APPENDIX A
TAXONOMY CATEGORIZATION

Here, we provide more detailed descriptions of how we categorize the axes and factors in ★-Gen based on the combinations of policy modalities they affect.

Visual: Factors and axes in this category modulate the image inputs of the initial state, but do not affect the required behavior in the base task or the language instruction. Example factors include lighting, camera pose, and distractor objects (see the green sector of Fig. 1).

Semantic: Semantic factors and axes modulate the language instruction without changing the initial image or required behavior. Example factors include replacing verbs with synonyms, and changing the instruction to use spatial relationships such as "in the ___" (see the orange sector of Fig. 1).

Behavioral: Behavioral factors and axes only affect required behavior without affecting policy inputs. Therefore, all isolated behavioral factors are necessarily *unobserved* from single observations. Example factors include changes to object mass or friction (see the blue sector of Fig. 1). These factors are often challenging for policies due to their unobservability.

Visual + Behavioral: These factors and axes affect the initial image and required behavior, without changing the language instruction. As we show in Section IV-B, many factors that prior work consider as "behavior" generalization fall into this category [6, 13]. Example factors include manipulated object poses and surface/table heights (see the cyan sector of Fig. 1).

Semantic + Behavioral: These factors and axes affect the language instruction and required behavior, without affecting the initial image. Example factors include changing the speed of a behavior ("quickly" vs. "slowly") in language, or specifying prepositional phrases like "into" or "in front of" that change the required behavior (see the purple sector of Fig. 1).

Visual + Semantic: These factors and axes affect the initial image and language instruction, without requiring change to behavior. An example of this would be if the base instruction is "pick up the purple cup" and the cup changes color to blue, changing the instruction to "pick up the blue cup". This is still an *atomic* perturbation, but since the color was specified in the original instruction, the instruction must also be changed. Had the initial instruction been "pick up the cup" and the cup was already blue, then this would be a **semantic** perturbation. See the brown sector of Fig. 1.

Visual + Semantic + Behavioral: This category of factors and axes affect all three modalities at once. An example is going from "pick up the carrot" to "pick up the zucchini" – this affects the initial image, the language, and the behavior required to pick up the new object.

APPENDIX B
FREQUENTLY ASKED QUESTIONS

Here, we provide answers to several possible questions readers may have about our taxonomy and its assumptions.

Q: Is ★-Gen meant to be comprehensive?

A: The axes presented in ★-Gen are meant as a starting point for the field, and although we did our best to make this list comprehensive, there could certainly exist other meaningful axes. In contrast, we consider the *categories* in ★-Gen (e.g., **visual + behavioral**) to be exhaustive for the policies we consider, since they are built from the seven unique combinations of our policy modalities.

Q: Are the axes and factors in ★-Gen subjective?

A: Yes, the axes and factors in ★-Gen are human-specified in a subjective manner, so there are certainly other ways to group perturbations into factors, and factors into axes. However, our goal is not to design an objective way to categorize generalization, as this is an inherently subjective process. Instead, we aim to provide greater structure and comprehensiveness when categorizing generalization using human interpretable concepts, like "task-relevant objects" and "verbs".

Q: What is the purpose of defining a taxonomy if it is inherently subjective?

A: As we will see in Section V, our taxonomy gives us the vernacular to discuss more fine-grained types of generalization. We intend this taxonomy to be a starting point for practitioners to gain greater insights about their models, and recognize the potential for reshaping our taxonomy as our understanding of how to effectively evaluate generalization grows.

Q: Is each axis equally important for generalization?

A: Whether or not an axis is "important" is very subjective, and depends on specific applications. Instead, we categorize different types of generalization more systematically to aid researchers and practitioners in evaluating their robot policies based on the needs of their downstream applications.

Q: Can you quantify how much different perturbations affect a base task?

A: It would be interesting to consider the "edit distance" that a perturbation induces, to quantify how much generalization is required. However, we do not consider this in ★-Gen, due to the challenging nature of defining such distances. Some options for this could include image or text embeddings for **visual** or **semantic** perturbations, dynamic time warping [50] for **behavioral**, or foundation models. Future work can investigate correlating such metrics with empirical generalization.

APPENDIX C
CASE STUDY 1: BRIDGEV2-★

A. Base Tasks

We consider the following four base tasks in our evaluation for BridgeV2-★:

1) Put carrot on plate.
2) Put knife on plate.
3) Flip pot upright.
4) Put plate in sink.

Each task is instantiated in a replication of a sink environment that was used in Bridge V2, and was also used to evaluate generalization in prior work [13]. We tuned the environment conditions such that the publicly released OpenVLA 7B model

trained on OXE was able to achieve reasonable zero-shot performance on the "put carrot" and "put knife" tasks, in order to increase the likelihood of transfer from the pre-training data for generalization. However, there are inevitably some minor differences from this environment and what was used to collect data in Bridge V2, such as in camera pose or lighting. While we could have chosen base tasks in an environment that deviates more from the pre-training data, this would have likely led to much weaker levels of generalization in our evaluation, which would make our results less informative.

The base tasks "put carrot" and "put knife" are both instantiated from the same initial configuration. "flip pot" and "put plate" are also both instantiated from the same initial configuration. We do this to increase the difficulty and realism for **semantic** generalization, because having multiple tasks from the same initial configuration makes it so that policies must understand the language instruction to know which of the tasks to perform. Therefore, the policy cannot simply ignore the language instruction and memorize which task to perform based on the visual appearance of the scene.

We choose these specific tasks to cover different levels of alignment with the demonstrations from Bridge V2 for this sink environment. We describe how each base task deviates from the support of these demonstrations, and how the publicly released OpenVLA model performs on them, as follows:

1) **Put carrot on plate**: This language instruction is found in Bridge V2 demonstrations for our sink environment. The carrot is different from the one in these demonstrations, but is of similar appearance and geometry. The plate is also different, with a different color, but similar geometry. The initial pose of the carrot and plate are significantly different than in any demonstration, but similar poses for other task-relevant objects are found in other demonstrations for this environment for similar tasks (e.g., "put knife on cutting board"). We find that OpenVLA is able to somewhat reliably perform this task zero-shot.
2) **Put knife on plate**: This language instruction is not found in Bridge V2 demonstrations for our sink environment, but demonstrations with similar instructions are (e.g., "put carrot on plate", "put knife on cutting board"). The knife is different from the one in these demonstrations, but is of similar appearance and geometry. The plate is also different, with a different color, but similar geometry. Similar initial poses for task-relevant objects are found in demonstrations for the task "put knife on cutting board" in this sink environment. We find that OpenVLA is able to somewhat reliably perform this task zero-shot.
3) **Flip pot upright**: This language instruction is found in Bridge V2 demonstrations for our sink environment. The pot used is significantly different from the one used in these demonstrations (e.g., it is plastic instead of metal, and has different geometry). The initial pose of the pot is also somewhat different. We find that OpenVLA is able to rarely perform this task zero-shot.
4) **Put plate in sink**: This language instruction is not found in Bridge V2 demonstrations for our sink environment. However, there is a somewhat similar language instruction in Bridge V2 for a slightly different sink environment ("put cup from anywhere into sink"). The initial pose, appearance, and geometry of the plate is significantly different than in any task demonstrations for our sink environment. We find that OpenVLA is unable to perform this task zero-shot, and exhibits largely meaningless behavior.

We aim to minimize variation in the initial state distribution for each base task (e.g., by attempting to always initialize objects with the same pose). However, in practice there will always be some level of variation in real world experiments such as ours. We aim to minimize variation in order to simplify our experimental setup and to more easily isolate the effect of our considered perturbations. However, our taxonomy and evaluation framework is more general and can be applied to base tasks with greater variation in initial states.

B. Evaluation Procedure Details

We perform our evaluations by executing each policy until either the policy succeeds as deemed by a human evaluator, it reaches a dangerous or irrecoverable state, or terminates after 100 timesteps. For tasks that involve placing an object on another object or surface, we consider success to be if the object is on top of the other object/surface in a stable configuration, and the object is not grasped by the robot. For tasks that involve placing an object on a plate, the plate must also not be knocked over from its initial position. For tasks that involve flipping a container upright, the base of the container must make contact with the base of the sink, the container must be in a stable configuration, and the container must not be grasped by the robot.

We aim to minimize differences between our evaluation conditions and in-domain training data, beyond the differences introduced by a specified perturbation. To do this, we periodically check the image difference between live observations and the base task demonstration data, and adjust the environment to minimize this when the difference is significant. We do this so that our evaluations are only reflective of generalization to a specified perturbation, and not other inadvertent changes [51].

C. Co-fine-tuning vs. Fine-tuning

For our evaluations, we incorporate in-domain data into each model by co-fine-tuning with the model's pre-training data. We do this to help preserve each model's original capabilities, to promote generalization and fair comparison of each model. However, this procedure may become impractical as robot pre-training datasets become larger in size, or if they are not openly available. One could also consider evaluating these models by only fine-tuning on in-domain data, without any pre-training data. However, this should be done carefully to avoid catestrophic forgetting. We leave exploration of best practices for fine-tuning generalist policies, and comparing the generalization of such fine-tuned policies, to future work.

D. In-Domain Data

We collect demonstrations for our in-domain data using a Meta Quest VR headset. For consistency, we collect all

our data using a single experienced human teleoperator. For the "put carrot" and "put knife" base tasks, we collect 10 demonstrations per base task. For the "flip pot" and "put plate" base tasks, we collect 50 demonstrations per base task, because we found that more demonstrations were needed for satisfactory in-distribution performance of models.

E. Model Details

General Training Details. Unless otherwise stated, for all co-fine-tuned models (designated with FT), we co-fine-tune for 5K gradient steps (which we found was sufficient for convergence of most models) with a batch size of 256 and a learning rate of $1e-6$. We adopt a smaller learning rate than what was used during pre-training because we found higher learning rates tend to degrade training metrics on the pre-training dataset. We use the Adam optimizer for all training. When co-fine-tuning, we normalize the in-domain dataset using the dataset statistics from Bridge V2.

For each model type, we co-fine-tune two separate models: one for the "put carrot" and "put knife" base tasks, and another for the "flip pot" and "put plate" base tasks. We do this because we had already trained and partially evaluated models co-fine-tuned only on "put carrot" and "put knife" before expanding our base task selection to include the others. However, it would also been fine to co-fine-tune models on all base tasks together. When co-fine-tuning models for the "put carrot" and "put knife" base tasks, we upsample the in-domain data by 100x relative to the pre-training data. For the "flip pot" and "put plate" base tasks, we upsample the in-domain data by 50x.

We describe the specific training details of each model we consider in our experiments below:

OpenVLA (OXE). We use the publicly released OpenVLA 7B model trained on a mixture of data from OXE [13].

OpenVLA (OXE, FT). We co-fine-tune OpenVLA (OXE) on a combination of the same OXE mixture used to originally train the model (except DROID [11]), as well as the in-domain data for a given set of base tasks.

OpenVLA (Bridge, FT). We first pre-train a version of OpenVLA 7B on Bridge V2 for 200K gradient steps with a batch size of 256. We then co-fine-tune with Bridge V2.

OpenVLA (Bridge, Rand Init, FT). We first pre-train a version of OpenVLA 7B on Bridge V2, except we initialize the model from random weights, rather than a pre-trained vision-language model. We pre-train for 200K gradient steps with a batch size of 256. We then co-fine-tune with Bridge V2.

OpenVLA (Bridge, VQA, FT). We first pre-train a version of OpenVLA 7B on Bridge V2 and the VQA dataset used for training LLaVA-1.5 ([52]). This is a subset of the dataset used for training the base Prism-7B used for initializing OpenVLA [53]. We set of 20% of our pre-training mixture to consist of VQA data. We pre-train for 200K gradient steps with a batch size of 256. We then co-fine-tune with Bridge V2, again using 20% VQA data.

MiniVLA (Bridge, FT). We start from the publicly released MiniVLA model trained on Bridge V2 [38], and then co-fine-tune this model with Bridge V2. We increase the amount of gradient steps during co-fine-tuning from 5K to 10K, which was necessary for convergence. During inference, we only execute the first action from each predicted action chunk.

MiniVLA (Bridge, No VQ, FT). We first pre-train a version of MiniVLA that uses the binning-based tokenizer from OpenVLA rather than the vector quantized action chunking tokenizer used in MiniVLA. We pre-train on Bridge V2 for 300K gradient steps with a batch size of 256 and learning rate of $2e-5$, and then co-fine-tune with Bridge V2.

π_0 (Bridge, FT). We use the π_0 reimplementation from https://github.com/allenzren/open-pi-zero [39]. We start from the publicly released checkpoint pre-trained on Bridge V2 using beta timestep sampling, and then co-fine-tune this model with Bridge V2. We co-fine-tune for 80K training steps and a learning rate of $1e-5$, with a linear warmup of 200 steps. We found it was necessary to increase the in-domain upsampling rate to 1000x relative to Bridge V2 for "put carrot" and "put knife", and 500x for "flip pot" and "put plate", in order to fit the in-domain data more efficiently. During inference, we only execute the first action from each predicted action chunk.

Unlike all other models, this model was pre-trained using image augmentation (random crop, lighting, and color). During co-fine-tuning, we remove image augmentation to increase training speed. Furthermore, this model only uses trajectories from Bridge V2 that have language annotations.

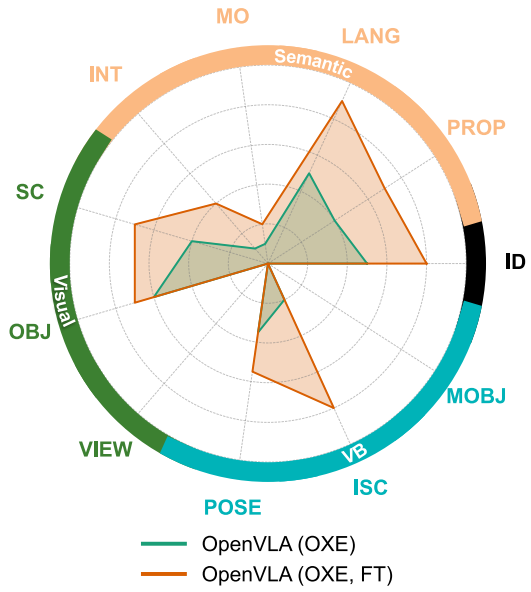


Fig. 11: We find that co-fine-tuning on in-domain data for our base tasks significantly improves both in-distribution performance of OpenVLA, as well as many of our axes.

F. Impact of In-Domain Data

To motivate our choice to assess generalization by comparing models co-fine-tuned on in-domain base task data, in Fig. 11 we compare the publicly released OpenVLA 7B model

zero-shot, and the same model co-fine-tuned with in-domain data, for the "put carrot" and "put knife" tasks. We fine that co-fine-tuning significantly improves both in-distribution base task performance and multiple axes of generalization. This highlights the importance of co-fine-tuning on base task data in order to properly define base tasks as in-distribution. Otherwise, it is likely there will be some significant distribution shift between the pre-training data and the chosen base tasks for evaluation. As a result, performance in general will suffer, making evaluation of generalization more challenging.

G. Random Initialization

In addition to our reported results, we additionally attempted to evaluate OpenVLA (Bridge, Rand Init, FT), a model with the same architecture as OpenVLA, but trained from randomly initialized weights, rather than pre-trained VLM weights. Although training metrics on Bridge V2 and our in-domain data did converge, we found that the model failed completely to succeed at our in-distribution base tasks. Therefore, we did not consider this model in our evaluations.

H. Evaluation Conditions

We detail every generalization condition we consider in our evaluation, including scene images, names for each evaluation condition, axes, language instructions, and additional notes. We provide this information for the main evaluations based on the "put carrot" and "put knife" tasks in Table IV, for the "flip pot" and "put plate" tasks in Table V, and for the compositional evaluations in Table VI.

Implementation Details. For the generalization conditions that involve recoloring the sink (Carrot Red Sink, Knife Red Sink, Pot Green Sink, Plate Green Sink), we do not physically change the sink color. Instead, we preprocess each image frame given to the policy using SAM 2.1 Large [54] to segment the sink, and then recoloring these pixels to the desired color.

For the generalization conditions that make the sink table shorter relative to the robot (Carrot Shorter Table, Knife Shorter Table, Pot Shorter Table, Plate Shorter Table), rather than lowering the sink table, we raise the table that the robot is mounted on, which achieves the same effect.

Scene Image	Condition Name	Axis	Language Instruction	Notes
	Carrot Base	In-distribution	put carrot on plate	N/A
	Knife Base	In-distribution	put knife on plate	N/A
	Carrot Color	S-PROP	put the orange object on the plate	refers to carrot by color
	Knife Color	S-PROP	put the gray and green object on the plate	refers to knife by color
	Carrot Lift/Place	S-LANG	lift carrot and place on plate	replaced verb "put" using "lift" and "place"
	Knife Lift/Place	S-LANG	lift knife and place on plate	replaced verb "put" using "lift" and "place"
	Carrot Counter	S-MO	put the object that is on the counter on the plate	refers to carrot by location (on counter)
	Knife Sink	S-MO	put the object that is in the sink on the plate	refers to knife by location (in sink)
	Carrot Basketball	S-INT	put the object that is the same color as a basketball on the plate	refers to carrot by color of a basketball (orange)
	Knife Typo	S-INT	put knif on plate	"knife" misspelled as "knif"
	Carrot in Sink	SB-SMO	put carrot in sink	goal for carrot is sink instead of plate
	Rotate Knife	SB-VRB	rotate knife clockwise	rotate knife instead of put on plate
	Carrot Distractors	V-SC	put carrot on plate	distractor objects (corn, salt shaker)
	Knife Distractors	V-SC	put knife on plate	distractor objects (corn, salt shaker)
	Carrot Red Sink	V-SC	put carrot on plate	red sink
	Knife Red Sink	V-SC	put knife on plate	red sink
	Carrot Orange Plate	V-OBJ	put carrot on plate	orange plate
	Knife Orange Plate	V-OBJ	put knife on plate	orange plate
	Carrot Camera	V-VIEW	put carrot on plate	new camera pose
	Knife Camera	V-VIEW	put knife on plate	new camera pose

TABLE IV: Generalization conditions for "put carrot" and "put knife".

Scene Image	Condition Name	Axis	Language Instruction	Notes
	Carrot Farther	VB-POSE	put carrot on plate	carrot slightly farther from robot
	Carrot Start Sink	VB-POSE	put carrot on plate	carrot start in sink, oriented vertically
	Knife Raised	VB-POSE	put knife on plate	knife raised by placing it on a block
	Knife Right	VB-POSE	put knife on plate	knife moved to the right
	Carrot Shorter Table	VB-ISC	put carrot on plate	shorter sink table
	Knife Shorter Table	VB-ISC	put knife on plate	shorter sink table
	Baby Carrot	VB-MOBJ	put carrot on plate	real baby carrot
	Small Knife	VB-MOBJ	put knife on plate	smaller knife

TABLE IV: Generalization conditions for “put carrot” and “put knife”.

Scene Image	Condition Name	Axis	Language Instruction	Notes
	Ball	VSB-NOBJ	put ball on plate	carrot replaced with ball
	Pizza	VSB-NOBJ	put pizza on plate	knife replaced with pizza

TABLE IV: Generalization conditions for “put carrot” and “put knife”.

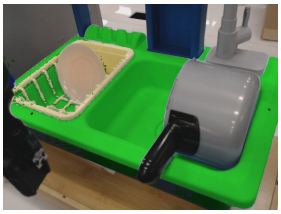
Scene Image	Condition Name	Axis	Language Instruction	Notes
	Pot Base	In-distribution	flip pot upright which is in sink	N/A
	Plate Base	In-distribution	put plate in sink	N/A
	Pot Color	S-PROP	flip the gray object upright which is in sink	refers to pot by color
	Plate Color	S-PROP	put the pink object in the sink	refers to plate by color
	Pot Lift/Place	S-LANG	lift pot upright and place in sink	replaced verb "flip" using "lift" and "place"
	Plate Lift/Place	S-LANG	lift plate and place in sink	replaced verb "put" using "lift" and "place"
	Pot Sink	S-MO	flip the object that is in the sink upright	refers to pot by location (in sink)
	Plate Drying Rack	S-MO	put the object that is in the drying rack in the sink	refers to plate by location (in drying rack)
	Pot Boiling	S-INT	flip the object that can be used for boiling water upright	refers to pot by ability to boil water
	Plate Typo	S-INT	put plait on plate	"plate" misspelled as "plait"
	Plate to Counter	SB-SMO	put plate on counter	goal for plate is counter instead of sink
	Pot to Left	SB-VRB	move pot to the left side of the sink	move pot left instead of flip upright
	Pot Distractors	V-SC	flip pot upright which is in sink	distractor objects (eggplant, fork, cheese)
	Plate Distractors	V-SC	put plate in sink	distractor objects (eggplant, fork, cheese)
	Pot Green Sink	V-SC	flip pot upright which is in sink	green sink
	Plate Green Sink	V-SC	put plate in sink	green sink
	Gray Plate	V-OBJ	put plate in sink	gray plate
	Pot Camera	V-VIEW	flip pot upright which is in sink	new camera pose
	Plate Camera	V-VIEW	put plate in sink	new camera pose

TABLE V: Generalization conditions for "flip pot" and "put plate".

Scene Image	Condition Name	Axis	Language Instruction	Notes
	Pot Left	VB-POSE	flip pot upright which is in sink	pot moved to left
	Pot Angled	VB-POSE	flip pot upright which is in sink	pot rotated and angled to the right
	Plate Closer	VB-POSE	put plate in sink	plate slightly closer to robot
	Plate Counter	VB-POSE	put plate in sink	plate flat on counter
	Pot Shorter Table	VB-ISC	flip pot upright which is in sink	shorter sink table
	Plate Shorter Table	VB-ISC	put plate in sink	shorter sink table
	Thin Pot	VB-MOBJ	flip pot upright which is in sink	thinner and taller metal pot
	Red Bowl	VB-MOBJ	put plate in sink	plate replaced with red bowl

TABLE V: Generalization conditions for “flip pot” and “put plate”.

Scene Image	Condition Name	Axis	Language Instruction	Notes
	Cup	VSB-NOBJ	flip cup upright which is in sink	pot replaced with cup
	Spoon	VSB-NOBJ	flip pot upright which is in sink	plate replaced with spoon

TABLE V: Generalization conditions for "flip pot" and "put plate".

Scene Image	Condition Name	Axis	Language Instruction	Notes
	Carrot Color + Lift/Place	S-PROP + S-LANG	lift the orange object and place on plate	refers to carrot by color and replaced verb "put" using "lift" and "place"
	Knife Color + Lift/Place	S-PROP + S-LANG	lift the gray and green object and place on plate	refers to knife by color and replaced verb "put" using "lift" and "place"
	Carrot Distractors + Orange Plate	V-SC + V-OBJ	put carrot on plate	distractor objects (corn, salt shaker) and orange plate
	Knife Distractors + Orange Plate	V-SC + V-OBJ	put knife on plate	distractor objects (corn, salt shaker) and orange plate
	Carrot Farther + Shorter Table	VB-POSE + VB-ISC	put carrot on plate	carrot slightly farther from robot and shorter sink table
	Knife Right + Shorter Table	VB-POSE + VB-ISC	put knife on plate	knife moved to right and shorter sink table

TABLE VI: Generalization conditions for compositional experiments.

I. More Evaluation Results

1) Composition: We conduct additional evaluations on compositions of our Table VII. We once again see that training on the larger OXE mixture seems to improve many **semantic** axes, specifically the composition of referring to object properties in language and rephrasing language instructions (S-PROP + S-LANG), possibly due to a larger variety of language instructions in the training data. Furthermore, some models are fairly robust to combinations of distractors and object colors (V-SC + V-OBJ), with MiniVLA and π_0 being the most performant, similarly as in the main results. Some models are also surprisingly robust to the composition of different object poses and scene factors (VB-POSE + VB-ISC), with π_0 performing the best in this setting.

2) Detailed Results: We provide individual success rates for all generalization conditions listed in Appendix C-H. We provide this for the main evaluations based on the "put carrot" and "put knife" tasks in Table VIII, for the "flip pot" and "put plate" tasks in Table IX, and for the compositional evaluations in Table X.

	OpenVLA (OXE)	OpenVLA (OXE, FT)	OpenVLA (Bridge, FT)	OpenVLA (Bridge, VQA, FT)	MiniVLA (Bridge, FT)	MiniVLA (Bridge, –VQ, FT)	π_0 Reimplement (Bridge, FT)
Semantic (S-PROP + S-LANG)	6/10	8/10	4/10	0/10	4/10	2/10	1/10
Visual (V-SC + V-OBJ)	3/10	6/10	5/10	6/10	8/10	5/10	7/10
Visual + Behavioral (VB-POSE + VB-ISC)	5/10	6/10	3/10	7/10	5/10	6/10	8/10
Overall	14/30	20/30	12/30	13/30	17/30	13/30	16/30

TABLE VII: Compositional results for two axes from each of **semantic**, **visual**, and **visual + behavioral**.

Condition Name	OpenVLA (OXE)	OpenVLA (OXE, FT)	OpenVLA (Bridge, FT)	OpenVLA (Bridge, VQA, FT)	MiniVLA (Bridge, FT)	MiniVLA (Bridge, FT, -VQ)	π_0 Reimplment (Bridge, FT)
Carrot Base	3/5	5/5	3/5	4/5	5/5	4/5	4/5
Knife Base	2/5	3/5	5/5	5/5	5/5	4/5	5/5
Carrot Color	2/5	4/5	4/5	2/5	2/5	2/5	2/5
Knife Color	2/5	3/5	0/5	0/5	0/5	0/5	0/5
Carrot Lift/Place	3/5	5/5	2/5	4/5	5/5	3/5	5/5
Knife Lift/Place	2/5	4/5	4/5	3/5	4/5	1/5	5/5
Carrot Counter	0/5	0/5	0/5	0/5	0/5	0/5	0/5
Knife Sink	1/5	2/5	1/5	3/5	0/5	0/5	3/5
Carrot Basketball	0/5	0/5	0/5	1/5	0/5	0/5	0/5
Knife Typo	1/5	4/5	4/5	4/5	1/5	1/5	4/5
Carrot in Sink	–	–	5/5	–	5/5	–	3/5
Rotate Knife	–	–	1/5	–	0/5	–	0/5
Carrot Distractors	3/5	5/5	3/5	3/5	3/5	2/5	4/5
Knife Distractors	1/5	3/5	2/5	3/5	5/5	3/5	4/5
Carrot Red Sink	3/5	3/5	1/5	3/5	3/5	1/5	5/5
Knife Red Sink	1/5	3/5	2/5	2/5	3/5	4/5	2/5
Carrot Orange Plate	2/5	3/5	2/5	3/5	4/5	0/5	3/5
Knife Orange Plate	4/5	4/5	2/5	4/5	5/5	5/5	5/5
Carrot Camera	0/5	0/5	0/5	0/5	1/5	0/5	0/5
Knife Camera	0/5	0/5	0/5	0/5	1/5	0/5	5/5
Carrot Farther	2/5	3/5	2/5	4/5	3/5	3/5	2/5
Carrot Start Sink	3/5	3/5	3/5	2/5	5/5	3/5	5/5
Knife Raised	0/5	2/5	3/5	4/5	4/5	3/5	1/5
Knife Right	2/5	3/5	3/5	5/5	4/5	2/5	5/5
Carrot Shorter Table	2/5	5/5	2/5	4/5	3/5	4/5	5/5
Knife Shorter Table	0/5	3/5	3/5	3/5	2/5	2/5	5/5
Baby Carrot	0/5	0/5	0/5	0/5	1/5	0/5	1/5
Small Knife	0/5	0/5	1/5	0/5	0/5	0/5	0/5
Ball	–	–	0/5	–	3/5	–	1/5
Pizza	–	–	1/5	–	0/5	–	0/5

TABLE VIII: Detailed generalization results for “put carrot” and “put knife”.

Condition Name	OpenVLA (Bridge, FT)	MiniVLA (Bridge, FT)	π_0 Reimplment (Bridge, FT)
Pot Base	3/5	5/5	5/5
Plate Base	3/5	4/5	4/5
Pot Color	4/5	1/5	1/5
Plate Color	0/5	0/5	0/5
Pot Lift/Place	0/5	2/5	1/5
Plate Lift/Place	0/5	0/5	0/5
Pot Sink	1/5	1/5	5/5
Plate Drying Rack	0/5	0/5	0/5
Pot Boiling	0/5	0/5	0/5
Plate Typo	0/5	0/5	0/5
Plate to Counter	0/5	0/5	0/5
Pot to Left	0/5	2/5	0/5
Pot Distractors	3/5	4/5	5/5
Plate Distractors	3/5	1/5	0/5
Pot Green Sink	3/5	3/5	4/5
Plate Green Sink	2/5	1/5	3/5
Gray Plate	4/5	3/5	3/5
Pot Camera	0/5	1/5	0/5
Plate Camera	0/5	0/5	0/5
Pot Left	3/5	0/5	0/5
Pot Angled	3/5	3/5	5/5
Plate Closer	0/5	3/5	4/5
Plate Counter	2/5	0/5	0/5
Pot Shorter Table	5/5	2/5	5/5
Plate Shorter Table	3/5	0/5	2/5
Thin Pot	3/5	0/5	1/5
Red Bowl	0/5	2/5	5/5
Cup	3/5	0/5	3/5
Spoon	0/5	0/5	0/5

TABLE IX: Detailed generalization results for “flip pot” and “put plate”.

Condition Name	OpenVLA (OXE)	OpenVLA (OXE, FT)	OpenVLA (Bridge, FT)	OpenVLA (Bridge, VQA, FT)	MiniVLA (Bridge, FT)	MiniVLA (Bridge, FT, No VQ)	π_0 Reimplment (Bridge, FT)
Carrot Color + Lift/Place	3/5	4/5	4/5	0/5	4/5	2/5	1/5
Knife Color + Lift/Place	3/5	4/5	0/5	0/5	0/5	0/5	0/5
Carrot Distractors + Orange Plate	1/5	4/5	2/5	3/5	3/5	0/5	3/5
Knife Distractors + Orange Plate	2/5	2/5	3/5	3/5	5/5	5/5	4/5
Carrot Farther + Shorter Table	3/5	3/5	3/5	3/5	2/5	3/5	3/5
Knife Right + Shorter Table	2/5	3/5	0/5	4/5	3/5	3/5	5/5

TABLE X: Detailed results for our compositional evaluation.

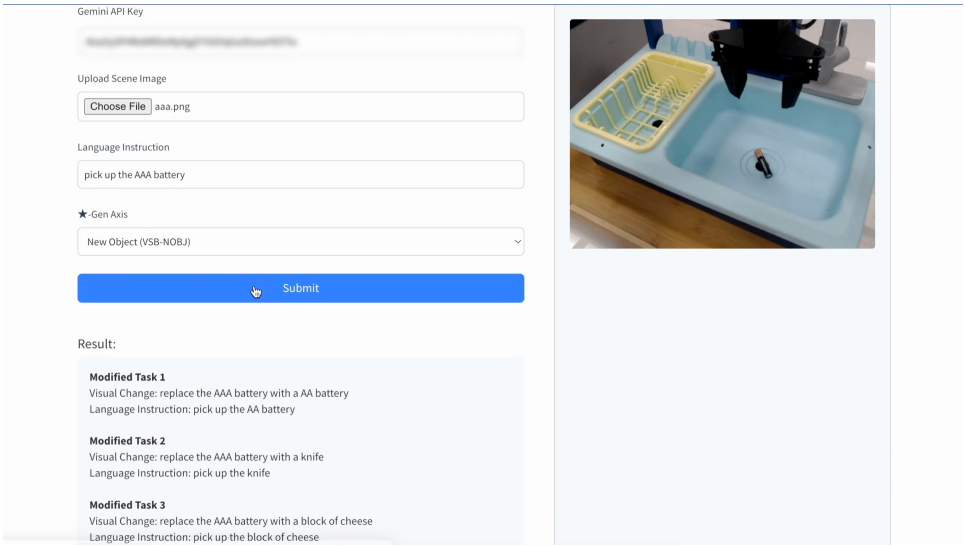


Fig. 12: Example of using our demonstration to generate *New Object* perturbations for the base task "pick up the AAA battery".

APPENDIX D
AUTOMATIC BENCHMARK DESIGN

We intend for ★-Gen to provide guidance for more extensive and fine-grained evaluations of generalist robot policies. However, we acknowledge that the evaluation conditions considered in BridgeV2-★, while based on ★-Gen, were still chosen using human oversight. We believe that it would be beneficial to limit human involvement when constructing future evaluations, because it can require a significant amount of human effort to design a thorough set of evaluation conditions, and human involvement can result in bias, even if unintentional.

Therefore, we make a preliminary step towards reducing human involvement in robot benchmark design by using foundation models to automatically propose evaluation conditions. In particular, we condition a vision-language model (VLM) on a base task, and ask it to propose perturbations of the base task to evaluate a given axis of generalization from ★-Gen.

To do this, we prompt the VLM with an initial scene image for the base task, as well as a text prompt that consists of the base task language instruction, and directions for how to modify the task according to a given axis from ★-Gen. If the axis is **visual**, we ask the VLM to specify the perturbation as an instruction for an image-editing model that would perform the perturbation. If the axis is **semantic**, we ask for the VLM to generate new language instructions for the perturbed tasks. The VLM can generate both image edits and new language instructions if the given axis is both **visual** and **semantic**.

We developed an interactive demonstration of this system where users can provide their own base tasks (as initial scene images and language instructions) on our website. We use Gemini 2.0 Flash as our VLM. For more reliable generations, we configure Gemini to generate perturbed tasks as JSON, using a specified schema. We instantiate the system for the following 5 ★-Gen axes:

1) *Visual Task Object* (V-OBJ)
2) *Object Properties* (S-PROP)
3) *Object Poses* (VB-POSE)
4) *Action Verbs* (SB-VRB)

> This is an image of a scene where a robot is to complete the task "put carrot on plate". Suggest 3 changes to the task that each involve changing a single task-relevant object to a new object with a different visual appearance, semantic description, and physical characteristics. Do this by providing 3 updated language instructions for each of the modified tasks, and corresponding text prompts to an image-editing model that would each perform a single change to the scene to create the modified task. Remember to only change one object, and to only change an object that is involved in the task.

Fig. 13: Example text prompt for generating perturbations of the base task "put carrot on plate" for the axis *New Object*.

```
import { SchemaType } from "@google/generative-ai";

const visualSemanticSchema = {
    description: "Visual and language instruction changes for a task.",
    type: SchemaType.ARRAY,
    items: {
      type: SchemaType.OBJECT,
      properties: {
        visualChange: {
          type: SchemaType.STRING,
          description: "Text prompt for an image-editing model to modify the task",
          nullable: false,
        },
        languageChange: {
          type: SchemaType.STRING,
          description: "Updated language instruction for the modified task",
          nullable: false,
        },
      },
      required: ["visualChange", "languageChange"],
    },
};
```

Fig. 14: Node.js JSON schema used for Gemini API to generate perturbations that are both **visual** and **semantic**, such as those for the axis *New Object*.

5) *New Object* (VSB-NOBJ)

In Fig. 12, we provide an example screenshot of the demonstration used to generate perturbations for a base task that was not used in BridgeV2-★. In Fig. 13, we provide an example text prompt used for the axis *New Object*, and in Fig. 14 we provide the Node.js JSON schema used for this axis.