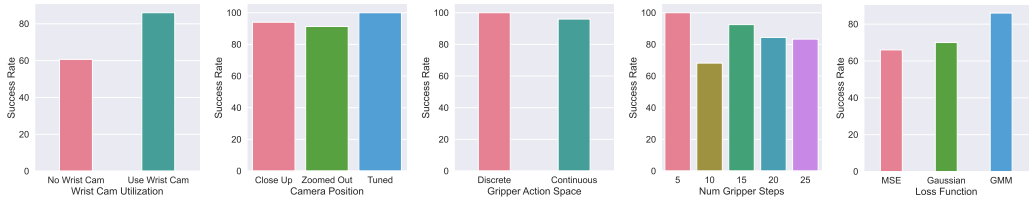
Appendix

A Agent Structure



(a) Wrist Camera (b) Camera Position (c) Gripper Control (d) Gripper Steps (e) Loss

Figure A.1: **Effect of Observation, Action and Loss Decisions.** We ablate a variety of design decisions in OPTIMUS and demonstrate that each produces a clear improvement.

Observation spaces: We use the same set of proprioceptive observations across all tasks: end-effector position, end-effector orientation (quaternion), gripper position. For each task, we select a different camera view that maximizes scene coverage. For Shelf and Microwave, we use two views, left and right shoulder views, whereas for the rest of the tasks we use a single forward facing view. Additionally, we use a wrist camera for every task, which greatly improves the performance. We use camera images of size 84x84. We empirically validate these decisions in Sec. B.1 and visualize the results in Fig. A.1.

Action spaces: As mentioned in the main text, we use task space control for moving the arm. In Robosuite, we use the built-in OSC controller [59]. In IsaacGym, we used a simple IK-based task-space controller. With regard to gripper control, we discuss and resolve two challenges related to TAMP. 1) Continuous gripper actions produced by the TAMP solver can be challenging for the network to fit, as the network does not fully commit to predicting grasps. To that end, we modify the gripper actions to be binary open and close motions which improves performance and reduces noise in policy execution. We validate that this results in a performance improvement in Sec. B.1. 2) TAMP demonstrations can include"stall regions": segments of the trajectory in which the robot is not moving, such as when TAMP executes gripper-only actions for grasps and placements. This results in trained policies that may freeze after grasping an object, as the data does not contain cues for when to exit the stall region. To address this issue, we tune the length of stall regions during data collection against the agent's history length to ensure data collection success rate remains high while minimizing policy freezing behavior.

B Additional Learning Results

OPTIMUS can learn to adapt its behavior based on the scene configuration. We evaluate OPTIMUS on two tasks that involve adapting the task plan based on the configuration of objects in the scene: StackAdapt and MicrowaveAdapt, and two that require adapting motions to randomized receptacle sizes: ShelfReceptacle and MicrowaveReceptacle. As shown in Table B.1, OPTIMUS is able to effectively leverage visual input to learn when additional stacking operations are needed (StackAdapt) or when the area in front of the microwave needs to be cleared (MicrowaveAdapt), achieving 96% and 75% respectively, compared to the best baseline (96% and 40%). Additionally, we demonstrate that OPTIMUS is able to effectively learn to generalize to unseen receptacle sizes with high success rates, achieves 80% and 70% on held out shelves and microwaves respectively. These results illustrate that OPTIMUS can distill scene conditioned task plan adaptation and motion generalization across scene configurations from TAMP supervision.

Dataset	BC-MLP	BC-RNN	BeT	OPTIMUS
StackAdapt	**96**	92	81	**96**
MicrowaveAdapt	25	40	13	**75**
ShelfReceptacle	72	71	59	**80**
MicrowaveReceptacle	48	55	31	**70**

Table B.1: **Scene-based adaptation results**. OPTIMUS can learn to vary the task plan it executes based on the scene configuration *(rows 1 and 2)* as well as adapt to unseen receptacles (*rows 3 and 4*).

We describe and empirically validate three advantages of the distilled policies over the TAMP system: 1) success rate improvement over the TAMP supervisor, 2) faster run-time, 3) operation from perceptual instead of state input.

OPTIMUS almost doubles the performance of the TAMP supervisor. To evaluate TAMP, we execute 50 trials averaged over three random seeds on each single-task environment and record the performance in Table B.2. We find that OPTIMUS is able to outperform the TAMP system by a wide margin, from 20% on the easiest task, PickPlace, to 64% on Microwave-1 and 44% on the hardest task, PickPlaceFour. TAMP with joint space control has better performance on average than TAMP with task space control (52% vs. 45%), but still performs significantly worse than OPTIMUS (52% vs. 87%). We instead find that not all grasps execute perfectly every time, likely due to differences in simulation, planning and control schemes from the ACRONYM paper. As a result, we observe grasp execution failures and object slippage during placement motions. OPTIMUS avoids learning these failure cases by only distilling the successful trajectories, which enables it to successfully generalize to unseen configurations of the task.

Dataset	TAMP-joint	TAMP-task	OPTIMUS
PickPlace-1	82	82	**100**
PickPlaceTwo	52	58	**96**
PickPlaceThree	40	50	**91**
PickPlaceFour	34	16	**60**
Shelf-1	58	44	**91**
Microwave-1	46	22	**86**
Average	52	45	**87**

Table B.2: **Comparison of OPTIMUS vs. TAMP**. We plot percentage success on randomly chosen states from the environment. We find OPTIMUS greatly outperforms the TAMP supervisor, whether TAMP uses task space control or joint space.

OPTIMUS executes 5-7.5x faster than TAMP. We evaluate the run-time of OPTIMUS against TAMP by computing the average time per step for both systems across 100 trials. We run the evaluation on a machine with an RTX 3090 GPU and Intel i9-10980XE CPU and include the results in Table B.3. TAMP takes 0.15s per action on average while OPTIMUS (30M parameters) takes 0.021s per action and OPTIMUS (100M parameters) takes 0.031s per action. TAMP pays a high up-front cost of 2-5 seconds, and then executes a feedback controller to quickly track the planned way-points. In contrast, OPTIMUS spends a constant amount of time per action. Furthermore, it is possible to greatly improve the inference time performance of OPTIMUS by employing techniques such as FlashAttention [70], model compilation, and TensorRT.

TAMP	OPTIMUS (30M)	OPTIMUS (100M)
.15s	.021s	.031s

Table B.3: **Timing Results.** We measure the average time taken per action (lower is better). On average, OPTIMUS is 5-7.5x faster to execute than TAMP.

By distilling TAMP, we obtain a performant policy that executes high-frequency *low-level* control from *purely perceptual* input. OPTIMUS produces policies that are fast to execute, reactive and perform visuomotor control at similar performance to policies that have access to state information (Fig. A.1) and out-performs the privileged TAMP expert (Table B.2).

B.1 Ablations

In this section, we ablate additional components of OPTIMUS, namely the gripper control scheme and data generation process, observation space design and loss function.

Discrete gripper control and short "stall" regions directly impact the performance of TAMP imitation. We first analyze the impact of switching from continuous to discrete gripper control on the Stack task in Fig. A.1. By using discrete control, we can improve the success rate by 4%, while qualitatively we observe smoother gripper control and decisive grasps. On the other hand, we find that the decision to tune the length of "stall" regions, namely TAMP grasp and release actions, is crucial to the performance of OPTIMUS. As observed in Fig. A.1, reducing the number of control actions per grasp and release action greatly improves performance, from 78% at 25 steps to 100% at 5 actions. This is likely due to two reasons, 1) we shorten the overall length of the roll-outs, easing the learning burden, and 2) we reduce the likelihood of the policy to encounter a series of states where the observations and actions do not change, which can result in freezing behavior in the policy.

Camera view selection enables greatly improved visuomotor learning. We evaluate two camera views on the Stack task. Both camera poses keep all objects as well as the robot in view; one is close up which hinders accurate estimation of scene geometry while the other is farther away which decreases the size of the objects in the frame, making it difficult for the policy to focus on them. As a result, we find in Fig. A.1 that a well-tuned camera view that is angled and positioned appropriately performs best. We additionally evaluate the impact of using a wrist camera. For tasks with primitive objects such as blocks, we found that the wrist cam had little impact. However, moving to tasks such as Microwave, where close up views of the handle and target object enable improved perception of grasp geometries, the wrist camera affords a significant performance improvement as we show in Fig. A.1.

GMM loss enables OPTIMUS to better handle the multi-modality of TAMP supervision. TAMP generates highly multi-modal action distributions through randomized planning and non-deterministic IK. Therefore, as we note in Sec. 3.3, we use Gaussian Mixture Models to model the multi-modality. We experimentally validate that GMM output distributions greatly improve learning performance by comparing against MSE loss, which produces a deterministic, uni-modal output distribution, and Gaussian log-likelihood, which produces a non-deterministic, uni-modal output distribution. We find that GMM loss greatly out-performs both output distributions (86% vs. 66% and 70%). While including a stochastic output distribution such as a Gaussian does improve performance by 4%, the multi-modality of GMM produces a further improvement of 16% performance. The results demonstrate that by providing the policy a more expressive output distribution, we can greatly improve how well the policy can model the TAMP expert.

C Environments

In this section, we provide a detailed description of the environments we use to evaluate OPTIMUS. We begin by describing settings which are common across environments. We then discuss each task individually.

For all tasks, we use a Franka Panda 7-DOF manipulator with the default Franka gripper, though the TAMP system is capable of generating supervision using any manipulator, provided the robot URDF. For the Stack task, we use the block stacking environment from Robosuite [58], modifying it to include up to 5 blocks and a larger workspace region. For all other tasks we use IsaacGym [71] with the PhysX [72] back-end. For each task, we use a fixed reset pose for the robot, while randomizing the positions of sampled objects. Object orientation about the z-axis is sampled uniformly at random from 0 to 360 degrees for all tasks.

For PickPlace, Multi-step PickPlace, Shelf and Microwave, we sample objects from ShapeNet [49]. We select objects that have valid grasps in the Acronym [51] dataset. We further refine our dataset by filtering out objects that do not simulate well in our IsaacGym environments. From the remaining objects, we form two datasets with 19 and 72 objects respectively.

We next provide additional details for each task.

Stack: The goal is to stack the blocks in a fixed ordering. Each block is a different color. The block positions are sampled uniformly in an area of size 28cm x 28cm. The base block is of size $2.5cm^3$; the rest are of size $2cm^3$. The task is considered solved if all of the blocks are stacked in the correct ordering.

StackAdapt: The task is the same as Stack, except there are two platforms, the blocks must be stacked on the target platform only. There is a 50/50 chance for the base block to be spawned on the target platform, in which the task simply involves stacking, and the base block to be spawned on the other platform, which requires the agent to first place the base block on the target platform then stack on top of it.

PickPlace: The task involves picking and placing ShapeNet objects from the left platform to the right platform. The platforms are of size .25 by .25 and are kept .5 apart. The object positions are sampled uniformly at random on the platform. The task success criteria is fulfilled if the object is placed anywhere on the target platform.

Multi-step PickPlace: The task involves picking and placing ShapeNet objects from platforms on the left to bins on the right. Up to four objects: a basket, vase, magnet or cup are sampled on separate platforms. Each platform is of size .15x.15 and each bin is of size .2x.2m. Each object's position is sampled uniformly at random on its associated platform. The task is solved when all objects are in their associated bins.

Shelf: The task involves moving ShapeNet objects from the lower rung of the shelf to the middle one. The shelf is 1m tall and has three rungs of size .5m x .25. The position and size of the shelf are constant. Object positions are sampled on the lowest rung, uniformly at random across the surface. The task is solved when the object is placed on the middle rung.

ShelfReceptacle: This task is the same as Shelf, but the shelf size is randomized within the following intervals: height (.8-1m), rungs: (.5x.25m - .4x.75m).

Microwave: The goal is to open the microwave by pulling open the handle, grasp a ShapeNet object, and place it inside the microwave. The microwave is .3m tall, 50cm wide and 20 cm deep. Microwave position and size are held fixed. The initial angle of the microwave door is 0, i.e. fully closed. Object positions are sampled on a platform of size .25x.25m. The agent has succeeded when the object is inside the microwave.

MicrowaveReceptacle: This task is the same as Microwave, but the microwave size is randomized within the following intervals: height (.3-.4m), width: (.5-.6m), depth: (.2-.3m).

MicrowaveAdapt: The task is the same as the microwave task, except with 50% probability an object is spawned in front of the microwave door, requiring the agent to first move the object aside then open the door and place the target object inside.

D Experiment Details

Hyper-parameter	Value
Learning Rate	0.0001
Batch Size	16/512
Warmup Steps	0
Linear Scheduling Steps	100K
Final Learning Rate	0.00001
Weight Decay	0.01
Gradient Clip Threshold	1.0
Number of Gradient Steps	1M
Optimizer Type	AdamW
Loss Type	GMM
GMM Components	5
GMM Min. Std. Dev.	0.0001
GMM Std. Dev. Activation Fn.	SoftPlus

Table D.1: Hyper-parameters used during training.

	OPTIMUS (30M/100M)	MLP (30M/100M)	RNN (30M/100M)	BeT (30M/100M)
Num Layers	6/12	2/6	2/3	6/12
Hidden Dimension		1024/1024	1000/2000	
Context Length	8/8		10/10	10/10
Num Heads	8/16			8/16
Transformer Embed. Dim.	256/512			256/512
Embedding Dropout Prob.	0.1/0.1			0.1/0.1
Attention Dropout Prob.	0.1/0.1			0.1/0.1
Output Dropout Prob.	0.1/0.1			0.1/0.1
Positional Embed.	Learned/Learned			Learned/Learned
Positional Embed. Type	Relative/Relative			Relative/Relative
Num. Clusters				24/24
Offset Loss Scale				100/100

Table D.2: Model hyper-parameters.

Network and Training Details: We include the model hyper-parameters for the 30M and 100M parameter variants of each method in Table D.2. For the vision-backbone, as discussed in the main text, we use a Resnet-18 [60] with a Spatial Softmax [61] output to encode each image separately. For details, please see the Robomimic paper [29]. We include learned positional embeddings with each token and employ relative, rather than absolute, position embeddings to enable the network to adapt to longer horizons at test time. We use a linear annealing schedule that reduces the learning rate from 10^{-4} to 10^{-5} over 100K gradient steps and then keeps the learning rate constant. We train with the AdamW optimizer with a weight decay of 0.01 and no learning rate warm-up. For single-task learning, we train with a batch size of 16 on a single V100 GPU, while for multi-task learning we train using batch size of 512 to 1024 depending on the task, across 8 V100 GPUs. For visuomotor learning, we train with multiple camera views with image size 84x84, and we augment the data with random crops [29, 73, 74]. We additionally list the hyper-parameters used for training in Table D.1. One note of interest: for multi-task training, we found that increasing the batch size greatly improved the results; hence we use a batch size of 512.

For BeT, we tried using the original authors codebase, which we augmented with our vision backbone, but found that the performance was extremely low. Instead, we re-implemented BeT as a modification of OPTIMUS, using the same network structure but predicting a discrete cluster center and offset head instead and training using the focal and MT losses from the BeT paper. We found that the standard hyper-parameters for BeT did not perform well, and after significant hyper-parameter tuning found that the combination of 24 cluster centers and offset loss scale of 100 performed best.

Evaluation Protocol: We note additional details regarding our evaluation protocol as follows. We split each dataset into a set of training and validation trajectories (using a 90/10 split). From the validation trajectories, we save the initial state of the demonstration. During evaluation, we reset the simulator state to an initial state from the validation set, and execute the policy from there. By comparing on the same set of validation states, we can better evaluate performance across seeds and algorithms. Note this means evaluation is performed from states that the TAMP solver is able to solve. As we note in Sec. 4.1, in practice this distinction matters little, as the TAMP system does not have a systematic failure case which could be passed on to the policy. Therefore we observe similar success rates when evaluating on randomly sample poses from the environment.

E Related Work

E.1 Offline Learning from Demonstrations

Imitation Learning (IL) is a paradigm for training robots to perform manipulation tasks by leveraging a set of expert demonstrations. In this work, we focus on offline learning, in which a policy learns a dataset of demonstrations, without any additional interaction. This is typically done through Behavior Cloning (BC) [30], in which a policy is trained to imitate the actions in the dataset through supervised learning. While this is a simple approach, it has proved incredibly effective for robotic manipulation [29, 31, 32, 33, 34, 35, 36, 37], particularly when coupled with a large number of demonstrations [10, 20, 38, 39]. Concurrent work has proposed leveraging Diffusion Models [75] to train policies via BC [76] in order to handle multi-modality of demonstrations. Our work instead focuses on how to best imitate TAMP with Transformers; Diffusion Policies, in particular their Transformer variants, could be straightforwardly integrated into OPTIMUS.

Human supervision is a common source of demonstrations. Several prior works use kinesthetic teaching [77, 78, 79, 80], in which a human manually guides an arm through a task, but this does not scale. Many works have leveraged teleoperation systems [13, 14, 15, 20, 35, 38, 39, 81, 82, 83], in which a human remote controls a robot arm to guide it through a task. However, scaling teleoperation is costly because it can require months of data collection and numerous human operators [10, 20, 81]. This has motivated the development of intervention-based systems, in which humans provide smaller corrective behaviors to an agent [84, 85, 86, 87, 88, 89], enabling more sample-efficient learning and less operator burden. Instead of relying on human operators for supervision, we learn policies from demonstrations provided by a TAMP supervisor, which can generate large, diverse datasets without human supervision.

E.2 Transformers for Robot Control

Recent work explores the application of Transformers to controlling robot manipulators. Transformer-based policy architectures such as Gato [12], PerAct [40], VIMA [41], RT-1 [10], Dasari and Gupta [42], and Behavior Transformer [43] have demonstrated impressive results across a range of robotic manipulation tasks, yet make use of discretization of the input observations and output actions, limiting their applicability to tasks requiring precise manipulation. Additionally, PerAct [40] and VIMA [41] use abstracted actions to ease the learning burden at the cost of expressivity and execution speed. HiveFormer [67] is closest to our method in terms of architecture and training protocol but also assumes temporally-extended motion planner actions. As a result, these systems require privileged knowledge of the geometry of the environment to ensure safety. In contrast, OPTIMUS uses a Transformer architecture that is efficient to train and scale, fast-to-execute, consumes raw observations, and outputs low-level control actions.

E.3 Task and Motion Planning

Task and Motion Planning (TAMP) [27] addresses controlling a hybrid system through planning a sequence of discrete of manipulation types (*task planning*) realized through continuous motions (*motion planning*). TAMP approaches consume kinematic or dynamic models [44] of individual manipulation types and search over combining them in a manner that achieves a goal. Classically, these models are engineered; however, recently, they have been learned using methods such as Gaussian Processes [64] or Deep Neural Networks [65, 90, 91]. These mixed engineering-learning TAMP techniques can be quite effective, but they impose a strong human design bias, capping policy performance. Also, they are too computationally expensive to be run in real-time, preventing them from quickly reacting to new observations.

There has been recent interest in approaches that imitate planning [45, 46, 47]; however, these approaches generally focus on single-step motion generation. The exception is [28], which recently proposed an approach, Guided TAMP, that directly imitates TAMP. Our work builds on this direction in several ways. First, Guided TAMP primarily addresses control from privileged state, while we focus exclusively on visuomotor learning, which requires fewer assumptions. Second, Guided TAMP proposes a hierarchical policy that first predicts a discrete task-level action and then, conditioned on that action, predicts the next control. In order for the learner to predict a task-level action, they require a fixed set of ground actions, preventing the same policy from being deployed in tasks, for example,

with varying numbers of objects. In contrast, our Transformer architecture does not explicitly reason about task-level actions and thus does not require grounding and fixing the objects in the scene. Finally, we identify new considerations when using TAMP as a data generation pipeline.