kPAM 2.0: Feedback Control for Category-Level Robotic Manipulation

Wei Gao and Russ Tedrake

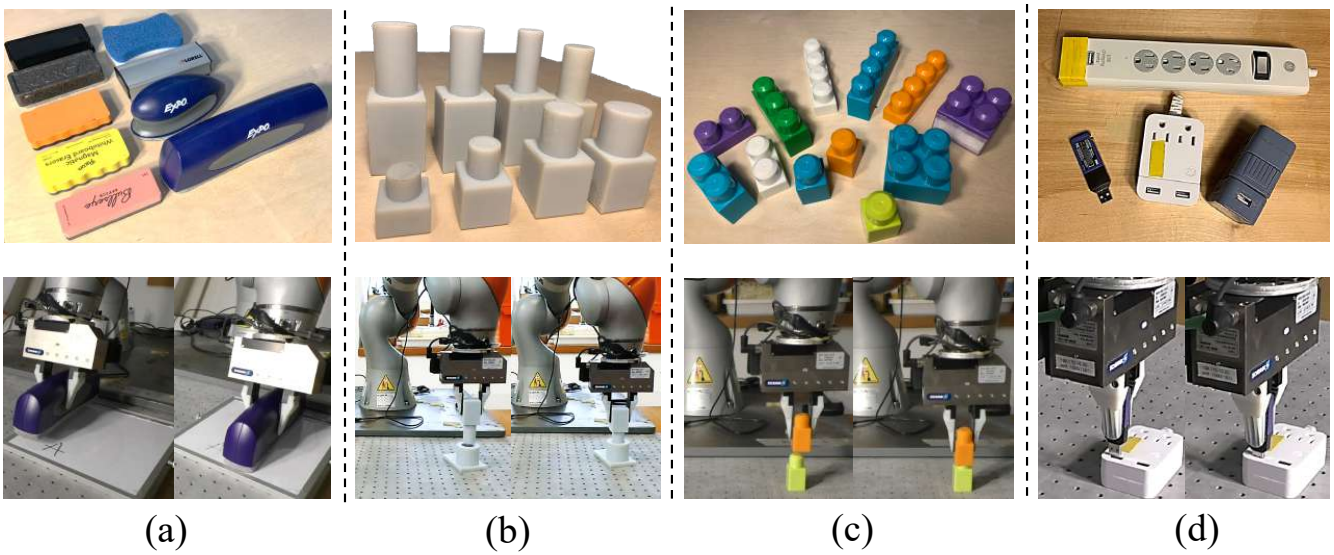

Fig. 1: We explore generalizable, perception-to-action robotic manipulation for contact-rich tasks that can automatically handle a category of objects with large intra-category shape variation. Here we demonstrate: (a) wiping a whiteboard using erasers with different shape and size; (b)-(d) Peg-hole insertion for (b) 0.2 [mm] tight tolerance pegs and holes, (c) LEGO blocks and (d) USB ports. The particular novelty of these demonstrations is that our method automatically handle objects under significant intra-category shape variation (top row) without any instance-wise tuning, for each task (a)-(d). The video demo is on https://sites.google.com/view/kpam2/home.

I. Introduction

To achieve general-purpose, easily deployable manipulation skills in industrial and assistive applications, robots need a perception-to-action manipulation pipeline that is closed-loop (reactive), precise and generalizable. In this paper, we take a step towards this goal with emphasis on generalizability: the manipulation policy should generalize to a category of objects with potentially unknown instances with different shapes, sizes, and appearances. Furthermore, the policy should be able to handle different initial object configurations and robot grasp poses for practical applicability.

While a large body of work addresses robotic grasping for arbitrary objects [17], [16], these methods are typically limited to picking up an object; extending them to other tasks is not straightforward. Contributions on visuomotor policy learning exploit neural network policies trained with data-driven algorithms [9], [2], [18], [20], and many interesting manipulation behaviours emerge from them. However, how to efficiently generalize the trained policy to different objects, camera positions, object initial configurations and/or robot grasp poses remains an active research problem.

On the other hand, several vision-based closed-loop manipulation pipelines [7], [6], [13] use 6-DOF pose as the object representation. They build a feedback loop on top of a real-time pose estimator. However, as detailed in Sec. 4 of kPAM [11], representing an object with a parameterized pose defined on a fixed geometric template, as these works do, may not adequately capture large intra-class shape or topology variations. Thus, kPAM [11] uses semantic 3D keypoints as the object representation instead of 6-DOF pose. The robot action in kPAM [11] is limited to rigid transformations of the keypoints. Although this action representation works well for pick-and-place as demonstrated in [11], it is not suitable for closed-loop policies and contact-rich manipulation tasks.

Contribution. Our main contribution is a novel manipulation framework that is capable of precise, contact-rich manipulation for a category of objects, despite large intra-category variations in shapes, sizes and appearances. To achieve this, we adopt and extend the keypoint-based object representation in [11] proposed for pick-and-place. We first augment keypoints with local orientation information. Using the oriented keypoint, we propose a novel object-centric action representation as the linear/angular velocity or force/torque of an oriented keypoint. This action representation enables closed-loop policies and contact-rich tasks, despite intra-category shape and size variations of manipulated objects. Moreover, our framework is agnostic

CSAIL, Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, USA. Emails: {weigao, russt}@mit.edu.

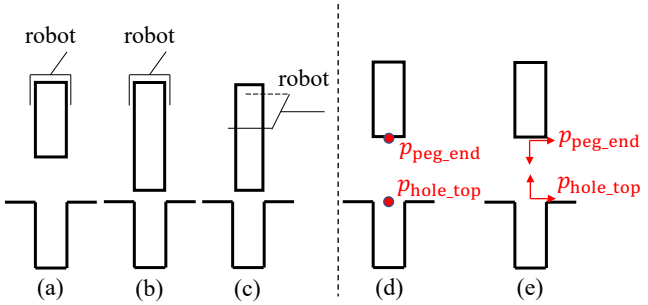

Fig. 3: The object representation using peg-hole insertion as an example. We would like the manipulation policy generalize to (b) a different peg; and (c) a different robot grasp pose. We adopt the semantic 3D keypoint proposed in kPAM [11] as a local but task-specific object representation, as shown in (d). Since the task depends on the local relative orientation between the peg and hole, we augment keypoint with orientation information, as if a rigid coordinate is attached to each keypoint shown in (e).

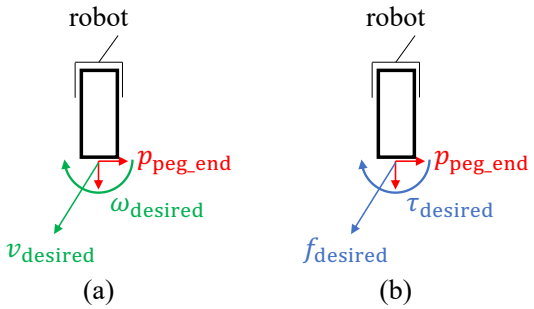

Fig. 4: Overview of the object-centric action space. With the oriented keypoint in Fig. 3 (e) as the object representation, the action space can be defined as: (a) the desired linear/angular velocity of an oriented keypoint; or (b) the desired force/torque of an oriented keypoint. Note that these two action representations are agnostic to the robot grasp pose.

to the grasp pose and object initial configuration, making it flexible for integration and deployment.

Another desirable property of our framework is the extendibility. As shown in Sec. II-B, our framework includes a perception module and a feedback agent, establishes their interfaces but leaves the room for their actual implementation. Thus, various existing model-based or data-driven algorithms for perception and control can potentially be plugged into our framework and automatically generalize to new objects and task setups, as long as the proposed object and action representation are used as their input/output.

Our framework is instantiated and implemented on a hardware robot. We demonstrate several contact-rich manipulation tasks that requires precision and dexterity for a category of objects, such as peg-hole insertion for pegs and holes with significant shape variations and tight clearance.

II. Manipulation Framework

A. Concrete Motivating Example

Consider the task of peg-hole insertion, as illustrated in Fig. 3 (a). We want to come up with a manipulation policy that automatically generalizes to a different peg in Fig. 3 (b), and a different robot grasp pose in Fig. 3 (c).

kPAM [11] proposed to represent the object by a set of semantic 3D keypoints. The motivation is: keypoint is well defined within a category while 6-DOF pose cannot capture large shape variation (see Sec. 4 of [11] for details). We adopt this idea and choose two keypoints: the $p_{\text{peg\_end}}$ that is attached to the peg and the $p_{\text{hole\_top}}$ that is attached to the hole, as shown in Fig. 3 (d). Similar to kPAM, we assume that we have a keypoint detector that can produce these specified keypoints in real-time.

These two keypoints provide the location information. However, the peg-hole insertion task also depends on the relative orientation of the peg and hole. Thus, we augment the keypoint with orientation information, as if a rigid coordinate is attached to each keypoint, as shown in Fig. 3 (e). For the peg-hole insertion task, we let the z axis of the $p_{\text{peg\_end}}$, $p_{\text{hole\_top}}$ be the axis of the peg and hole, respectively. The x axis of $p_{\text{peg\_end}}$, $p_{\text{hole\_top}}$ can be chosen arbitrarily, but when the x axes of $p_{\text{peg\_end}}$ and $p_{\text{hole\_top}}$ are aligned the peg should be able to insert into the hole.

The coordinate in Fig. 3 (e) is also used to illustrate 6-DOF pose in the literature. The key difference between the oriented keypoint and 6-DOF pose is: oriented keypoint is a *local* but task-specific characterization of the object geometry, while pose with geometric template is global. The choice of a local object representation is inspired by the observation that in many manipulation tasks, only a local object part interacts with the environment and is important for the task. For instance, the $p_{\text{peg\_end}}$ keypoint only characterizes a local object part that will be engaged with the hole, and it does not imply task-irrelevant geometric details such as the handle grasped by the robot. This locality enables generalization to novel objects as the unrelated geometric details are ignored.

As illustrated in Fig. 4, with the oriented keypoint as the object representation we propose to represent the robot action as either 1) the desired linear and angular velocity of the $p_{\text{peg\_end}}$ keypoint, as shown in Fig. 4 (a); or 2) the desired force and torque at the $p_{\text{peg\_end}}$ keypoint, as shown in Fig. 4 (b). Note that these two object-centric action representations are agnostic to the robot grasp pose, since these actions are defined only w.r.t the object (not the robot). Under the assumption of no relative motion between the peg and the robot gripper (the grasp is tight), these actions can be mapped to joint space commands, as described in Sec. II-E.

Suppose we have implemented an agent (which can be a hand-written controller or a neural network policy) using the object and action representations mentioned above as the input and output, together with a perception module that produces the required keypoints in real-time and a joint-level controller that maps the agent output to joint command, then the resulting manipulation policy would automatically generalize to different objects and robot grasp poses, for instance the ones in Fig. 3 (a), (b) and (c). Even if the policy doesn't directly transfer due to unmodeled factors, it would be a good initialization for many data-driven or model-based algorithms [8], [10], [15].

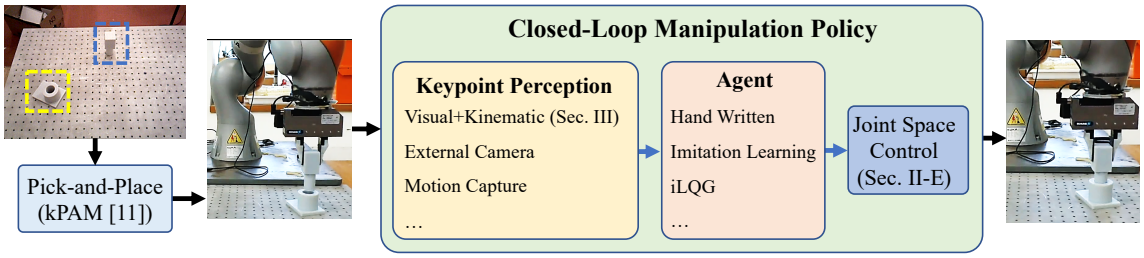

Fig. 5: Overview of the manipulation framework. The closed-loop policy consists of 1) a perception module that produces oriented keypoint in real time; 2) an agent with the state and action space shown in Fig. 3 and Fig. 4, respectively; 3) a joint-space controller that maps agent outputs to joint-space commands. Note that many different implementations of the perception module and agent can be used within our framework and the resulting pipeline automatically generalize to new objects and task setups. For many applications the objects are randomly placed initially. In this scenario, we perform a kinematic pick-and-place to move the object to some desired initial condition (for instance moving the peg right above the hole), from where the closed-loop policy starts operating (see kPAM [11] for details).

B. General Formulation

We can think of a robot as a programmable force/motion generator [5]. We propose to represent the task-specific motion profile as the motion of a set of oriented keypoints, and the force profile as the spatial force/torque w.r.t some keypoints.

Thus, given a category-level manipulation problem we propose to solve it in the following manner. First the modeler selects a set of 3D oriented keypoints that capture the task-specific force/motion profile. Once we have chosen keypoints, the manipulation framework can be factored into: 1) the perception module that outputs the oriented keypoint from sensory inputs; 2) the agent that takes the perceived keypoint as input and produces the desired linear/angular velocity or force/torque of an oriented keypoint as output; 3) the low-level controller that maps the agent output to joint-space command. An illustration is shown in Fig. 5. The framework can be extended with force/torque measurements and the generalization to different object initial configurations, as shown in Sec. II-C and Sec. II-D. For many applications, objects are randomly placed initially. In this case, we perform a kinematic pick-and-place to move the object to some desired initial configurations, from where the closed-loop policy starts, using the method in kPAM [11]. To make the overall manipulation operation generalizable, all these extensions should work for a category of objects.

It should be emphasized that our framework establishes the interfaces (input/output) of the perception module and closed-loop agent, but leaves the room for their instantiation. The only requirement is that for the perception module it should output oriented keypoints in real time, and for the agent it should use the state and action space mentioned above. There are many solutions for both of them. For instance, in our experiment we combine the wrist-mounted camera and robot kinematics for keypoint perception. Alternatively, external cameras or motion capture markers can also be used for keypoint tracking. Similarly one might explore various model-based or data-driven controllers as the feedback agent according to the task in hand. Integrating these perception module and controllers into our framework would achieve automatic generalization to new objects and task setups, as long as the proposed object and action representation are used as the input/output.

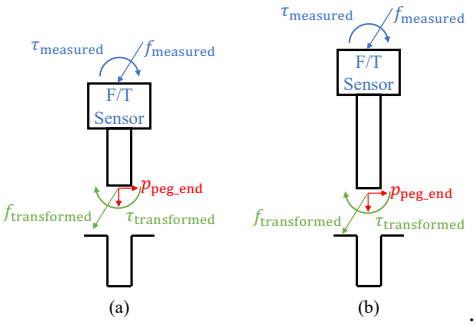

Fig. 6: The processing of force/torque measurement in our formulation. For a wrist-mounted force/torque sensor, its raw measurements f_{measured} and τ_{measured} vary with the object geometry and grasp pose. Thus, we propose to transform them to an oriented keypoint ($p_{\text{peg\_end}}$ here), as the transformed measurement captures the task-specific force/torque profile. The closed-loop agent takes the transformed force/torque measurement as input would generalize w.r.t object geometry.

C. Force/Torque Measurement

Some robots are equipped with wrist-mounted force/torque sensors or joint torque sensors. For contact-rich manipulation tasks, it's beneficial to use this information as the input of the agent. However, the raw output from these sensors varies with the object geometry and robot grasp pose, as shown in Fig. 6. As a result, directly feeding these measurements into the agent does not generalize automatically.

The solution to this problem is to transform the measured force/torque to the kinematic frame of an oriented keypoint, as shown in Fig 6. Using the peg-hole insertion as an example, we can transform the force/torque measurement from the robot wrist to the coordinate of $p_{\text{peg\_end}}$, as if a "virtual sensor" is mounted at $p_{\text{peg\_end}}$. In this way, we can expect similar force/torque profiles across different objects and robot grasp pose, as shown in Fig 6.

D. Generalization w.r.t Global Rigid Transformation

Suppose we want to re-target the peg-hole insertion policy to the hole at a different location. Intuitively, this re-targeting

is essentially a rigid transformation of the manipulation policy. Can we somehow "apply" this transformation directly?

In our framework, both the agent input (oriented keypoints and force/torque w.r.t keypoints) and output (linear/angular velocity or force/torque of an oriented keypoint) are expressed in 3D space. In other words, we can apply a rigid transformation to both the agent input and output. This property provides generalization w.r.t the global rigid transformation. Before feeding the input to the agent, we can transform the input from the world frame to some "nominal frame". After agent computation, we can transform its output back to the world frame. The "nominal frame" can be chosen arbitrary, for instance in the peg-hole insertion task we can align it with the initial configuration of $p_{\text{hole\_top}}$. Thus, the global rigid transformation is transparent to the agent.

On the contrary, many existing contributions [9], [2], [3] work with input/output in joint space and/or image space (raw image or 2D keypoint), on which a rigid transformation cannot be applied. Thus, re-targeting agents in these methods to new object initial configurations and camera positions might need re-training.

E. Joint Space Control

The agent output is the desired linear/angular velocity or force/torque of an oriented keypoint p. An important observation is: if we assume the object is rigid and grasp is tight (grasped object is static w.r.t the gripper), then the object can be regarded as part of the robot and consequently standard joint-space controllers can be used map these agent outputs to joint-space commands. This generalizes the "object attachment" in collision-free manipulation planning [1].

With this observation, we discuss several possible implementations of joint-space controller according to the robot interface. Let J_p be the Jacobian the maps the joint velocity $\dot{q}$ to the linear/angular velocity of p. One straightforward method to transform the commanded velocity v_p into joint space velocity command $\dot{q}_{\text{desired}}$ is

$$\dot{q}_{\text{desired}} = \text{argmin}_{\dot{q}} |J_p \dot{q} - v_p|^2 + \text{reg}(q) \tag{1}$$

where $\text{reg}(\dot{q})$ is a regularizer term. If the robot driver accepts joint velocity commands, we can send $\dot{q}_{\text{desired}}$ to the robot directly. Similarly, the desired force/torque F_p can also be transformed into joint space using J_p by

$$\tau_{\text{desired}} = J_p^T F_p + g \tag{2}$$

where g is the gravitational force in joint space. Here we ignore the inertia and Coriolis force of the robot.

Since standard joint-space controllers can map the agent output to joint commands, some more sophisticated controllers can also be used and might provide better tracking performance, for instance the task-space impedance controller described in [12]. The detailed discussion is omitted as they are out of the scope of this paper.

III. Results

We test our framework on a hardware robot and demonstrate a variety of contact-rich manipulation tasks, as shown in Fig. 1. The video demo is on this link.

TABLE I: #Failure/#Trial (Failure Rate) Comparison

Task	Our Method	Open-Loop Baseline
Whiteboard wiping	1/25 (4%)	16/20 (80%)
Printed peg-hole	12/45 (26%)	19/20 (95%)
LEGO block	2/25 (8%)	7/20 (35%)
USB port	9/20 (45%)	20/20 (100%)

TABLE II: Summary for 6DOF-Pose Baseline

Task	Failure Rate	Note on Failure
Whiteboard wiping	10/25 (40%)	Eraser collide into table; Eraser not aligned with whiteboard edge
Printed peg-hole	25/45 (55%)	Peg collide into hole; No overlapping between peg and hole (alignment error too large)
LEGO block	9/25 (36%)	
USB port	17/20 (85%)	

A. Perception Implementation

We use robot kinematics to track the oriented keypoint in real-time and use it as the input of the closed-loop agent. Suppose we know the oriented keypoint relative to the robot gripper, then when robot moves we can compute the oriented keypoint in the world frame using the forward kinematics. Here we assume and the grasp is tight (no relative motion between the object and the gripper), which is well satisfied in our experiment.

We use a robot wrist-mounted camera to perform object detection, keypoint detection and grasp planning with the method in kPAM [11]. Given visual perception results, we execute robot grasping and compute the keypoints expressed in the robot gripper frame, using the keypoints in the world frame (from camera perception) and the robot gripper pose (from robot kinematics). After grasping, we use robot kinematics for real-time keypoint tracking and feed the result into the closed-loop agent, as mentioned above. Other objects (and keypoints on these objects) are assumed static. Benefit from the automatic keypoint annotation pipeline using multiview consistency in kPAM, it only takes four hours to annotate the training data for all experiments.

B. Experimental Results

Our method is compared with two baselines. The first baseline simply follows a trajectory of an keypoint without feedback. This baseline is similar to the policy in kPAM [11], Form2Fit [19] and KETO [14]. The second baseline uses 6DOF pose as the object representation, and the 6DOF pose baseline (Fig. 4) in kPAM [4] is used as the pose estimator: first initialize the alignment with detected keypoints, then perform ICP fitting between the observed point cloud and geometric template to get the 6-DOF pose. As shown in Table. I and Table. II, our approach has a much better performance than these baselines.

References

[1] R. Diankov. Automated construction of robotic manipulation programs. 2010.

[2] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[3] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 2019.

[4] W. Gao and R. Tedrake. kpam-sc: Generalizable manipulation planning using keypoint affordance and shape completion. *arXiv preprint arXiv:1909.06980*, 2019.

[5] R. Holladay, T. Lozano-Pérez, and A. Rodriguez. Force-and-motion constrained planning for tool use.

[6] T. R. Institute. Tri manipulation for human-assist robots, 2019.

[7] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg. Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, 3(3):1864–1871, 2018.

[8] V. Kumar, E. Todorov, and S. Levine. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[10] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. *arXiv preprint arXiv:1501.05611*, 2016.

[11] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *The International Symposium on Robotic Research (ISRR)*, 2019.

[12] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757, 2008.

[13] Nvidia. Celebrating robotics in seattle, 2019.

[14] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese. Keto: Learning keypoint representations for tool manipulation. *arXiv preprint arXiv:1910.11977*, 2019.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[16] M. Schwarz, C. Lenz, G. M. García, S. Koo, A. S. Periyasamy, M. Schreiber, and S. Behnke. Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3347–3354. IEEE, 2018.

[17] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473, 2017.

[18] H. Van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934. IEEE, 2016.

[19] K. Zakka, A. Zeng, J. Lee, and S. Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. *arXiv preprint arXiv:1910.13675*, 2019.

[20] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.