DeliGrasp: Inferring Object Properties with LLMs for Adaptive Grasp Policies

William Xie[1], Jensen Lavering[1], Nikolaus Correll[1]

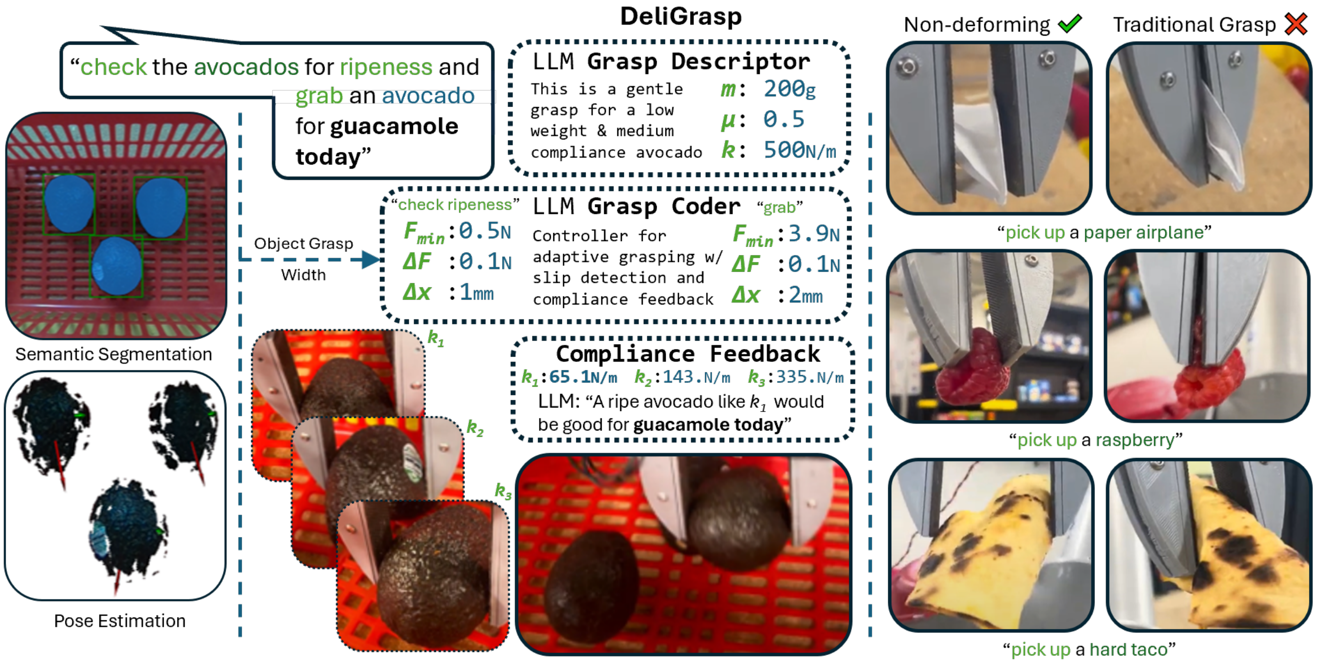
DeliGrasp
"check the avocados for ripeness and grab an avocado for guacamole today"
LLM Grasp Descriptor
This is a gentle grasp for a low weight & medium compliance avocado
m: 200g
μ: 0.5
k: 500N/m
Semantic Segmentation
Object Grasp Width
"check ripeness"
LLM Grasp Coder
"grab"
Fmin: 0.5N
ΔF : 0.1N
Δx : 1mm
Controller for adaptive grasping w/ slip detection and compliance feedback
Fmin: 3.9N
ΔF : 0.1N
Δx : 2mm
k1
k2
k3
Compliance Feedback
k1: 65.1N/m k2: 143.N/m k3: 335.N/m
LLM: "A ripe avocado like k1 would be good for guacamole today"
Pose Estimation
Non-deforming ✓
Traditional Grasp ✗
"pick up a paper airplane"
"pick up a raspberry"
"pick up a hard taco"

Fig. 1: Large language models (LLMs) have rich physical knowledge about worldly objects, but cannot directly reason robot grasps for them. Paired with open-world localization and pose estimation (left), our method (middle), queries LLMs for the salient physical characteristics of mass, friction, and compliance as the basis for an adaptive grasp controller. *DeliGrasp* policies successfully grasp delicate and deformable objects (right). These policies also produce compliance feedback as measured spring constants, which we leverage for downstream tasks like picking ripe produce (middle).

***Abstract*— Large language models (LLMs) can provide rich physical descriptions of most worldly objects, allowing robots to achieve more informed and capable grasping. We leverage LLMs' common sense physical reasoning and code-writing abilities to infer an object's physical characteristics—mass m, friction coefficient μ, and spring constant k—from a semantic description, and then translate those characteristics into an executable adaptive grasp policy. Using a current-controllable, two-finger gripper with a built-in depth camera, we demonstrate that LLM-generated, physically-grounded grasp policies outperform traditional grasp policies on a custom benchmark of 12 delicate and deformable items including food, produce, toys, and other everyday items, spanning two orders of magnitude in mass and required pick-up force. We also demonstrate how compliance feedback from *DeliGrasp* policies can aid in downstream tasks such as measuring produce ripeness. Our code and videos are available at: `https://deligrasp.github.io`**

I. INTRODUCTION

Large language models (LLMs) are able to supervise robot control and learning in manipulation from high-level step-by-step task planning [1]–[3] to low-level motion planning [4], [5]. LLMs additionally aid in robot manipulation via understanding a given object's semantic properties and delineating appropriate grasp positions conditioned on those semantic affordances [6]–[8].

These works usually assume that the acts of "picking" and "placing" are straightforward tasks. This is not the case for contact-rich manipulation. Existing methods for LLM-supervised robot control cannot account for tasks like grasping a paper airplane or origami, deformable plastic bags containing rice or dry noodles, or particularly ripe fruits and vegetables. Current methods for grasping such delicate objects require custom sensing solutions combined with adaptive force-control models [9]–[11] and are tested on a limited set of items. LLMs thus provide an opportunity to leverage their common-sense physical reasoning [12] to produce grasp skills which are both force-adaptive and for the open-world. This is particularly important for semi-structured environments like supermarkets that are subject to a constantly rotating stock or dealing with loose food items such as fruits, vegetables, and pastries that come in a large variety of changing shapes.

We propose *DeliGrasp*, an extension of LLM-supervised

[1] All authors are with the University of Colorado at Boulder, Boulder, CO. Corresponding email: `wixi6454@colorado.edu`

robot learning to contact-rich manipulation, and posit that LLMs can infer the physical characteristics of any arbitrary gripper-object interaction, including mass, spring constant, and friction. We formulate an adaptive grasp controller with slip detection derived from the inferred characteristics, endowing LLMs embodied with any current-controllable gripper with adaptive, open-world grasp skills for objects spanning a range of weight, size, fragility, and compliance.

We pair *DeliGrasp* with an open-world localization pipeline which, given an "object description," identifies the object and an initial grasp position. The same "object description" and associated "grasp verb" are the inputs to *DeliGrasp*, which generates a grasp policy: executable Python code controlling a gripper's compliance, force, and aperture as a complete grasp skill for the given description.

Experimentally, we observe that *DeliGrasp* performs successful, minimally-damaging grasps on a custom benchmark of delicate objects which traditional grasping methods are not capable of. We also demonstrate how LLMs can leverage compliance feedback from DeliGrasp policies in downstream tasks such as picking ripe produce. These generalizable, open-world grasping skills for minimal deformation as well as real sensor data can be used to address difficult manipulation tasks across environments and modes of HRI.

II. RELATED WORK

LLMs, with an internet-scale amount of common sense information, are able to combine high-level task knowledge, physical context, and robot affordances. For task and motion planning, LLMs can generate navigation and pick-and-place instructions to complete complex and novel tasks [1]. LLM code-writing further augments robot capabilities with closed-loop control [2], new skill generation [3], and translating language to robot parameters for low-level dynamic control [4]. In conjunction with learning methods [8], [13], LLMs have also shown semantic understanding of grasping such as identifying appropriate grasp locations [6], [7], [14] and have been fine-tuned on object's physical properties to improve manipulation performance [15], but neither method addresses grasping dynamics requisite for delicate grasping.

Grasping delicate objects can be achieved by relying on the compliance of soft grippers, which still requires semantic information for control if these grippers must grasp both lightweight and heavy objects [16]. In this paper, we adopt a control strategy that relies on interaction force and slip velocity measurements from a rigid gripper similar to [9], [10], [17], [18] to perform minimally deforming grasps and to measure spring constants [19]. We use semantic object descriptions together with an LLM to (1) select appropriate parameters for robust and damage-free grasping of delicate objects, and (2) use tactile inspection to classify relative fruit ripeness and connect them back to common-sense policies on how they can be consumed.

III. METHODS

As there does not exist a dataset focusing on delicate objects, we construct our own preliminary delicate objects dataset. We source dataset objects primarily from supermarkets, kitchens, and food pantries. Tabulated object mass and object-specific thresholds for unsuccessful, "deforming" grasps are shown in Appendix A.

Our pipeline takes as input a "grasp verb" and an "object description". For the delicate objects benchmark, we use "pick" as the "grasp verb," and the specific "object descriptions" used are provided in A. Our perception method, adapted from [6], takes the "object description" and semantically segments the object from an RGB-D image with OWL-ViT [20] and "Segment Anything" [21]. We crop the corresponding depth image with the generated segmentation mask, produce a point cloud object representing the segmented object, and perform Principal Component Analysis to compute a grasp pose that is aligned with the first three principal axes as well as a minimum object grasp width [22].

For grasping, we use a UR5 robot arm with the open-source MAGPIE gripper [23], which has a palm-integrated Intel RealSense D-405 camera. The gripper is powered by two separate Dynamixel AX-12 motors, which allow maximum current control, allowing us to implement simple torque control (default output force is 4 *N*). Translating torque via a four-bar linkage, the gripper force can be controlled and sensed in a range from 0.15 *N* to 16 *N*, and has an aperture range from 106 *mm* fully open to 0 *mm* fully closed (default closure speed of 100 $\frac{mm}{s}$).

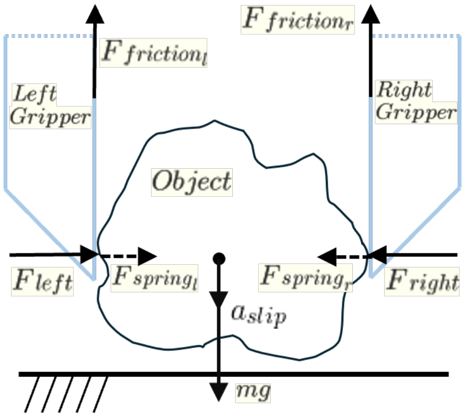

Fig. 3: Free body diagram describing gripper interactions with an object at rest, adapted from [9].

A. Grasp Force Modeling

Fig. 3 shows the interaction between our gripper and an arbitrary object with mass m and spring constant (compliance) k. The gripper exerts a composite applied gripper force $F_a = F_{left}+F_{right}$ that leads to a frictional force $F_f = \mu F_a$, where μ is the Coulomb friction coefficient between the gripper and object [22] that counteracts the force of gravity mg (g is the gravitational constant). For compliant objects, approximated as ideal springs, we can additionally describe the left and right gripper forces $F_{l,r} = F_{spring_{l,r}} = kx_{l,r}$, where x is the compression of the grasped object.

Typically, object slip within a gripper can only be detected after the gripper grasps an object at rest, the gripper begins some upward acceleration a_{lift}, and an object begins slipping with some downward acceleration a_{slip}. Increasing F_a to account for ma_{lift} yields an adaptive minimum applied grasp force $F_{min} = F_a$ which prevents slip and is minimally deforming [17]: $F_{min} = \frac{m(g+a_{lift})}{\mu}$ Conversely, when an object is slipping with a_{slip}, the applied force $F_{a_{slip}} = \frac{m(g-a_{slip})}{\mu}$ [9]. And when $a_{lift,slip}$ are 0 and the gripper and object are at static equilibrium, $F_{min} = \frac{mg}{\mu}$. F_{min} can then be arranged in relation to these quantities:

$$F_{a_{slip}} < \frac{mg}{\mu} \leq F_{min} \tag{1}$$

For the delicate objects dataset, we estimate a minimum applied grasping force $F_{min} = \frac{mg}{\mu}$ with a conservative, uniform Coulomb friction coefficient of $\mu = 0.33$ (A).

B. Problem Statement

Here m and μ, which determine successful (non-slip) and minimally deforming grasp forces, as well as spring constant k, are unknown values. Suppose then that a highly capable reasoning agent can reliably determine values of said m, μ, thereby approximating the lower bound of F_{min}, and k. These values form the basis of our force-adaptive algorithm for minimally deforming grasps on delicate objects.

Given a gripper driven by a current-controlled servo motor, we define a closed-loop force compensation controller: starting from an estimated target aperture that corresponds to the estimated object's width w from RGB-D data, we increase the gripper output force F_{out} and decreasing gripper aperture x until sensing a contact force F_c greater than the target F_{min} [18]. To determine the gain terms, ΔF_{out} and Δx, i.e. how fast we close the gripper and ratchet up force, the controller uses the agent-determined k and Δx, we change F_{out} by $c \cdot k\Delta x$, where $c = 0.1$ is a dampening constant. We describe the controller in Algorithm 1.

Algorithm 1 Adaptive Grasping with Slip Detection

```
$F_c = \texttt{SetGripper}(x = w, 0)$
while $F_c \leq F_{min}$ do
    $F_{out} \mathrel{+}= c \cdot k\Delta x$
    $x \quad -= \Delta x$
    $F_c = \texttt{SetGripper}(x, F_{out})$
```

We task an LLM (GPT-4) with predicting these quantities for an arbitrary object. If such predictions are accurate, performing minimally deforming grasps on a variety of objects becomes algorithmically simpler and removes the need for parameter tuning [18] and additional gripper sensors [9], [11]. By default, we instruct the LLM to compute $F_{min} = \frac{mg}{\mu}$ and we do not account for a_{lift}, choosing to err closer to object slip than deformation. However, we also provide the LLM with agency to deviate from the default F_{min} depending on the "grasp verb" provided.

We query the LLM to define and generate grasps with a dual-prompt structure similar to that of Language to Rewards [4], with an initial grasp "descriptor" prompt which produces a structured description, which the subsequent "coder" prompt translates into an executable Python grasp policy that modulates gripper compliance, force, and aperture according to Alg. 1. See Appendix D and E for the full prompts.

IV. EXPERIMENTS

Our robot experiment platform consists of a UR5 robot arm and a MAGPIE gripper looking top-down on a table with a palm-integrated Intel RealSense D-405 camera. We benchmark *DeliGrasp* (*DG*) against three grasp policies: closing the gripper fully or until it is output force limited (F.L.), closing the gripper to the visual object width determined by our perception method, and an ablated *DeliGrasp* policy which directly generates F_{min}, ΔF_{out}, and Δx without first reasoning an object's physical properties (*DG-D*). We employ each grasp policy 10 times, objects placed randomly within a 30 x 45 cm bounding area on a table, and do not record attempts which receive faulty poses from perception.

TABLE I: Successful Minimally Deforming Grasps on Delicate and Deformable Objects (*10 trials per object*)

ID	Object	*DG*	*DG-D*	Visual Width	F.L.
1	Paper Airplane	***10***	***10***	0	0
2	Cup (empty)	***10***	***10***	***10***	0
3	Dried Yuba	**9**	7	3	0
4	Raspberry	**9**	8	0	0
5	Hard Taco	**9**	7	5	0
6	Mandarin	***10***	***10***	***10***	***10***
7	Stuffed Toy	7	8	0	**10**
8	Cup (water)	**10**	8	3	4
9	Bag (noodles)	**7**	4	0	5
10	Avocado	**9**	7	4	0
11	Spray Bottle	**6**	5	0	3
12	Bag (rice)	0	0	0	**2**

DeliGrasp and the ablated direct *DeliGrasp* both perform successful, minimally-deforming grasps that the visual-only and hardware-limited grasp policies are not capable of on objects like the paper airplane and raspberry. *DeliGrasp* dominates hardware-limited and vision-only grasps in 8 out of 12 objects, and is better or on par in 10 out of 12 objects (full *DeliGrasp* details are provided in App. B)

We look closely at failure modes to better understand the performance of each policy. Given two bounds of failure—*slip* grasps and *deforming* grasps—we observe that, categorically, force-limited grasps *deform* and vision-only grasps *slip*. Force-limited grasps succeed only with robust, compliant objects. Vision-only grasps succeed when the objects are relatively stiff, but slight deviations in grasp position alter the grasp width and contact area, leading to failures with the taco, yuba, water cup, and avocado.

As we conduct a small number of trials, we cannot conclusively explain the performance differences between the full and ablated *DeliGrasp* policies, however, the F_{min} terms estimated by each *DeliGrasp* policy, shown in Table II, potentially clarify such differences. *DeliGrasp* primarily *slips* and performs only 3 *deforming* grasps, all of which are minor, but sufficiently damaging, occurring once each time with the yuba, raspberry, taco. For each of those

TABLE II: Full vs. Ablated *DeliGrasp* Estimated F_{min}

ID	$\frac{mg}{\mu}$ (N)	F_{max} (N)	F_{DG} (N)	F_{DG} *Err.* (N)	F_{DGD} (N)	F_{DGD} *Err.* (N)
1	0.02	0.03	0.10	*0.07*	0.15	*0.12*
2	0.11	0.13	0.25	*0.13*	0.5	*0.38*
3	0.16	0.20	0.39	*0.21*	0.5	*0.32*
4	0.18	0.22	0.06	*0.14*	0.2	*0.02*
5	0.44	0.55	0.98	*0.48*	1.5	*1.00*
6	1.65	2.05	1.88	*0.20*	1.25	*0.60*
7	2.18	2.71	0.61	*1.84*	1.5	*0.95*
8	3.12	3.88	4.08	*0.58*	3	*0.50*
9	5.62	6.98	12.3	*6.00*	4	*2.30*
10	6.00	7.46	3.92	*2.81*	1	*5.73*
11	7.36	9.14	12.2	*3.95*	5	*3.25*
12	26.5	32.9	19.6	*10.1*	8	*21.7*

items, the ablated *DeliGrasp-Direct* estimates a higher F_{min} and *deforms* the same items at a higher rate, though also with minor deformations. Conversely, the ablated *DeliGrasp* estimates a lower F_{min} for the cup of water, bag of noodles, avocado, and resulting grasps *slip* more. Both policies *slip* at high rates on the stuffed animal (grasped by the tail), spray bottle, bag of noodles, and bag of rice. These objects are non-linearly and/or very compliant and high volume, exerting lever forces on the gripper that likely exceed F_{min}.

DeliGrasp on average overestimates object mass by a factor of 1.5, and underestimates mass only for one object (App. B), and we do not empirically measure spring constant of objects. We do not comprehensively measure the consistency of *DeliGrasp* m and μ predictions, but we observe that re-generated m, μ predictions do not deviate significantly, resulting in a range of $\frac{2}{3}F_{min} \rightarrow \frac{4}{3}F_{min}$ for the initial F_{min}. While this is a large range, *DeliGrasp Direct* is more volatile, producing an even larger range, $\frac{1}{3}F_{min} \rightarrow 3F_{min}$. Since mass, a common quantity, is more reliably overestimated by an LLM, resulting adaptive grasp policies parameterized from mass are more robust than policies directly parameterized by an LLM with more erroneous predictions.

A. Sensorizing DeliGrasp to Pick Ripe Produce

DeliGrasp explicitly experimentally measures a spring constant k while performing Alg 1: the adaptive grasp controller checks for contact force F_c with each small motion Δx, and $k = \frac{F_c}{\Delta x}$. Though this measured k (App. B) shows that *DeliGrasp* is not able to accurately map object-specific knowledge to spring constants, it can be utilized as a compliance sensor in tasks such as appraising the ripeness (a correlate of compliance) of produce items. We perform this measurement on two assortments of produce: avocados and tomatoes. When receiving the "grasp verb" of "check" or "inspect for ripeness", *DeliGrasp* bypasses the default minimum grasp force value of $\frac{mg}{\mu}$ and manually sets a lower F_c of 0.5 N for avocados and 0.2 N for tomatoes.

Qualitatively, the relative ordering of the measured k for each item type corresponds with judgments from our own human hand grasps. We query an LLM (GPT-3.5) with the item name and compliance data, and show in Table III that it is able to reason over and answer abstract, relative questions about when to eat them. Further qualitative questions such as "how" to eat a specific item "right now" yield dishes appropriate for the ripeness of the produce (App. C).

TABLE III: Measured Produce Compliance and LLM Reasoning

Query	Object	k_1 (N/m)	k_2 (N/m)	k_3 (N/m)	k_4 (N/m)
	Avocado	65.1	143.5	335.4	—
	Tomato	271.6	331.8	606.2	1559.
At a given time, which should I eat?					
Next week?	Avocado			✓	
	Tomato				✓
Right now?	Avocado	✓			
	Tomato	✓	✓		
Three Days?	Avocado		✓	✓	
	Tomato	✓	✓	✓	
When will this go bad? (days)					
	Avocado	3–5	4–6	5–7	
	Tomato	1–2	2–4	3–5	7+

V. DISCUSSION

We introduce DeliGrasp, which uses LLMs to 1) infer semantic, common sense physical information and then 2) generate adaptive grasp policies to grasp a variety of delicate and deformable items such as ripe produce, that traditional parallel jaw grippers cannot grasp. These inferred m, μ, k characteristics enable more consistent control than direct estimation of grasping forces, and we also leverage *DeliGrasp*'s compliance feedback to measure produce ripeness.

In our experiments, we generate each grasp policy only once without feedback, as typically a grasp failure leads to irreversible deformation. However, finetuning on iterative feedback in LLM-supervised human-robot control may improve initial policy performance, which is of high importance for delicate objects, and generate new, complex policies [5]. In our future work we plan to explore other potential downstream tasks in dense, cluttered, and unstructured settings like supermarkets, home kitchens, and other such representative domains, as well as the semantic spaces of "grasp verbs" and "object descriptions" and how they affects generated policies. Our adaptive grasping algorithm is gripper-agnostic, but we have not validated it on other end-effectors. Furthermore, the controller is employed only while robot is at rest, but if synchronized with robot motion, our algorithm can be extended for in-motion slip-detection [9], [10].

Additionally, we can use multimodal foundation models like GPT-4V for *DeliGrasp*, thereby generating "grasp verbs" and "object descriptions" from images, rather than using hand-annotated inputs, and grounding grasp descriptions in both visual and semantic physical knowledge. Mulitmodal feedback and learning, in addition to augmented grasp controllers and model fine-tuning on object physical concepts [15], can then improve predictions and measurements of mass, friction, and other physical properties.

REFERENCES

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as I can, Not As I Say: Grounding language in robotic affordances," *arXiv:2204.01691*, 2022.

[2] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9493–9500.

[3] M. G. Arenas, T. Xiao, S. Singh, V. Jain, A. Z. Ren, Q. Vuong, J. Varley, A. Herzog, I. Leal, S. Kirmani, D. Sadigh, V. Sindhwani, K. Rao, J. Liang, and A. Zeng, "How to prompt your robot: A promptbook for manipulation skills with code as policies," in *2nd Workshop on Language and Robot Learning: Language as Grounding*, 2023. [Online]. Available: https://openreview.net/forum?id=T8AiZj1QdN

[4] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, brian ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=SgTPdyehXMA

[5] J. Liang, F. Xia, W. Yu, A. Zeng, M. G. Arenas, M. Attarian, M. Bauza, M. Bennice, A. Bewley, A. Dostmohamed, C. K. Fu, N. Gileadi, M. Giustina, K. Gopalakrishnan, L. Hasenclever, J. Humplik, J. Hsu, N. Joshi, B. Jyenis, C. Kew, S. Kirmani, T.-W. E. Lee, K.-H. Lee, A. H. Michaely, J. Moore, K. Oslund, D. Rao, A. Ren, B. Tabanpour, Q. Vuong, A. Wahid, T. Xiao, Y. Xu, V. Zhuang, P. Xu, E. Frey, K. Caluwaerts, T. Zhang, B. Ichter, J. Tompson, L. Takayama, V. Vanhoucke, I. Shafran, M. Mataric, D. Sadigh, N. Heess, K. Rao, N. Stewart, J. Tan, and C. Parada, "Learning to learn faster from human feedback with language model predictive control," *arXiv:2402.11450*, 2024.

[6] R. Mirjalili, M. Krawez, S. Silenzi, Y. Blei, and W. Burgard, "Langrasp: Using large language models for semantic object grasping," *arXiv:2310.05239*, 2023.

[7] S. Jin, J. Xu, Y. Lei, and L. Zhang, "Reasoning grasping via multimodal large language model," *arXiv:2402.06798*, 2024.

[8] A. Rashid, S. Sharma, C. M. Kim, J. Kerr, L. Y. Chen, A. Kanazawa, and K. Goldberg, "Language embedded radiance fields for zero-shot task-oriented grasping," in *7th Annual Conference on Robot Learning*, 2023. [Online]. Available: https://openreview.net/forum?id=k-Fg8JDQmc

[9] Z. Ding, N. Paperno, K. Prakash, and A. Behal, "An adaptive control-based approach for 1-click gripping of novel objects using a robotic manipulator," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 4, pp. 1805–1812, 2019.

[10] M. Al-Mohammed, R. Adem, and A. Behal, "A switched adaptive controller for robotic gripping of novel objects with minimal force," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 1, pp. 17–26, 2023.

[11] Y. Gong, Y. Xing, J. Wu, and Z. Xiong, "Tactile-Based Slip Detection Towards Robot Grasping," in *Intelligent Robotics and Applications*, H. Yang, H. Liu, J. Zou, Z. Yin, L. Liu, G. Yang, X. Ouyang, and Z. Wang, Eds. Singapore: Springer Nature Singapore, 2023, pp. 93–107.

[12] Y. Wang, J. Duan, D. Fox, and S. Srinivasa, "NEWTON: Are large language models capable of physical reasoning?" in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9743–9758. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.652

[13] H.-S. Fang, C. Wang, H. Fang, M. Gou, J. Liu, H. Yan, W. Liu, Y. Xie, and C. Lu, "Anygrasp: Robust and efficient grasp perception in spatial and temporal domains," *IEEE Transactions on Robotics (T-RO)*, 2023.

[14] C. Tang, D. Huang, W. Ge, W. Liu, and H. Zhang, "GraspGPT: Leveraging semantic knowledge from a large language model for task-oriented grasping," *arXiv:2307.13204*, 2023.

[15] J. Gao, B. Sarkar, F. Xia, T. Xiao, J. Wu, B. Ichter, A. Majumdar, and D. Sadigh, "Physically grounded vision-language models for robotic manipulation," 2024.

[16] N. Farrow, Y. Li, and N. Correll, "Morphological and embedded computation in a self-contained soft robotic hand," *arXiv preprint arXiv:1605.00354*, 2016.

[17] H. Hasegawa, Y. Mizoguchi, K. Tadakuma, A. Ming, M. Ishikawa, and M. Shimojo, "Development of intelligent robot hand using proximity, contact and slip sensing," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 777–784.

[18] K. Sullivan, H. Chizeck, and A. Marburg, "Using a rigid gripper on objects of different compliance underwater," in *OCEANS 2022, Hampton Roads*, 2022, pp. 1–4.

[19] T. M. Caldwell, D. Coleman, and N. Correll, "Optimal parameter identification for discrete mechanical systems with application to flexible object manipulation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 898–905.

[20] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen, X. Wang, X. Zhai, T. Kipf, and N. Houlsby, "Simple open-vocabulary object detection with vision transformers," *arXiv:2205.06230*, 2022.

[21] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023.

[22] N. Correll, B. Hayes, C. Heckman, and A. Roncone, *Introduction to autonomous robots: mechanisms, sensors, actuators, and algorithms*. MIT Press, 2022.

[23] N. Correll, D. Kriegman, S. Otto, and J. Watson, "A versatile robotic hand with 3d perception, force sensing for autonomous manipulation," *arXiv:2402.06018*, 2024.

APPENDIX

A. Delicate Objects Physical Characteristics

For objects with empty entries in the “Input Phrase” column, we do not modify or add descriptions beyond the name of the object. The squeeze bottle is a uniquely difficult object for a parallel jaw gripper to grasp. It is extremely sensitive to grasp position as it has high stiffness at all but the “squeeze” point. However, if it is squeezed too hard, the bottle will spill. The bag of rice is very heavy, though theoretically within the possible grasping force. However due to its extreme compliance, successful grasps require as close to full-finger contact.

TABLE IV: Delicate and Deformable Object Properties

ID	Object	Width (mm)	Mass (g)	$\frac{mg}{\mu}$ (N)	F_{max} (N)	“Object Description” Input to LLM & VLM	Invalidating Deformation
1	Paper Airplane	20	0.8	0.02	0.03	—	crumples
2	Cup (empty)	75	3.6	0.11	0.13	”empty paper cup”	crumples, creased
3	Dried Yuba	30	5.5	0.16	0.20	”yuba (dried tofu skin)”	cracks, shatters
4	Raspberry	20	6	0.18	0.22	—	juices, torn
5	Hard Taco	65	15	0.44	0.55	”hard-shell tortilla”	cracks, broken
6	Mandarin	50	56	1.65	2.05	—	inelastic deformation
7	Stuffed Toy	28	74	2.18	2.71	”tail of a stuffed animal”	inelastic deformation
8	Cup (water)	75	106	3.12	3.88	”paper cup filled with water”	spillage
9	Bag (noodles)	90	191	5.62	6.98	”plastic bag containing dried noodles”	cracks
10	Avocado	60	204	6.00	7.46	—	inelastic deformation
11	Spray Bottle	50	250	7.36	9.14	”squeeze bottle filled with water”	spillage
12	Bag (rice)	80	900	26.49	32.9	”plastic bag containing rice”	N/A

B. Full Details of DeliGrasp Performance on Delicate Objects Dataset

Computed ΔF_{out} which are less than 0.01 N are set to 0.01 N. Computed F_{min} values below the force sensing threshold of 0.15 N are set to that threshold when checking for slip or setting output force.

TABLE V: DeliGrasp Performance on Delicate Objects Dataset (10 trials)

		LLM Inferred m, μ, k and Downstream Terms						Mean Values of Experimentally Produced Terms					
ID	Success	m (g)	μ	k (N/m)	F_{min} (N) $= \frac{mg}{\mu}$	ΔF_{out} (N)	Δx (mm)	x_{final} (mm)	x_{goal} (mm)	F_{out} (N)	Time (s)	k_{max} (N/m)	k (N/m)
1	10	5	0.5	20	0.098	0.01	2	11.9	7.8	0.54	4.23	332.46	184.91
2	10	10	0.4	50	0.25	0.025	5	67.83	59.5	0.5	2.94	707.93	389.76
3	9	20	0.5	200	0.39	0.04	2	24.74	19.82	0.6	4.08	240.27	157.69
4	9	5	0.8	50	0.063	0.01	1	15.4	5.98	0.28	7.26	97.46	60.96
5	9	50	0.5	1000	0.98	0.2	2	53.17	46.64	1.59	3.19	342.05	188.39
6	10	150	0.8	500	1.88	0.1	2	45.13	39.83	2.12	3.08	236.41	151.83
7	9	50	0.8	100	0.61	0.02	2	9.79	4.19	0.97	5.29	308.73	192.77
8	9	250	0.6	200	4.08	0.04	2	58.23	53.3	4.36	4.27	958.61	373.24
9	7	500	0.4	300	12.3	0.15	5	71.72	61.11	12.77	2.89	7484.7	4324.33
10	9	200	0.5	500	3.92	0.1	2	50.88	46.4	4.39	3.04	729.11	298.66
11	6	500	0.4	150	12.2	0.03	2	41.94	35.08	12.6	3.64	3456.4	1383.11
12	0	1000	0.5	200	19.6	0.1	5	35	27.5	20.8	5.7	13671.	6192.2

We also show the corresponding values for the ablated *DeliGrasp-Direct* policy. We did not record time, experimental k values, or x_{final} in these experiments.

TABLE VI: Ablated DeliGrasp-Direct Performance on Delicate Objects Dataset (10 trials)

ID	Success	F_{min} (N)	ΔF_{out} (N)	Δx (mm)	x_{goal} (mm)	F_{out} (N)
1	10	0.15	0.05	1	4.48	0.98
2	10	0.5	0.2	1	57.4	0.922
3	7	0.5	0.3	2	8.6	1.14
4	8	0.2	0.1	1	6.4	0.83
5	7	1.5	0.2	1	49.2	2.3
6	10	1.25	0.5	5	32.4	2.01
7	8	1.5	0.5	2	3.0	3.2
8	8	3	1	5	49.6	3.18
9	4	4	1	4	52.8	6
10	7	1	1	5	47.2	2.2
11	5	5	2	2	25.0	7.1
12	0	8	2	5	105.	10

C. Ripeness Reasoning with LLMs

1. Asking ”how should I eat avocado k_3” yields: sliced avocado, grilled avocado, salad toppings, stuffed avocado
2. Asking ”how should I eat avocado k_1” yields: guacamole, avocado toast, sushi rolls, smoothies
3. Asking ”how should I eat tomato k_4” yields: sliced for burgers, grilled or roasted, salsa, stuffed
4. Asking ”how should I eat tomato k_1” yields: caprese salad, gazpacho, pasta sauce

We additionally observe that multimodal models like GPT-4V or OWL-ViT alone cannot reason ripeness over visual images. We query GPT-4V with an image of the same items 5 times and request a ripeness ranking. For the 4 tomatoes, GPT- 4V produces the same ranking each query, from most ripe to least ripe, of: $[k4, k2, k3, k1]$, reasoning entirely on the “redness” of the tomatoes. For the 3 avocados, it refuses to generate a ranking, citing a fundamental lack of sensory capabilities.

OWL-ViT when directly queried for ”the ripest” or ”least ripe” produce arbitrarily detects items (sometimes the same item) depending on the angle (and thus lighting, resolution, relative object size) of the captured image.

When we pair the spring constant data with the image in a query to GPT-4V, it ignores the visual characteristics and reasons solely on the spring data.:

Query:

```
I have labeled 3 avocados for you between 0, 1, 2 and provide their average spring constants from an
    initial ripeness check. Spring constants are inversely proportional to ripeness--a low spring
    constant means a soft object, and high means stiff. Combining this information with the image,
    please do the following tasks:
1. Estimate the ripeness of a ripe and unripe avocado.
2. Rank the given avocados by ripeness.
E. Using the estimated ripeness range, estimate when each one will go bad (avocados are stored in the
    open at room temperature).

Spring constant data:
Avocado 1: 65 N/m
Avocado 2: 143 N/m
Avocado 3: 335 N/m
```

Response:

```
To perform the tasks, we need to use the given spring constants and the visual clues from the image to
    estimate the ripeness and predict when each avocado will spoil. Typically, a ripe avocado has a
    spring constant at the lower end because it is softer, while a less ripe or unripe avocado has a
    higher spring constant because it is firmer.

0. Estimate the ripeness of a ripe and unripe avocado:
   - A ripe avocado typically has a low spring constant, indicating softness, perhaps similar to Avocado
     1, which has the lowest spring constant
   - An unripe avocado has a high spring constant, as it’s stiffer, like Avocado 3, with the highest
     spring constant

1. Rank the given avocados by ripeness:
   - Avocado 1, with the lowest spring constant range 65, would generally be the ripest.
   - Avocado 2, with a spring constant range 143, would be less ripe than avo1.
   - Avocado 3, having the highest spring constant range 335, would be the least ripe.

So the ripeness rank from ripest to least ripe would be:
Avocado 1 > Avocado 2 > Avocado 3
```

```
2. Estimate when each avocado will go bad:
   - Avocado 1, being the ripest, based on its lower spring constant, could spoil in about 2-3 days if
     stored at room temperature.
   - Avocado 2 is less ripe than avo1 and may take around 4-5 days to spoil at room temperature.
   - Avocado 3 is the least ripe, with quite a high spring constant and could take upwards of a week to
     spoil, potentially around 7-10 days at room temperature.

Please note that these estimates are quite rough and actual spoilage times can vary due to numerous
    factors such as specific room temperature, humidity, and the avocado’s initial condition. The visual
    clues are limited, but from the image, the physical characteristics that usually indicate ripeness
    (dark skin, slightly yielding to gentle pressure) don’t provide significant additional information
    beyond the spring constant data.
```

D. DeliGrasp Desciptor Prompt

The descriptor takes as input a “grasp verb”, “object description”, and “object width.” It outputs a structured summary of the object, how it should be grasped, and physical characteristics of the object.

```
Control a robot gripper with torque control and contact information.
This is a gripper with two independently actuated fingers.
The gripper’s parameters can be adjusted corresponding to the type of object that it is trying to grasp.
As well as the kind of grasp it is attempting to perform.
The gripper has a measurable max force of 16N and min force of 0.15N, a maximum aperture of 105mm and a
    minimum aperture of 1mm.
Some grasps may be incomplete, intended for observing force information about a given object.
Describe the grasp strategy using the following form:

[start of description]
* This {CHOICE: [is, is not]} a new grasp.
* This grasp should be [GRASP_DESCRIPTION: <str>].
* This is a {CHOICE: [complete, incomplete]} grasp.
* This grasp {CHOICE: [does, does not]} contain multiple grasps.
* This grasp is for an object with {CHOICE: [high, medium, low]} compliance.
* This grasp is for an object with {CHOICE: [high, medium, low]} weight.
* The object has an approximate mass of [PNUM: 0.0] grams
* The object has an approximate spring constant of [PNUM: 0.0] Newtons per meter.
* The gripper and object have an approximate friction coefficient of [PNUM: 0.0]
* This grasp should set the goal aperture to [PNUM: 0.0] mm.
* If the gripper slips, this grasp should close an additional [PNUM: 0.0] mm.
* Based on object mass and friction coefficient, grasp should initially set the force to [PNUM: 0.0]
    Newtons.
* If the gripper slips, this grasp should increase the output force by [PNUM: 0.0] Newtons.
* [optional] This grasp {CHOICE: [does, does not]} use the default minimum grasp force force.
* [optional] This grasp sets the force to  [PNUM: 0.0], which is {CHOICE: [lower, higher]} than the
    default initial force because of [GRASP_DESCRIPTION: <str>].
[end of description]

Rules:
1. If you see phrases like {NUM: default_value}, replace the entire phrase with a numerical value. If
    you see {PNUM: default_value}, replace it with a positive, non-zero numerical value.
2. If you see phrases like {CHOICE: [choice1, choice2, ...]}, it means you should replace the entire
    phrase with one of the choices listed. Be sure to replace all of them. If you are not sure about the
    value, just use your best judgement.
3. If you see phrases like [GRASP_DESCRIPTION: default_value], replace the entire phrase with a brief,
    high level description of the grasp and the object to be grasp, including physical characteristics
    or important features.
4. By default the minimum grasp force can be estimated by dividing the object weight (mass *
    gravitational constant) by the friction coefficient: (m*g/(mu)).
5. Using knowledge of the object and the grasp description, set the initial grasp force either to this
    default value or an appropriate value.
6. If you deviate from the default value, explain your reasoning using the optional bullet points. It is
    not common to deviate from the default value.
7. Using knowledge of the object and how compliant it is, estimate the spring constant of the object.
    This can range broadly from 20 N/m for a very soft object to 2000 N/m for a very stiff object.
8. Using knowledge of the object and the grasp description, if the grasp slips, first estimate an
    appropriate increase to the aperture closure, and then the gripper output force.
9. The increase in gripper output force the maximum value of (0.01 N, or the product of the estimated
    aperture closure, the spring constant of the object, and a damping constant 0.1:
    (k*additional_closure*0.0001)).
10. I will tell you a behavior/skill/task that I want the gripper to perform in the grasp and you will
    provide the full description of the grasp plan, even if you may only need to change a few lines.
    Always start the description with [start of description] and end it with [end of description].
11. We can assume that the gripper has a good low-level controller that maintains position and force as
    long as it’s in a reasonable pose.
12. The goal aperture of the gripper will be supplied externally, do not calculate it.
```

````
13. Do not add additional descriptions not shown above. Only use the bullet points given in the template.
15. Use as few bullet points as possible. Be concise.
````

E. DeliGrasp Coder Prompt

The coder takes as input the structured summary of the grasp and inferred characteristics and generates an grasp policy which implements the adaptive grasping algorithm 1 and compliance sensing.

~~~
We have a description of a gripper’s motion and force sensing and we want you to turn that into the
    corresponding program with following class functions of the gripper:
The gripper has a measurable max force of 16N and min force of 0.15N, a maximum aperture of 105mm and a
    minimum aperture of 1mm.

```
def get_aperture(finger=’both’)
```
finger: which finger to get the aperture in mm, of, either ’left’, ’right’, or ’both’. If ’left’ or
    ’right’, returns aperture, or distance, from finger to center. If ’both’, returns aperture between
    fingers.

```
def set_goal_aperture(aperture, finger=’both’, record_load=False)
```
aperture: the aperture to set the finger(s) to (in mm)
finger: which finger to set the aperture in mm, of, either ’left’, ’right’, or ’both’.
record_load: whether to record the load at the goal aperture. If true, will return array of (pos, load)
    tuples
This function will move the finger(s) to the specified goal aperture, and is used to close and open the
    gripper.
Returns a position-load data array of shape (2, n) --> [[positions], [loads]], average force, and max
    force after the motion.

```
def set_compliance(margin, flexibility, finger=’both’)
```
margin: the allowable error between the goal and present position (in mm)
flexibility: the compliance slope of motor torque (value 0-7, higher is more flexible) until it reaches
    the compliance margin
finger: which finger to set compliance for, either ’left’, ’right’, or ’both’

```
def set_force(force, finger=’both’)
```
force: the maximum force the finger is allowed to apply at contact with an object(in N), ranging from
    (0.1 to 16 N)
finger: which finger to set compliance for, either ’left’, ’right’, or ’both’

```
def check_slip(load_data, force, finger=’both’)
```
load_data: the position-load data array from set_goal_aperture
force: the force to check if the contact force is met (in N), which is set by set_force()
finger: which finger to check the contact force for, either ’left’, ’right’, or ’both’
Returns True if the contact force is not reached, meaning the gripper has slipped, False otherwise (the
    gripper has not slipped and has a good grasp).
Also returns the average and max force experienced by the gripper in the load data.


Example answer code:
```
from magpie.gripper import Gripper # must import the gripper class
G = Gripper() # create a gripper object
import numpy as np  # import numpy because we are using it below

new_task = {CHOICE: [True, False]} # Whether or not the task is new
# Reset parameters to default since this is a new, delicate grasp to avoid crushing the raspberry.
if new_task:
    G.reset_parameters()

# [REASONING]
goal_aperture = {PNUM: goal_aperture}
complete_grasp = {CHOICE: [True, False]}
# Initial force. The default value of object weight / friction coefficient.
initial_force = {PNUM: {CHOICE: [({PNUM: mass} * 9.81) / {PNUM: mu}, {PNUM: different_inital_force}] }}}
# [REASONING for initial force choice]
~~~

~~~
additional_closure = {PNUM: additional_closure}
# Additional force increase. The default value is the product of the object spring constant and the additional_closure, with a dampening constant 0.1.
additional_force = np.max([0.01, additional_closure * {PNUM: spring_constant} * 0.0001])

# Move quickly (without recording load) to a safe aperture that is wider than the goal aperture
G.set_goal_aperture(goal_aperture + additional_closure * 2, finger='both', record_load=False)

# [REASONING]
# [PREDICTION]
G.set_compliance(1, 3, finger='both')
G.set_force(initial_force, 'both')
load_data = G.set_goal_aperture(goal_aperture, finger='both')

# [REASONING]
# [PREDICTION]
curr_aperture = G.get_aperture(finger='both')
applied_force = inital_force
slip_threshold = initial_force
slippage, avg_force, max_force = G.check_slip(load_data, slip_threshold, 'both')

# record spring constants over slip detection
prev_aperture = curr_aperture
k_avg = []

while slippage:
  goal_aperture = curr_aperture - additional_closure
  if np.mean(avg_force) > 0.10: # low-pass filter force readings so we don't increase force when there is no contact
    applied_force += additional_force
  G.set_force(applied_force, 'both')
  print(f"Previous aperture: {curr_aperture} mm, Goal Aperture: {goal_aperture} mm, Applied Force: {applied_force} N.")
  load_data = G.set_goal_aperture(goal_aperture, finger='both')

  # Report data after each adjustment
  curr_aperture = G.get_aperture(finger='both')
  slippage, avg_force, max_force = G.check_slip(load_data, slip_threshold, 'both')

  # record spring constants over slip detection
  distance = np.abs(curr_aperture - prev_aperture)
  k_avg.append(np.mean(avg_force) * distance * 1000.0)
  prev_aperture = curr_aperture


if complete_grasp:
    curr_aperture = G.get_aperture(finger='both')
    G.set_goal_aperture(curr_aperture - additional_closure, finger='both', record_load=False)
    print(f"Final aperture: {curr_aperture} mm, Controller Goal Aperture: {goal_aperture} mm, Applied Force: {applied_force} N.")
else:
    G.open_gripper()
```

Remember:
1. Always format the code in code blocks. In your response all five functions above: get_aperture, set_goal_aperture, set_compliance, set_force, check_slip should be used.
2. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do not leave unimplemented code blocks in your response.
3. The only allowed library is numpy. Do not import or use any other library. If you use np, be sure to import numpy.
4. If you are not sure what value to use, just use your best judge. Do not use None for anything.
5. If you see phrases like [REASONING], replace the entire phrase with a code comment explaining the grasp strategy and its relation to the following gripper commands.
6. If you see phrases like [PREDICTION], replace the entire phrase with a prediction of the gripper's state after the following gripper commands are executed.
7. If you see phrases like {PNUM: default_value}, replace the value with the corresponding value from the grasp description.
8. If you see phrases like {CHOICE: [choice1, choice2, ...]}, it means you should replace the entire phrase with one of the choices listed. Be sure to replace all of them. If you are not sure about the value, just use your best judgement.
9. Remember to import the gripper class and create a Gripper at the beginning of your code.
10. Remember to check the current aperture after setting the goal aperture and adjust the goal aperture if necessary.
~~~

```
11. Before checking for slip, remember to create two new variables, applied_force and slip_threshold,
    set equal to the initial initial_force. Slip detection continues checking the unchanged
    slip_threshold, but the applied_force increases.
12. Remember to reassign the goal aperture to the current aperture after completing the slip check for
    complete grasps.
```