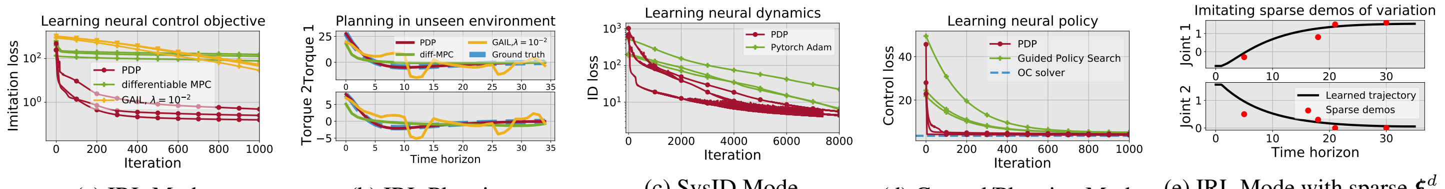
Response to Rev. #1: **(A)** We have performed comparison with other end-to-end learning frameworks in Appendix F, Page 11, including comparison with 'differentiable MPC'. In this response, we also provide additional comparison with 'differentiable MPC' in Fig. 1. In revision, we will move all comparisons to the primary text. **(B)** We try to unify the fundamental IRL, OC, SysID problems, as they can be viewed the same problem with different locations of unknown parameters. Background currently reviews the most related control/learning methods for these problems. In revision, we will discuss other related topics like Bayesian models, MDP, and MPC. **(C)** 'Control/Planning mode' formulates a typical OC problem, which can be solved by iLQR, while PDP is *a new OC method* and empirically shows more efficiency than iLQR. MPC is rather a strategy of solving OC with shifting horizons, and its implementation still requires OC solvers. **(E)** the neural policy is to imitate a policy, while inverse KKT learns a cost function. With limited data (see Appendix E), the former can be over-fitting and the latter generally has *better generality to unseen conditions*, as a cost function is a high-level representation of policies.

Response to Rev. #2: **(A)** PDP allows the use of (deep) neural networks to represent unknowns of a system. As supportive examples, we use neural networks to re-do experiments of IRL, SysID, and OC in Fig. 1 (will be added to revision). **(B)** PDP uses gradient descent to solve non-convex bi-level optimization, and only local optima are achieved. But if we make assumptions, e.g., convexity/smoothness, on inner OC and outer loss, we could have global convergence by bi-level programming theory [S. Ghadimi, et al, Approximation Methods for Bilevel Programming. arXiv:1802.02246]. However, such conditions are too restrictive to inner OC. In the future, we will investigate mild conditions using control (e.g., Lyapunov) theory. **(C)** Empirically, parameterization matters in its convergence. E.g., neural network introduces high non-linearity and is more likely to trap in local optima than polynomial; thus simply parameterization is preferred. **(D)** A failure source of PDP is catastrophic error of OC solver – causing error of trajectory gradient. Please see Appendix H for other discussions. **(E)** All codes will be released.

Response to Rev. #3: **(A)** We will mathematically declare Equ (2) as argmin of (1) in place of language exposition (here we followed exposition conventions in control literature). **(B)** The reviewer makes an interesting comment about the needs of having intermediate Jacobian $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$. Before explaining why, we argue that PDP is also capable of directly computing $\frac{\partial l}{\partial \boldsymbol{\theta}}$ without needing $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$. Consider Equ (13) (stack for all t) as a big linear equation for $\frac{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}}{\partial \boldsymbol{\theta}}$, $\frac{\partial \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T}}{\partial \boldsymbol{\theta}}$, and $\frac{\partial \boldsymbol{\lambda}^{\boldsymbol{\theta}}_{1:T}}{\partial \boldsymbol{\theta}}$. By multiplying the inverse of coefficient matrix of this equation to the backward pass vector $\frac{\partial l}{\partial \boldsymbol{\xi}}$ from loss, we obtain the analytical gradient $\frac{\partial l}{\partial \boldsymbol{\theta}} = \frac{\partial l}{\partial \boldsymbol{\xi}}\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ without explicit $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$, similar to Equ (7) in OptNet (Equ (9) in differentiable MPC) and Neural ODE. However, we have not adopted this way, because it *causes huge computation and memory complexity*: the size of this linear equation (Equ (13)) is $(m{+}n)T \times (m{+}n)T$ (T is the time horizon, usually very large, m, n are state, input dims, also see Equ (6) in differentiable MPC); and computing the inverse of such a big matrix costs at least a complexity of $(m{+}n)^2T^2$ (e.g., in our UAV experiment, it requires to solve a 1000*1000-sized linear equation). Due to these inefficiency, PDP proposes to explicitly solve the intermediate Jacobian $\frac{\partial \boldsymbol{\xi}}{\partial \boldsymbol{\theta}}$ using auxiliary control system, where the memory is only $2(m{+}n)T$ and computation is also $2(m{+}n)T$ thanks to recursion structure. Aside benefit: a side-product of solving $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ is that *gradient of individual trajectory point* $\frac{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_t}{\partial \boldsymbol{\theta}}$ is obtained. It enables to *learn from sparse data* as shown in Fig. 1e : in IRL, given sparse demo $\boldsymbol{x}^d_t$ only at time t, PDP can use $\frac{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_t}{\partial \boldsymbol{\theta}}$ to $\min_{\boldsymbol{\theta}} \|\boldsymbol{x}^{\boldsymbol{\theta}}_t - \boldsymbol{x}^d_t\|^2$. **(C)** In Appendix F, Page 11, we have stated the above advantage and reported the timing of PDP in Fig. 7 compared to OptNet (differentiable MPC): *PDP is much faster than OptNet*. **(D)** The revision will cite all mentioned papers, improve Sections 2, 3, and address the minor issues. **(E)** For reproducibility, please find experiment/algorithm details in Appendix D/E. Codes will be released.

Response to Rev. #5: **(A)** If both *dynamics* and *feedback policy* are parameterized, PDP still applies. But we shall be cautious here: if we set the loss as imitation loss as in Equ (6), it is fine but more desirable to learn each separately by supervised learning; if we define the loss as a control cost like Equ (8), very likely we will arrive at trivial (zero) models since parameterizing both is redundant. **(B)** Equ (4) states: $\min_{\boldsymbol{\theta}} \|\boldsymbol{\xi}_{\boldsymbol{\theta}} - \boldsymbol{\xi}^d\|$ s.t. $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is the optimal trajectory of OC system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in Equ (1). It says that we want to find a cost function $J(\boldsymbol{\theta})$ in the OC system such that its produced optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is closest to the demonstration $\boldsymbol{\xi}^d$. **(C)** The end-to-end concept in our context refers to a way to look at PDP framework as a whole neural network (representing a control system). Then PDP trains this neural network by directly minimizing the loss over the parameter of interest using gradient descent. However, unlike generic NNs, we inject the inductive bias to this PDP NN using optimal control theory: the trajectory of OC satisfies PMP, which makes the PDP NN more efficient to solve control/learning problems. **(D)** Explainability is relative to the classic RL frameworks, where the key is to learn a less-interpretable value function. Differently, PDP attacks control/learning problems from trajectory optimization perspective (PMP), bypassing the need to solve value functions. Also, the trajectory has a direct meaning of system behavior. **(E)** Structure means that PDP investigates the relation of *states/inputs at level of each time step*, versus existing black-box NNs, which treat the entire trajectory as a single variable (see Appendix F, Page 11). **(F)** Given demonstration of significant variations, PDP can still find the 'best' objective function within the parameterized function set $J(\boldsymbol{\theta})$ such that its produced $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ has *minimal distance* to the demonstrations. Please see a supportive experiment in Fig. 1e.

Response to Rev. #6: **(A)** PDP does not limit the parameterization of dynamics and allows it to have any differentiable parameterization. Given little physical knowledge of system dynamics, one can represent it by generic neural networks (but would be less efficient to learn than physics-based parameterization), as in Fig. 1c. **(B)** As additional comparison, we here compare PDP with the suggested GAIL and guided policy search. Please see Fig. 1 (will be added to revision). The results show that PDP outperforms GAIL and guided policy search for higher efficiency, lower training loss, and better generality. **(C)** The suggested three references are about how to inject physical laws into construction of physics-informed NNs for learning physically-plausible ODEs. Differently, PDP is not concerned about how to obtain physical-inspired NNs; instead, PDP considers a *generic mathematical parameterization of a control system* and how to efficiently train such parameterized control system. We inject the inductive bias of optimal control theory to PDP framework: the trajectory of OC satisfies PMP, which makes the PDP suitable to efficiently solve control/learning problems. **(D)** The parameterization in PDP can be, of course, the physics-informed NNs e.g. deep Lagrangian networks. Although in submission, the experiments use parameterized physical dynamics, PDP allows for using neural networks to implement dynamics/policy/control-objective as in Fig. 1. **(E)** For reproducibility/clarity, we kindly refer the reviewer to Appendix D, Page 3, for algorithm/implementation details, Appendix E, Page 4, for experiment details, Appendix F, Page 11, for relationship with other learning frameworks. All the mentioned references will be discussed in revision.



(a) IRL Mode (b) IRL Planning (c) SysID Mode (d) Control/Planning Mode (e) IRL Mode with sparse $\boldsymbol{\xi}^d$

Figure 1: Support experiment (robot arm): parameterized control-objective/dynamics/policy are implemented by fully-connected neural networks with 2 hidden layers with 20 nodes each. PDP outperforms others. Dataset/other parameters follow Appendix E in supplementary.