ge 0 \\ \lambda_L &\ge 0 \\ \lambda_U &\ge 0 \\ \lambda_L z &= 0 \\ \lambda_U(1 - z) &= 0 \end{aligned} \tag{18}$$

Solving these conditions, we obtain:

$$z^*(a) = \begin{cases} 0 & a \le 0 \\ a & 0 < a < 1 \\ 1 & a \ge 1 \end{cases} \tag{19}$$

The corresponding constraint descriptions are $c_L = (\{L\}, \{L\}, \{L\})$, $c_M = (\varnothing, \varnothing, \varnothing)$, and $c_U = (\{U\}, \{U\}, \{U\})$. At $\bar{a} = 0.5$, neither bound is binding, so $c = c_M$, $z_{c_M}(a) = a$, and direct differentiation gives $J_a(z_{c_M}(a)) = 1$. Hence, the stored formula is:

$$\begin{aligned} z_{c_M}(a) &= a \\ J_a(z_{c_M}(a)) &= 1 \\ s_L(a) &= a \\ s_U(a) &= 1 - a \end{aligned} \tag{20}$$

Both slacks are positive exactly when $0 < a < 1$, which is the range in which this formula can be reused.

Now consider the following five predictions encountered during training:

$$\{\hat{a}_t\}_{t=1}^{5} = (0.4, 0.5, 0.7, 1.2, 0.8) \tag{21}$$

The formula library is initially empty, and a formula is stored after its complete binding set $\mathcal{A}$ has been observed twice.

t	$\hat{a}_t$	Computation	Outcome
1	0.4	KKT conditions	First observation of $\mathcal{A} = \varnothing$; $z = 0.4$, $J_a(z) = 1$
2	0.5	KKT conditions	Second observation of $\mathcal{A} = \varnothing$; store Φ_{c_M}
3	0.7	Reuse Φ_{c_M}	$s_L = 0.7$, $s_U = 0.3$; $z = 0.7$, $J_a(z) = 1$
4	1.2	KKT conditions	Φ_{c_M} fails since $s_U = -0.2$; obtain c_U
5	0.8	Reuse Φ_{c_M}	$s_L = 0.8$, $s_U = 0.2$; $z = 0.8$, $J_a(z) = 1$

FastDFL computes the decision and Jacobian from the KKT conditions at three of the five points. At the other two points, substituting the prediction into Eq. (20) returns the same decision and Jacobian.

Example 2. Consider the following two-dimensional problem:

$$\begin{aligned} z^*(a) = \arg\min_{z \in \mathbb{R}^2} \quad & \frac{1}{2}(z_1^2 + z_2^2) + a_1 z_1 + a_2 z_2 \\ \text{s.t.} \quad & -z_1 \leq 0 \\ & -z_2 \leq 0 \\ & -z_1 - z_2 \leq 0 \end{aligned} \tag{22}$$

It should be noted that the last constraint is redundant because the first two already imply $z_1 + z_2 \geq 0$. Completing the square gives the following solution over the entire parameter space:

$$z^*(a) = \begin{bmatrix} \max(-a_1, 0) \\ \max(-a_2, 0) \end{bmatrix} \tag{23}$$

We first derive the formula in the region $a_1 > 0$ and $a_2 > 0$. At a given prediction $\bar{a} = [3, 1]^\top$, Eq. (23) gives $z^*(\bar{a}) = [0, 0]^\top$, so all three inequalities bind and $\mathcal{A} = \{1, 2, 3\}$. The binding-row matrix is:

$$G_{\mathcal{A}} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ -1 & -1 \end{bmatrix} \tag{24}$$

The third row is the sum of the first two. We choose the independent rows $\mathcal{I} = \{1, 3\}$ and let μ_1 and μ_3 denote their coefficients in the reduced KKT system. The resulting equations are:

$$\begin{aligned} z_1 + a_1 - \mu_1 - \mu_3 &= 0 \\ z_2 + a_2 - \mu_3 &= 0 \\ -z_1 &= 0 \\ -z_1 - z_2 &= 0 \end{aligned} \tag{25}$$

The last two equations give $z = [0, 0]^\top$. Substitution into the first two gives $\mu_{\mathcal{I}}(a) = [a_1 - a_2, a_2]^\top$, and differentiation gives $J_a(z) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. These reduced coefficients only solve the independent linear system, while the nonnegative multipliers satisfying the full KKT conditions may use different rows. One such multiplier vector is:

$$\lambda_{\mathcal{A}}(a) = [a_1, a_2, 0]^\top \tag{26}$$

It satisfies full KKT stationarity and is nonnegative throughout $a_1 > 0, a_2 > 0$. Its positive support is $\mathcal{S} = \{1, 2\}$. The first constraint description and its formula are therefore:

$$\begin{aligned} c^{(1)} &= (\{1, 2, 3\}, \{1, 3\}, \{1, 2\}) \\ z_{c^{(1)}}(a) &= [0, 0]^\top \\ J_a(z_{c^{(1)}}(a)) &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \lambda_{\mathcal{S}, c^{(1)}}(a) &= [a_1, a_2]^\top \end{aligned} \tag{27}$$

So we know that $\mathcal{A}$ records all binding rows, $\mathcal{I}$ supplies independent equations, and $\mathcal{S}$ supplies positive multipliers for the original KKT conditions.

The signs of a_1 and a_2 define four open regions. Table 5 lists the decision, constraint description, and Jacobian in every region. The multiplier and slack expressions that complete these four formulas are:

Table 5: The four local formulas in Example 2

Formula	Region	$z^*(a)$	$c = (\mathcal{A}, \mathcal{I}, \mathcal{S})$	$J_a(z^*(a))$
$\Phi_{c^{(1)}}$	$a_1 > 0,\ a_2 > 0$	$[0, 0]^\top$	$(\{1,2,3\}, \{1,3\}, \{1,2\})$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\Phi_{c^{(2)}}$	$a_1 < 0,\ a_2 > 0$	$[-a_1, 0]^\top$	$(\{2\}, \{2\}, \{2\})$	$\begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}$
$\Phi_{c^{(3)}}$	$a_1 > 0,\ a_2 < 0$	$[0, -a_2]^\top$	$(\{1\}, \{1\}, \{1\})$	$\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$
$\Phi_{c^{(4)}}$	$a_1 < 0,\ a_2 < 0$	$[-a_1, -a_2]^\top$	$(\varnothing, \varnothing, \varnothing)$	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$

$$
\begin{aligned}
\lambda_{\mathcal{S},c^{(1)}}(a) &= [a_1, a_2]^\top \\
\lambda_{2,c^{(2)}}(a) &= a_2 \\
s_{\{1,3\},c^{(2)}}(a) &= [-a_1, -a_1]^\top \\
\lambda_{1,c^{(3)}}(a) &= a_1 \\
s_{\{2,3\},c^{(3)}}(a) &= [-a_2, -a_2]^\top \\
s_{\{1,2,3\},c^{(4)}}(a) &= [-a_1, -a_2, -a_1 - a_2]^\top
\end{aligned}
\tag{28}
$$

On the boundaries $a_1 = 0$ or $a_2 = 0$, a multiplier or inactive slack becomes zero and the Jacobian in Eq. (23) changes. These boundary points are excluded from the four open regions and trigger a solver evaluation under the strict validity test.

Let $\mathcal{A}^{(k)}$ denote the complete binding set in $c^{(k)}$. Starting with an empty formula library, each $\mathcal{A}^{(k)}$ must be observed twice before its formula is stored. A later prediction then demonstrates reuse. Consider one sample revisited over 12 predictor updates. The following trajectory interleaves formula construction and reuse:

$$
\begin{aligned}
\{\hat{a}_t\}_{t=1}^{12} = (&[3.0, 1.0]^\top, [2.0, 1.5]^\top, [-1.0, 1.0]^\top, [-0.8, 1.2]^\top, \\
&[1.5, 0.8]^\top, [1.0, -1.0]^\top, [1.2, -0.8]^\top, [-0.7, 0.9]^\top, \\
&[-1.0, -1.0]^\top, [-0.8, -1.2]^\top, [-0.6, -0.9]^\top, [0.9, -0.7]^\top)
\end{aligned}
\tag{29}
$$

t	$\hat{a}_t$	Computation	Outcome
1	$[3.0, 1.0]^\top$	KKT conditions	First observation of $\mathcal{A}^{(1)}$
2	$[2.0, 1.5]^\top$	KKT conditions	Second observation of $\mathcal{A}^{(1)}$; store $\Phi_{c^{(1)}}$
3	$[-1.0, 1.0]^\top$	KKT conditions	First observation of $\mathcal{A}^{(2)}$
4	$[-0.8, 1.2]^\top$	KKT conditions	Second observation of $\mathcal{A}^{(2)}$; store $\Phi_{c^{(2)}}$
5	$[1.5, 0.8]^\top$	Reuse $\Phi_{c^{(1)}}$	$\lambda_{\mathcal{S}} = [1.5, 0.8]^\top$
6	$[1.0, -1.0]^\top$	KKT conditions	First observation of $\mathcal{A}^{(3)}$
7	$[1.2, -0.8]^\top$	KKT conditions	Second observation of $\mathcal{A}^{(3)}$; store $\Phi_{c^{(3)}}$
8	$[-0.7, 0.9]^\top$	Reuse $\Phi_{c^{(2)}}$	$\lambda_2 = 0.9$ and $s_{\{1,3\}} = [0.7, 0.7]^\top$
9	$[-1.0, -1.0]^\top$	KKT conditions	First observation of $\mathcal{A}^{(4)}$
10	$[-0.8, -1.2]^\top$	KKT conditions	Second observation of $\mathcal{A}^{(4)}$; store $\Phi_{c^{(4)}}$
11	$[-0.6, -0.9]^\top$	Reuse $\Phi_{c^{(4)}}$	All three inactive slacks are positive
12	$[0.9, -0.7]^\top$	Reuse $\Phi_{c^{(3)}}$	$\lambda_1 = 0.9$ and $s_{\{2,3\}} = [0.7, 0.7]^\top$

The searches at $t = 5$, $t = 8$, and $t = 12$ assume that the budget in Eq. (8) allows the required checks on older formulas. FastDFL obtains the decision and Jacobian from the KKT conditions at $t \in \{1, 2, 3, 4, 6, 7, 9, 10\}$ and reuses a stored formula at the remaining four updates. Formula construction and reuse therefore occur throughout the trajectory.

B.2 Redundant constraints and complete conditions

Let G_j denote row j of the inequality matrix G. The constraint $G_j z \leq h_j$ has the boundary $G_j z = h_j$. A displacement δ_z along this boundary satisfies $G_j \delta_z = 0$, so the column vector $G_j^\top$ is normal to the boundary. For example, $z_1 + 2z_2 \leq 3$ has the boundary $z_1 + 2z_2 = 3$. Starting from $z_0 = [1, 1]^\top$ and taking $\delta_z = [2, -1]^\top$ gives $[1, 2](z_0 + t\delta_z) = 3$ for every $t \in \mathbb{R}$. Thus, δ_z moves along the boundary and $[1, 2]^\top$ is its normal vector.

For $c = (\mathcal{A}, \mathcal{I}, \mathcal{S})$ used in Section 3.2, the rows of $C_\mathcal{I} = \begin{bmatrix} A \\ G_\mathcal{I} \end{bmatrix}$ are linearly independent. Let L_c contain the coefficients that express every discarded binding row as a linear combination of these independent rows. The same coefficients must express the corresponding right-hand sides, giving:

$$\begin{aligned} G_{\mathcal{A}\setminus\mathcal{I}} &= L_c C_\mathcal{I} \\ h_{\mathcal{A}\setminus\mathcal{I}} &= L_c \begin{bmatrix} b \\ h_\mathcal{I} \end{bmatrix} \end{aligned} \tag{30}$$

Consequently, any decision satisfying $C_\mathcal{I} z = [b^\top, h_\mathcal{I}^\top]^\top$ also satisfies every discarded binding equation because $G_{\mathcal{A}\setminus\mathcal{I}} z = L_c C_\mathcal{I} z = h_{\mathcal{A}\setminus\mathcal{I}}$. The affine multipliers further satisfy:

$$P z_c(a) + q_0 + Qa + A^\top \eta_c(a) + G_\mathcal{S}^\top \lambda_{\mathcal{S},c}(a) = 0 \tag{31}$$

where $\lambda_{\mathcal{S},c}(a) > 0$ and all other inequality multipliers are zero. Equations (30) and (31) ensure that the reduced system preserves every binding equation and satisfies stationarity for the original QP. The positive multiplier set need not be unique; every choice satisfying these conditions gives the same decision and decision Jacobian.

C Proofs

C.1 Proof of Proposition 1

Let $C_\mathcal{I} = \begin{bmatrix} A \\ G_\mathcal{I} \end{bmatrix}$ and group the multiplier coordinates into one block. The reduced KKT matrix in Eq. (4) can then be written as $K_\mathcal{I} = \begin{bmatrix} P & C_\mathcal{I}^\top \\ C_\mathcal{I} & 0 \end{bmatrix}$. We first show that this matrix is invertible. Suppose that (v, w) belongs to its null space, so we have:

$$\begin{bmatrix} P & C_\mathcal{I}^\top \\ C_\mathcal{I} & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = 0 \tag{32}$$

The two block rows give $Pv + C_\mathcal{I}^\top w = 0$ and $C_\mathcal{I} v = 0$. Multiplying the first equality by $v^\top$ and substituting the second one, we obtain:

$$0 = v^\top P v + v^\top C_\mathcal{I}^\top w = v^\top P v + (C_\mathcal{I} v)^\top w = v^\top P v \tag{33}$$

Because $P \succ 0$, the last equality implies $v = 0$. The first block row then reduces to $C_\mathcal{I}^\top w = 0$. The rows of $C_\mathcal{I}$ are linearly independent, so $C_\mathcal{I}^\top$ has a trivial null space and $w = 0$. Hence, $K_\mathcal{I}$ is invertible.

We next verify the affine KKT solution. Substituting Eqs. (5) and (6) into Eq. (7), we obtain:

$$\begin{aligned} K_\mathcal{I} \left[y_c(\bar{a}) + J_a(y_c(a))(a - \bar{a}) \right] &= K_\mathcal{I} y_c(\bar{a}) + K_\mathcal{I} J_a(y_c(a))(a - \bar{a}) \\ &= d_c + D_c \bar{a} + D_c(a - \bar{a}) \\ &= d_c + D_c a \end{aligned} \tag{34}$$

The first equality distributes $K_\mathcal{I}$, the second applies the two construction equations, and the last collects the terms containing D_c. Because $K_\mathcal{I}$ is invertible, the affine expression is the unique solution of the reduced KKT system at a.

The constraint rows of this system give $A z_c(a) = b$ and $G_\mathcal{I} z_c(a) = h_\mathcal{I}$. Equation (30) then implies $G_{\mathcal{A}\setminus\mathcal{I}} z_c(a) = h_{\mathcal{A}\setminus\mathcal{I}}$, so every inequality in $\mathcal{A}$ binds. Since $a \in \mathcal{R}_c^\circ$, every inequality in $\mathcal{A}_*$ has

positive slack. Equation (31) provides stationarity with positive multipliers on $\mathcal{S}$. Set every remaining inequality multiplier to zero. The resulting quantities satisfy the following KKT system:

$$\begin{aligned} Pz_c(a) + q_0 + Qa + A^\top \eta_c(a) + G^\top \lambda_c(a) &= 0 \\ Az_c(a) &= b \\ h - Gz_c(a) &\geq 0 \\ \lambda_c(a) &\geq 0 \\ \lambda_c(a) \odot (h - Gz_c(a)) &= 0 \end{aligned} \tag{35}$$

Together, these relations are the KKT conditions for Eq. (1). They are sufficient for global optimality because the problem is convex. Strong convexity makes the optimal decision unique, which proves $z_c(a) = z^*(a)$.

Finally, differentiating the reduced KKT equation with respect to a gives:

$$K_{\mathcal{I}} J_a(y_c(a)) = D_c \tag{36}$$

The unique solution is $J_a(y_c(a)) = K_{\mathcal{I}}^{-1} D_c$, which is exactly the Jacobian stored by Eq. (6). Selecting its decision rows proves $J_a(z_c(a)) = J_a(z^*(a))$. For any differentiable decision loss ℓ, the chain rule gives:

$$\nabla_a \ell(z_c(a)) = J_a(z_c(a))^\top \nabla_z \ell(z_c(a)) = J_a(z^*(a))^\top \nabla_z \ell(z^*(a)) \tag{37}$$

Thus, a valid local formula returns the same decision and prediction gradient as KKT differentiation.

C.2 Proof of Theorem 1

Proof. We first propagate the KKT residuals to the decision and its Jacobian under the common $K_{\mathcal{I}}$ assumed in Theorem 1. Let E_z select the decision rows from a KKT vector and define $\kappa_{\mathcal{I}} = \|E_z K_{\mathcal{I}}^{-1}\|_2$. Subtracting the first two equations in Eq. (10) and multiplying by $E_z K_{\mathcal{I}}^{-1}$ gives:

$$\widetilde{z} - z = E_z K_{\mathcal{I}}^{-1}(\widetilde{\rho} - \rho) \tag{38}$$

The induced matrix-norm inequality followed by the vector triangle inequality yields:

$$\begin{aligned} \|\widetilde{z} - z\|_2 &\leq \|E_z K_{\mathcal{I}}^{-1}\|_2 \|\widetilde{\rho} - \rho\|_2 \\ &\leq \|E_z K_{\mathcal{I}}^{-1}\|_2 \left(\|\widetilde{\rho}\|_2 + \|\rho\|_2\right) \end{aligned}$$

where the second inequality uses the triangle inequality. We denote the upper bound by ϵ_z. Let $Z = E_z Y$ and $\widetilde{Z} = E_z \widetilde{Y}$ denote the two decision Jacobians. Subtracting the last two equations in Eq. (10) gives $K_{\mathcal{I}}(\widetilde{Y} - Y) = \widetilde{\Omega} - \Omega$. Applying the same two inequalities after selecting the decision rows, we obtain:

$$\begin{aligned} \|\widetilde{Z} - Z\|_2 &\leq \|E_z K_{\mathcal{I}}^{-1}\|_2 \|\widetilde{\Omega} - \Omega\|_2 \\ &\leq \|E_z K_{\mathcal{I}}^{-1}\|_2 \left(\|\widetilde{\Omega}\|_2 + \|\Omega\|_2\right) =: \epsilon_J \end{aligned} \tag{39}$$

The final line defines the Jacobian error bound ϵ_J. We next propagate these two errors to the gradient with respect to the prediction $\hat{a}$. Let L_ℓ be a Lipschitz constant for $\nabla_z \ell$, let B_ℓ bound $\|\nabla_z \ell(z)\|_2$, and let M_J bound $\|Z\|_2$. Define $g_z = \nabla_z \ell(z)$ and $\widetilde{g}_z = \nabla_z \ell(\widetilde{z})$. The corresponding prediction gradients are $g_a = Z^\top g_z$ and $\widetilde{g}_a = \widetilde{Z}^\top \widetilde{g}_z$. Adding and subtracting $Z^\top \widetilde{g}_z$, we can obtain:

$$\widetilde{g}_a - g_a = (\widetilde{Z} - Z)^\top \widetilde{g}_z + Z^\top (\widetilde{g}_z - g_z) \tag{40}$$

The Lipschitz and bounded-gradient assumptions imply:

$$\begin{aligned} \|\widetilde{g}_z - g_z\|_2 &\leq L_\ell \|\widetilde{z} - z\|_2 \leq L_\ell \epsilon_z \\ \|\widetilde{g}_z\|_2 &\leq \|g_z\|_2 + \|\widetilde{g}_z - g_z\|_2 \leq B_\ell + L_\ell \epsilon_z \end{aligned} \tag{41}$$

Taking norms in Eq. (40), applying the triangle inequality, and using the submultiplicativity of the matrix norm, we have that:

$$\begin{aligned} \|\widetilde{g}_a - g_a\|_2 &\leq \|\widetilde{Z} - Z\|_2 \|\widetilde{g}_z\|_2 + \|Z\|_2 \|\widetilde{g}_z - g_z\|_2 \\ &\leq \epsilon_J \left(B_\ell + L_\ell \epsilon_z\right) + M_J L_\ell \epsilon_z =: \epsilon_a(a) \end{aligned} \tag{42}$$

The second inequality uses Eqs. (39) and (41), together with the bound on Z.

For each sample i in batch $\mathcal{B}_r$, Eq. (42) bounds the prediction-gradient difference by $\epsilon_a(\hat{a}_i)$. Applying the chain rule through the corresponding predictor Jacobian, we obtain the following model gradient error:

$$\|\widetilde{g}_r(\widetilde{u}_r) - g_r(\widetilde{u}_r)\|_2 \le \frac{1}{|\mathcal{B}_r|} \sum_{i \in \mathcal{B}_r} \|J_\theta(m_\theta(x_i))\|_2 \epsilon_a(\hat{a}_i) \le e_r \tag{43}$$

This is the one-step model-gradient discrepancy used in Theorem 1.

Let $\Psi_r(u, g)$ denote optimizer update r applied to state u and gradient g. By definition, $F_r(u) = \Psi_r(u, g_r(u))$. The two realized updates are $u_{r+1} = F_r(u_r)$ and $\widetilde{u}_{r+1} = \Psi_r(\widetilde{u}_r, \widetilde{g}_r(\widetilde{u}_r))$. With $d_r = \|\widetilde{u}_r - u_r\|_2$, we add and subtract $F_r(\widetilde{u}_r)$ to obtain:

$$\begin{aligned} d_{r+1} &\le \|\Psi_r(\widetilde{u}_r, \widetilde{g}_r(\widetilde{u}_r)) - \Psi_r(\widetilde{u}_r, g_r(\widetilde{u}_r))\|_2 + \|F_r(\widetilde{u}_r) - F_r(u_r)\|_2 \\ &\le \beta_r e_r + L_r d_r \end{aligned} \tag{44}$$

The final inequality applies the optimizer's gradient Lipschitz bound to the first term, Eq. (43) to its gradient difference, and the state Lipschitz bound to the second term.

It remains to unroll this recurrence. The initial condition gives $d_0 = 0$. Assume that after update r, the error d_r satisfies:

$$d_r \le \sum_{r'=0}^{r-1} \beta_{r'} e_{r'} \prod_{r''=r'+1}^{r-1} L_{r''} \tag{45}$$

Substituting this inequality into Eq. (44), we obtain:

$$\begin{aligned} d_{r+1} &\le \beta_r e_r + L_r \sum_{r'=0}^{r-1} \beta_{r'} e_{r'} \prod_{r''=r'+1}^{r-1} L_{r''} \\ &= \sum_{r'=0}^{r} \beta_{r'} e_{r'} \prod_{r''=r'+1}^{r} L_{r''} \end{aligned} \tag{46}$$

where an empty product equals one. The result follows by induction with $r = R - 1$. If all residuals vanish, then $e_r = 0$ for every update and Eq. (44) gives $d_r = 0$ throughout training. □

C.3 Proof of Theorem 2

Proof. Under the construction assumption, the first two solver results that pass the construction checks for each complete binding set account for at most $2|\mathcal{C}_T|$ solver evaluations. Partition the remaining solver evaluations that pass these checks as follows. A solver evaluation associated with a resolved exit can be charged to that exit, and each exit is charged at most once. Among the evaluations left, at most one is assigned to each sample, and every other evaluation is counted by M_T. Each unresolved evaluation is counted by U_T. These categories cover every solver evaluation, so we have:

$$N_T \le 2|\mathcal{C}_T| + n_T + |\mathcal{X}_T| + M_T + U_T \tag{47}$$

For any $(i, k) \in \mathcal{X}_T$, set $x = \hat{a}_{i,k-1}$ and $y = \hat{a}_{i,k}$. The segment from $x \in \mathcal{D}^\circ_{c_i}$ to $y \notin \mathcal{D}^\circ_{c_i}$ meets the boundary at some x'. We therefore have:

$$\|x - y\|_2 \ge \|x - x'\|_2 \ge \operatorname{dist}\left(x, \partial\mathcal{D}^\circ_{c_i}\right) = d_{i,k-1} \tag{48}$$

Thus, each exit contributes at least one to $\mathcal{V}_T$, giving $|\mathcal{X}_T| \le \mathcal{V}_T$. Substitution into Eq. (47), together with $N_T \le T$, proves the first bound.

For the search bound, let t^- be the instant before the final permitted check. Equation (8) bounds the previous search cost, and the final check costs at most π_T. We therefore obtain:

$$\begin{aligned} P_T &\le P_{t^-} + \pi_T \\ &\le (H_{t^-} + 1)\bar{\tau}_{t^-} + \pi_T \\ &\le (H_T + 1) \max_{1 \le t \le T} \bar{\tau}_t + \pi_T \end{aligned} \tag{49}$$

The result is immediate when no older formula is checked. This proves the second bound. □

C.4 Proof of Corollary 1

Proof. Dividing the solver-count bound by T, we have the following limit:

$$0 \le \frac{N_T}{T} \le \frac{U_T + 2|\mathcal{C}_T| + n_T + \mathcal{V}_T + M_T}{T} \longrightarrow 0 \tag{50}$$

because every numerator term is $o(T)$. Dividing Eq. (15) by T, we obtain:

$$\frac{W_T}{T} \le \zeta_+ + \frac{N_T}{T}\nu_+ + \frac{B_T}{T} \tag{51}$$

The last two terms vanish, so we have:

$$\limsup_{T\to\infty} \frac{W_T}{T} \le \zeta_+ \tag{52}$$

Since $\bar{W}_T \ge T\tau_-$, the following bound holds:

$$\limsup_{T\to\infty} \frac{W_T}{\bar{W}_T} \le \frac{\zeta_+}{\tau_-} < 1 \tag{53}$$

which proves Eq. (16) and completes the proof.

□

D Experimental details

D.1 Benchmark details

Table 6 gives the dimensions seen by the predictor and optimization layer. The last two columns count constraint rows before removing redundant binding rows.

Table 6: Predictor and QP dimensions (dim.)

Task	Feature dim.	Prediction dim.	Decision dim.	Hessian nonzeros	Equalities	Inequalities
Fama–French	100	100	100	10,000	1	100
Battery	81	24	96	96	48	240
M5	293	70	70	70	1	140

Fama–French portfolio (Fama). We use monthly returns for 100 portfolios formed on size and book-to-market from January 2014 through December 2023 (French, 2026). Let Σ_{FF} be their sample covariance and let a denote predicted returns for the following month. The downstream problem is the following constrained mean-variance portfolio:

$$\begin{aligned} \min_{z\in\mathbb{R}^{100}} \quad & \frac{1}{2} z^\top (5\Sigma_{\text{FF}} + 10^{-4} I) z - a^\top z \\ \text{s.t.} \quad & \mathbf{1}^\top z = 1 \\ & z \ge 0 \end{aligned} \tag{54}$$

where z_j is the portfolio weight of asset j. For each draw, a fixed generator selects two consecutive months, adds Gaussian noise with mean zero and covariance $0.5\Sigma_{\text{FF}}$ to the earlier return vector to obtain x, and uses the later return vector as a. The training, validation, and test sets contain 1,024, 256, and 2,048 disjoint draws. Month pairs can recur across these sets, and Σ_{FF} is estimated from the full 2014–2023 window. Fama therefore tests computation on a semi-synthetic collection of instances rather than chronological prediction on unseen months.

PJM battery fleet (Battery). We use the public PJM data released by Donti et al. (2017). For storage unit $k \in \{1, 2\}$ and hour $\ell \in \{1, \dots, 24\}$, let $\gamma_{k\ell}$ be signed charging power and $E_{k\ell}$ be stored energy. Positive $\gamma_{k\ell}$ charges the unit. Its energy capacity is $\bar{E}_1 = 0.75$ or $\bar{E}_2 = 1.25$, and a_ℓ

is the predicted price at hour ℓ. The stacked power and energy variables form the QP decision z. The convex dispatch model is defined as follows:

$$\begin{aligned}\min_{\gamma,E} \quad & \sum_{k=1}^{2}\sum_{\ell=1}^{24}\left(a_\ell\gamma_{k\ell}+\frac{5}{\bar{E}_k}\gamma_{k\ell}^2+\frac{2.5}{\bar{E}_k}E_{k\ell}^2-2.5E_{k\ell}\right)\\ \text{s.t.} \quad & E_{k\ell}=E_{k,\ell-1}+\gamma_{k\ell}\\ & E_{k0}=\frac{\bar{E}_k}{2}\\ & -0.5\bar{E}_k\le\gamma_{k\ell}\le 0.5\bar{E}_k\\ & 0\le E_{k\ell}\le\bar{E}_k\\ & -0.5\le\sum_{k=1}^{2}\gamma_{k\ell}\le 0.5\end{aligned} \tag{55}$$

The linear dynamics contribute 48 equalities, while variable and feeder bounds contribute 240 inequalities. Each feature vector contains the preceding day's 24 prices, load and temperature forecasts for the current day, weekday indicators, and annual sine and cosine terms. The target contains the current day's 24 prices. Complete days from 4 January 2011 to 31 December 2016 are split chronologically 60/20/20 into 1,313, 438, and 438 days. The model with signed power is a convex fleet approximation; source efficiency fields are not used in Eq. (55).

M5 replenishment (M5). We use the public M5 Forecasting Accuracy data (Makridakis et al., 2022) and aggregate the series into 70 cells defined by store and department. Let a_j be the predicted demand for the following day, and let α_j be the cell's mean selling price divided by the mean across cells. The total replenishment capacity is $C_{\text{M5}} = 26{,}069.6$. We scale physical demand by $\sigma_{\text{M5}} = C_{\text{M5}}/70 = 372.423$, so one decision unit represents σ_{M5} physical units. We set $\bar{z}_j$ to the larger of one and the cell's training 99th demand percentile after division by σ_{M5}. The downstream allocation QP is defined as follows:

$$\begin{aligned}\min_{z\in\mathbb{R}^{70}} \quad & \frac{1}{2}\sum_{j=1}^{70}(\alpha_j+0.1)z_j^2-\sum_{j=1}^{70}\frac{\alpha_j a_j}{\sigma_{\text{M5}}}z_j\\ \text{s.t.} \quad & \sum_{j=1}^{70}z_j=70\\ & 0\le z_j\le\bar{z}_j\end{aligned} \tag{56}$$

The physical shipment to cell j is $\sigma_{\text{M5}}z_j$. Features include demand at lag 1, demand means over 7 and 28 days, daily prices, weekday indicators, annual calendar terms, an event flag, and three SNAP indicators. After an initial period of 28 days, observations from 26 February 2011 through 22 May 2016 are split chronologically 75/10/15 into 1,434, 192, and 287 days.

D.2 Metric details

The root mean squared error (RMSE) averages the squared prediction error over all test samples and prediction dimensions and is defined as follows:

$$\text{RMSE}=\sqrt{\frac{1}{pN_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\|\hat{a}_i-a_i\|_2^2} \tag{57}$$

where N_{test} is the number of test samples, p is the prediction dimension, and $\hat{a}_i$ and a_i are the predicted and true QP inputs for sample i.

The normalized decision regret (NDR) measures downstream decision quality and can be calculated as follows:

$$\text{NDR}=\frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\frac{f(\hat{z}_i;a_i)-f(z_i^*;a_i)}{f(z_i^*;a_i)} \tag{58}$$

where both objective values use the true input a_i and the evaluation objective f defined in Section 3.1. The unshifted oracle objective can be negative, as the feasible zero-power Battery schedule already has value -30. The added term $q(a_i)^\top P^{-1} q(a_i)/2$ makes the objective nonnegative by completing the square, and the oracle value is strictly positive for every evaluated instance. It does not alter the regret numerator or the training gradient. For a fixed instance, NDR increases strictly with regret because its positive denominator is independent of the predicted decision. Across different instances, averaging these differently normalized regrets need not preserve the ordering of mean unnormalized regret. The models are trained and their checkpoints selected using regret, while NDR is reported for evaluation.

Beyond prediction and decision quality, we compare the computational cost of a complete training run. Following recent studies of accelerated DFL (Berden et al., 2025; Xu & Teng, 2026), we report Training time as the total elapsed wall-clock time over all training epochs, beginning immediately before the initial validation checkpoint and ending immediately after the validation checkpoint following the final epoch. It includes predictor computation, the forward and backward computations of the optimization layer, loss evaluation, model backpropagation, parameter updates, and validation at each checkpoint. The tables report their mean and sample standard deviation across the 20 seeds.

To connect training time to formula reuse, we further measure changes in decision quality, avoided solver computations, and resulting acceleration. Prior DFL caching and ranking methods similarly relate computational effort to decision quality and training time (Mulamba et al., 2021; Mandi et al., 2022). We use Relative NDR difference to measure the change in decision quality from DAQP–KKT to FastDFL. It is the NDR of FastDFL minus that of DAQP–KKT, divided by the absolute NDR of DAQP–KKT and expressed as a percentage; a positive value indicates a higher NDR for FastDFL. Computation reduction is the percentage of optimization layer evaluations in which a stored formula replaces both the QP solve and KKT differentiation. Speedup is calculated within each seed as the training time of DAQP–KKT divided by that of FastDFL; a value above one indicates faster training.

To identify where computation is saved and spent within FastDFL, we record three further quantities. Solver evaluations count the optimization layer evaluations that invoke the QP solver and KKT differentiation. Formula constructions count the local formulas constructed during training. Search cost is the percentage of training time spent checking older stored formulas after the newest formula associated with a sample is rejected.

D.3 Implementation details

All methods use the same data split, predictor initialization, and fixed training schedule within each task and seed. Every method completes the number of epochs in Table 7, and no early stopping is applied. After training, PO selects the checkpoint with the lowest validation RMSE, while each DFL method selects the checkpoint with the lowest validation regret.

Table 7 reports the optimizer settings. The learning rate is shared by all DFL methods for each task and predictor. The same learning rate is used for PO and DFL with the MLP. The MLP contains two hidden layers of 128 units with Gaussian Error Linear Unit (GELU) activations. Its hidden layers use the default PyTorch initialization, and its output layer starts at zero. The linear predictor uses ridge initialization on Fama and Battery and zero initialization on M5. Within each seed, all compared methods start from identical predictor parameters.

Table 7: Fixed training settings for the three tasks

					Learning rate		
Task	Epochs	Batch size	Optimizer	Gradient norm limit	Linear PO	Linear DFL	MLP
Fama	10	64	Adam	100	10^{-3}	10^{-3}	10^{-3}
Battery	20	64	Adam	100	10^{-3}	3×10^{-4}	3×10^{-4}
M5	30	64	Adam	100	10^{-3}	10^{-4}	3×10^{-4}

Adam uses 0 weight decay, and the joint Euclidean norm of the predictor gradients is limited to 100 before each update. The absolute and relative solver tolerances are both 10^{-7} when the implementation exposes separate tolerances. CVXPYLayers uses the splitting conic solver (SCS) with a stopping tolerance of 10^{-6}, which controls the required accuracy of its primal residual, dual

residual, and duality gap (O'donoghue et al., 2016). FastDFL uses $\varepsilon_b = 10^{-6}$ for binding constraints and $\varepsilon_s = 10^{-7}$ for multiplier and slack signs. It constructs a formula after the same complete binding set $\mathcal{A}$ passes the construction checks twice and examines older formulas according to Eq. (8). All experiments run on a Dell computer running 64-bit Windows 11 (enterprise version), with an Intel Core i7-10700 CPU (2.90 GHz), 8 physical cores and 16 logical processors and 32 GB memory. The software environment uses Python 3.11.16. All reported timing runs execute sequentially with numerical library thread counts fixed at 1, so no two training runs share computational resources.

Table 8 summarizes the forward and backward computations included in the training time of each method. Let E_z select the decision coordinates from the reduced KKT vector, and let $\nabla_z \ell$ be the incoming gradient of a scalar decision loss. For OSQP–KKT, ProxQP–KKT, and DAQP–KKT, the backward pass computes the vector–Jacobian product using the following adjoint solve:

$$\begin{aligned} K_{\mathcal{I}}^{\top} v &= E_z^{\top} \nabla_z \ell \\ \nabla_a \ell &= D_c^{\top} v \end{aligned} \tag{59}$$

The same adjoint computation is used on FastDFL solver evaluations, while formula reuse computes the vector–Jacobian product from the stored decision Jacobian. Thus, the matched comparison does not charge the direct path for explicitly constructing a full Jacobian at every prediction.

Table 8: Forward and backward computations of the DFL methods

Method	Forward computation	Backward computation
OSQP–KKT	OSQP solve	Reduced KKT adjoint solve
OptNet	qpth interior point solve	qpth implicit derivative
CVXPYLayers	Canonicalization and SCS solve	diffcp derivative
dQP	OSQP solve	Reduced KKT, active set identified
DiffAPQP	OSQP with updates and warm starts	Reduced KKT, inactive rows removed
BPQP	OSQP solve	OSQP backward QP
ProxQP–KKT	ProxQP solve	Reduced KKT adjoint solve
DAQP–KKT	DAQP solve	Reduced KKT adjoint solve
FastDFL	Stored formula or DAQP solve	Stored Jacobian product or KKT adjoint solve

E Additional Results

This appendix evaluates FastDFL with an MLP predictor, compares reused decisions and Jacobians with direct computation, measures computational cost and memory as the QP grows, and tests sensitivity to the formula validity tolerances.

E.1 Comparison across predictor architectures

The main comparison uses a linear predictor to compare differences among optimization layers. Table 9 repeats the complete DFL comparison using the MLP with two hidden layers under the same settings.

Table 9: Decision quality and training time with an MLP predictor

	Fama			Battery			M5		
Method	RMSE (10^{-2})	NDR (10^{-3})	Training time (s)	RMSE	NDR (10^{-3})	Training time (s)	RMSE (10^{2})	NDR	Training time (s)
OSQP–KKT	$6.68_{\pm 0.01}$	$8.18_{\pm 0.23}$	$51.4_{\pm 1.7}$	$15.45_{\pm 0.04}$	$1.46_{\pm 0.03}$	$49.4_{\pm 0.9}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$37.9_{\pm 0.3}$
OptNet	$6.70_{\pm 0.02}$	$7.95_{\pm 0.30}$	$48.0_{\pm 2.2}$	$16.21_{\pm 0.16}$	$1.36_{\pm 0.02}$	$585.2_{\pm 1.7}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$229.3_{\pm 0.8}$
CVXPYLayers	$6.68_{\pm 0.01}$	$8.19_{\pm 0.23}$	$341.4_{\pm 18.8}$	$220.65_{\pm 35.63}$	$2.12_{\pm 0.16}$	$441.8_{\pm 2.5}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$221.3_{\pm 1.3}$
dQP	$6.68_{\pm 0.01}$	$8.21_{\pm 0.27}$	$119.9_{\pm 2.4}$	$20.89_{\pm 4.40}$	$2.32_{\pm 0.11}$	$532.1_{\pm 16.4}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$197.1_{\pm 0.5}$
DiffAPQP	$6.70_{\pm 0.02}$	$7.93_{\pm 0.25}$	$98.6_{\pm 4.4}$	$15.61_{\pm 0.17}$	$1.43_{\pm 0.10}$	$570.6_{\pm 4.5}$	$2.56_{\pm 0.13}$	$2.80_{\pm 0.24}$	$198.1_{\pm 0.5}$
BPQP	$6.68_{\pm 0.01}$	$8.17_{\pm 0.21}$	$83.9_{\pm 2.6}$	$15.46_{\pm 0.05}$	$1.46_{\pm 0.02}$	$93.4_{\pm 1.0}$	$2.56_{\pm 0.13}$	$2.80_{\pm 0.24}$	$121.1_{\pm 0.7}$
ProxQP–KKT	$6.68_{\pm 0.01}$	$8.16_{\pm 0.22}$	$59.0_{\pm 2.0}$	$15.45_{\pm 0.04}$	$1.46_{\pm 0.04}$	$65.8_{\pm 2.8}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$39.9_{\pm 1.2}$
DAQP–KKT	$6.68_{\pm 0.01}$	$8.18_{\pm 0.23}$	$31.3_{\pm 0.7}$	$15.45_{\pm 0.04}$	$1.46_{\pm 0.03}$	$39.4_{\pm 0.3}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$33.1_{\pm 0.2}$
FastDFL	$6.68_{\pm 0.01}$	$8.18_{\pm 0.23}$	$\mathbf{22.6_{\pm 0.8}}$	$15.45_{\pm 0.04}$	$1.46_{\pm 0.03}$	$\mathbf{29.0_{\pm 0.5}}$	$2.56_{\pm 0.13}$	$2.79_{\pm 0.25}$	$\mathbf{17.4_{\pm 0.2}}$

FastDFL remains the fastest DFL method on every task with the MLP and matches DAQP–KKT in RMSE and NDR at the displayed precision, showing that its acceleration extends beyond the linear predictor.

Figure 2 compares FastDFL with every DFL baseline under both predictors. For each task, method, and predictor, training speedup is calculated within each of the 20 seeds as the baseline training time divided by the FastDFL training time. Each point reports the mean of these paired ratios, and each connecting line links the two predictor results for the same baseline.

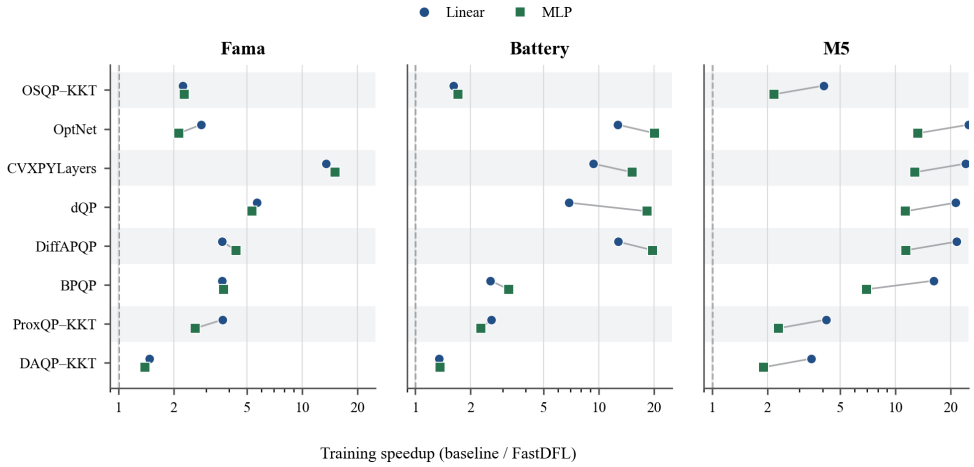

Figure 2: Training speedup over DFL baselines under linear and MLP predictors

FastDFL remains the fastest DFL method under both predictor architectures, showing that its benefit is not specific to a linear predictor. Its matched speedup changes little on Fama and Battery, while on M5 it decreases from 3.47 to 1.90 times as computation reduction falls from 87.2% to 75.3%. Because each predictor generates a different sequence of objective coefficients, the attainable acceleration depends on the recurrence of reusable constraint structures rather than the predictor architecture alone.

E.2 Numerical agreement of reused decisions and Jacobians

Proposition 1 states that a valid formula returns the same decision and Jacobian as direct KKT differentiation when both computations identify the same binding inequalities and use the same independent constraint rows. We test this agreement at every successful formula use in the linear predictor experiments. At each such prediction a, let $\widetilde{z} = z_c(a)$ and $\widetilde{J} = J_a(z_c(a))$ denote the outputs of formula evaluation. Let $z = z^*(a)$ and $J = J_a(z^*(a))$ denote the results of an independent DAQP–KKT computation. Their normalized differences are defined as follows:

$$\begin{aligned} \mathrm{err}_z &= \frac{\|\widetilde{z} - z\|_2}{1 + \|z\|_2} \\ \mathrm{err}_J &= \frac{\|\widetilde{J} - J\|_{\mathrm{F}}}{1 + \|J\|_{\mathrm{F}}} \end{aligned} \tag{60}$$

where $\|\cdot\|_2$ is the 2-norm and $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. The constant in each denominator keeps the measure well defined when the reference norm is zero. As the norm increases, the normalization approaches a relative error. The independent computation is used only for this comparison, so it neither affects training nor contributes to the reported training time.

Table 10 reports the largest pointwise errors, and Figure 3 summarizes typical agreement through seed-level means. Each point in the figure averages all successful formula uses in one run.

Across 1,177,471 successful formula uses, the mean decision error of each seed lies near numerical precision, and most mean Jacobian errors follow the same pattern. The maximum pointwise decision error remains 5.51×10^{-8}. Eight individual formula uses have $\mathrm{err}_J > 10^{-10}$. They correspond

Table 10: Maximum errors across successful formula uses

Task	Formula uses	Maximum err_z	Maximum err_J
Fama	134,562	1.68×10^{-14}	0
Battery	292,626	2.79×10^{-8}	9.27×10^{-2}
M5	750,283	5.51×10^{-8}	2.46×10^{-3}

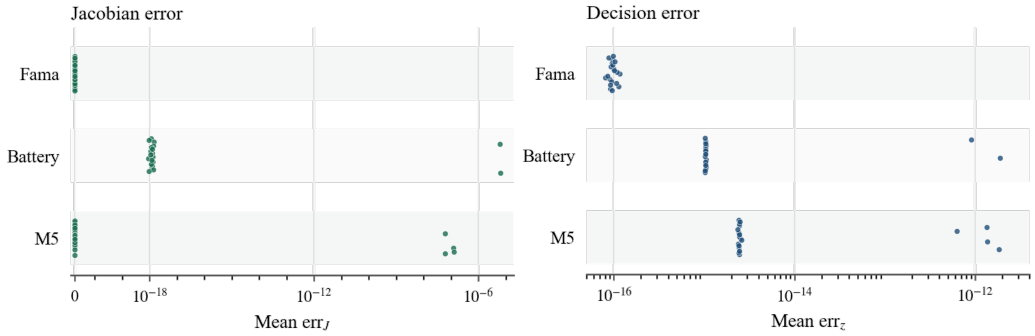

Figure 3: Seed-level mean decision and Jacobian errors of formula reuse

to six elevated seed-level means in the figure. Thus, formula reuse agrees with direct computation up to numerical precision under the shared active set condition, while the only material Jacobian differences occur at rare degenerate boundaries.

E.3 Computational cost and memory scaling

We next examine computational cost and memory as the downstream QP grows. We extend the coupled Battery QP to two, three, and four storage units while keeping the prediction dimension at 24 and using the linear predictor. Let n denote the number of decision variables, and let $m = m_{\mathrm{eq}} + m_{\mathrm{ineq}}$ denote the total number of constraints. The resulting scales are $(n, m) \in \{(96, 288), (144, 408), (192, 528)\}$.

DAQP–KKT and FastDFL train for 40 epochs, and Figure 4 traces the accumulation of computational savings during training. Formula construction initially offsets much of the saved solver work. By epoch 5, mean speedup exceeds one at every scale as computation reduction reaches approximately 25%. The advantage then grows with recurring constraint structures. At epoch 40, FastDFL replaces over 2/3 of solver and derivative computations and remains faster at every tested scale.

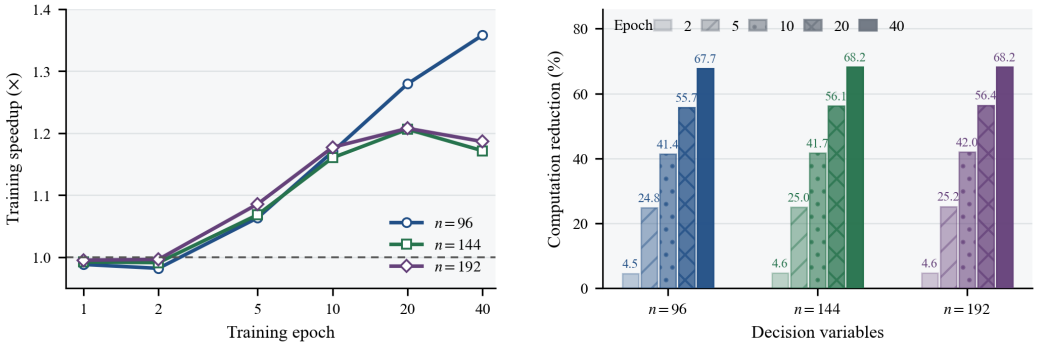

Figure 4: Training speedup and computation reduction across QP scales

Figure 4 shows when formula reuse becomes beneficial, while Table 11 reports its final time and memory costs. Formula processing includes checking and constructing formulas and synchronizing the solver state. Formula library memory sums the stored numerical arrays, while peak memory is

the maximum working set of the complete process. The training time and peak memory columns list the DAQP–KKT and FastDFL measurements in that order.

Table 11: Training cost and memory across QP scales

QP size (n/m)	Training time (s)	Speedup $(\times)$	Computation reduction (%)	Formula processing (s)	Formula library (MB)	Peak memory (MB)
96/288	$122.3_{\pm 0.8}/90.1_{\pm 1.1}$	$1.36_{\pm 0.01}$	$67.68_{\pm 0.48}$	$10.34_{\pm 0.15}$	$394.1_{\pm 3.7}$	$651.5_{\pm 8.2}/1083.0_{\pm 7.3}$
144/408	$169.3_{\pm 0.8}/144.5_{\pm 1.3}$	$1.17_{\pm 0.01}$	$68.16_{\pm 0.38}$	$14.27_{\pm 0.19}$	$564.0_{\pm 5.2}$	$834.8_{\pm 7.6}/1432.2_{\pm 21.5}$
192/528	$235.9_{\pm 0.8}/198.8_{\pm 2.4}$	$1.19_{\pm 0.01}$	$68.23_{\pm 0.40}$	$18.13_{\pm 0.30}$	$733.1_{\pm 5.9}$	$942.3_{\pm 4.5}/1739.2_{\pm 14.8}$

Computation reduction remains near 68% at all three scales, but speedup is not monotonic, showing that avoided solver work and formula overhead grow at different rates. FastDFL nevertheless lowers complete training time at every scale. Formula processing rises from 10.34 to 18.13 seconds, and formula library memory grows from 394.1 to 733.1 MB, raising peak memory to at most 1.74 GB. Formula reuse therefore preserves a time advantage as the QP grows, at the cost of increasing memory. Mean relative NDR differences remain below 0.057%, showing that this computational tradeoff does not change decision quality.

E.4 Sensitivity to formula validity tolerances

We finally test sensitivity to the two formula validity tolerances. The binding tolerance ε_b limits the residual on binding constraints, while the sign tolerance ε_s sets the positive margin for multipliers and inactive slacks. Using the linear predictor, we vary one tolerance at a time over $\varepsilon_b \in \{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$ and $\varepsilon_s \in \{10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}\}$ while holding the other at its default. Figure 5 reports the mean paired change in computation reduction, where the vertical dotted line marks the default.

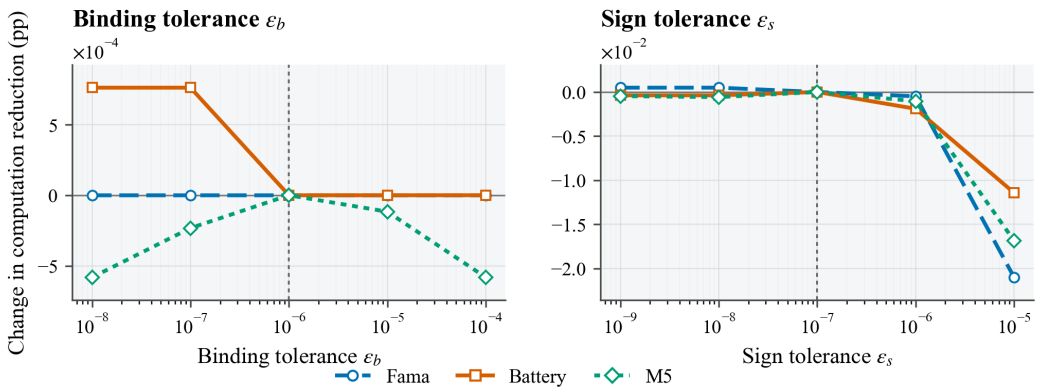

Figure 5: Changes in computation reduction under the formula validity tolerances

Across the four orders of magnitude tested, changing ε_b alters mean computation reduction by at most 0.0008 percentage points (pp), while the mean absolute relative NDR difference remains below $7 \times 10^{-8}\%$. Changing ε_s alters computation reduction by at most 0.0210 pp and yields a maximum mean absolute relative NDR difference of 0.0046%. Only the largest sign tolerance produces a visible decline in computation reduction because its stricter margin rejects more predictions near constraint boundaries. Both decision quality and formula reuse therefore remain stable throughout the tested tolerance ranges.