

SEMESTER PROJECT

Learning Algorithms and Systems Laboratory

LLM-guided constraint learning from demonstration

Student:

Marcus EDJOLO - 314564

Supervisor:

Baiyu PENG

Professor:

Aude BILLARD



Lausanne, Spring 2026

Contents

1	Introduction	1
2	Related work	2
2.1	Learning from Demonstration and constraint learning	2
2.2	LLMs in robotics	2
2.3	LLMs as priors for learning	2
2.4	The gap addressed by this project	2
3	Methods	3
3.1	Pipeline architecture overview	3
3.2	Generation of the safe demonstration	4
3.2.1	Variables to consider	4
3.2.2	Data points and trajectory generation	4
3.3	Constraint hypothesis by the LLM	7
3.4	Constraint learning	9
3.5	Evaluation against ground truth	11
4	Experiments and results	12
4.1	Experimental setup	12
4.2	Single-case walk-through: cup of water	12
4.3	Sample LLM outputs across categories	14
4.4	Aggregate results	15
4.5	Failure analysis	16
4.6	Per-task results	17
4.7	Consumability check: a minimal MPC	17
4.8	Discussion	18
5	Conclusion	20
A	System prompt (scaffolded variant)	23
B	Ground-truth annotation file (excerpt)	24
C	Per-task constraint figures	25

List of Figures

1	Pipeline flowchart.	3
2	Example orientation channels.	6
3	Illustration of two tasks	12
4	Safe demonstration of “Move a cup of water”.	13
5	Learned constraints for “Move a cup of water”.	14
6	Representative LLM outputs.	15
7	False-alarm failure mode.	16
8	Consumability check: MPC trajectory under learned constraints.	18
9	Per-task figure: glass of milk.	26
10	Per-task figure: bowl of soup.	27
11	Per-task figure: jar of beer.	28
12	Per-task figure: cup of coffee.	29
13	Per-task figure: glass vase.	30
14	Per-task figure: ceramic plate.	31
15	Per-task figure: egg.	32
16	Per-task figure: stack of books.	33
17	Per-task figure: open laptop.	34

1 Introduction

Robotic manipulation in unstructured environments has long faced a fundamental tension: the more capable and autonomous a robot becomes, the more difficult it is to specify exactly what it should and should not do. A factory robot performs a single repetitive motion that an engineer can hand-specify in detail; a domestic or service robot, by contrast, must handle an open-ended set of tasks (move the cup, carry the laptop, hand me the screwdriver) for which no engineer can enumerate the relevant safety constraints in advance. Until very recently, the only practical answer was to restrict deployment to controlled environments where the constraints could be hand-engineered case by case.

Large Language Models (LLMs) have begun to change this landscape, and they do so in two distinct ways. First: LLMs allow users to issue tasks in natural language, removing the requirement that the user know a robot-specific command vocabulary. Anyone can now say “move the cup of water to the table” and reasonably expect the robot to understand the request at the level of words. The second and much more important aspect for this project: LLMs carry an enormous amount of tacit physical knowledge acquired during pre-training on text describing the everyday world. A modern language model knows that water spills out of an open cup that is tilted past a certain angle or that thin glass shatters when impacted at speed. None of this knowledge is precise enough to drive a low-level controller, but it is the kind of qualitative information a human engineer would use to decide which physical variables matter for a given task. This second strength has been comparatively underexplored in robotic systems, which is the gap the present project addresses.

The structural bottleneck in classical Learning from Demonstration. Classical Learning-from-Demonstration (LfD) approaches can recover quantitative constraints from data very effectively – but only after a human expert has decided three things in advance: which kinematic variables the constraint operates on, what geometric type the safe region should be, and what mathematical family the boundary function should belong to. Only once this structural template is fixed can the data step fill in the numerical thresholds inside it. This pre-specification is the real bottleneck of classical LfD, examined further in the Related Work section: the data does not, on its own, tell you which variables to look at or what shape the safe region should take.

LLMs as priors for constraint learning. The core insight of this project is that an LLM is the natural source of the structural template that classical LfD would otherwise require from an expert. The LLM has read enough about the physical world to know, qualitatively, which variables matter and how a safe region should be shaped, but it cannot reliably produce calibrated numerical thresholds. Therefore, the natural division of labour is to ask the LLM for the template, and ask a demonstration for the numbers.

Proposed framework. This project implements an LLM-guided Learning from Demonstration framework. Given a natural-language instruction, the LLM acts as an automated constraint architect and proposes a structured hypothesis concerning the variables involved, the type of region they should stay in, and the mathematical family. A contrastive learning step then turns a safe demonstration and a set of synthesised unsafe states into the exact mathematical boundary, outputting a closed-form inequality $h(x) \leq 0$. Finally, a standard Model Predictive Control (MPC) framework is implemented to validate the efficacy of the learned constraints in a practical, downstream trajectory generation task.

2 Related work

This project sits at the intersection of three lines of work: classical Learning from Demonstration, the recent use of LLMs in robotics, and the broader idea of using LLMs as a source of prior knowledge for learning.

2.1 Learning from Demonstration and constraint learning

Learning from Demonstration (LfD) has been a very used model in robotics for over two decades [1, 2]. The idea is to let a robot acquire skills from human demonstrations rather than from explicit programming, it has produced a large family of methods including behaviour cloning, Dynamic Movement Primitives, and Gaussian Mixture Models for trajectory representation [3]. More recent work has shifted focus from learning motion from demonstrations to learning the constraints that demonstrations implicitly satisfy. Representative work in this thread include the work of [4] on learning safety constraints from positive-only data.

A limitation across this thread is that the structure of the constraint must be specified by an engineer before the data-driven step can begin. Constraint learning recovers the numerical boundary very effectively, but only within a structural template manually set. This is the bottleneck the present project aims to remove.

2.2 LLMs in robotics

The introduction of LLMs into robotics has produced rapid progress on a different problem: turning natural-language task descriptions into executable robot behaviour. SayCan [5] grounds an LLM’s task decomposition in a learned value function over a fixed set of robot skills. Code-as-Policies [6] uses an LLM to generate Python code that calls pre-existing motion primitives. VoxPoser [7] uses an LLM to specify cost and constraint maps in 3-D space, which are then optimised by a model-based planner.

What unites this line of work is that the LLM is used as a planner or orchestrator: it decides which skill to invoke, in what order, with what high-level parameters. The constraints that govern individual skills are still hand-engineered into the skills themselves. The LLM is not asked to supply constraint structure, only to choose among pre-existing options. All these still leave untouched the more fundamental problem of how a new task’s constraint structure is created in the first place.

2.3 LLMs as priors for learning

A closer relative to the present work, [8] explores using LLMs to generate constraint specifications for trajectory optimisation, though without a downstream data-driven step to fix the numerical thresholds.

The unifying observation in this thread is that LLMs are useful precisely because their pre-training has exposed them to vast amounts of qualitative human knowledge about how the physical and task world is organised. This knowledge is rarely quantitatively precise, but it is structurally rich and it is exactly the kind of knowledge an engineer would bring to a new task.

2.4 The gap addressed by this project

While these three threads touch on aspects of the problem, none provide a complete solution. Classical LfD requires pre-specified templates, LLM-in-robotics overlooks constraint structure, and LLM-as-prior methods focus on reward functions rather than safety bounds. This project bridges these domains. By combining LLM structural priors with a single safe demonstration to recover explicit safety constraints, we create a structured output ready for downstream planning.

3 Methods

3.1 Pipeline architecture overview

The core objective of this framework is to bridge the high-level reasoning capabilities of Large Language Models with the quantitative mathematical representations required by robotic planners. To achieve this without relying on manual parameterisation, the system implements a fully automated, end-to-end pipeline operating in five distinct phases. Figure 1 gives a block-diagram overview of the data flow; the rest of this section describes each phase in turn.

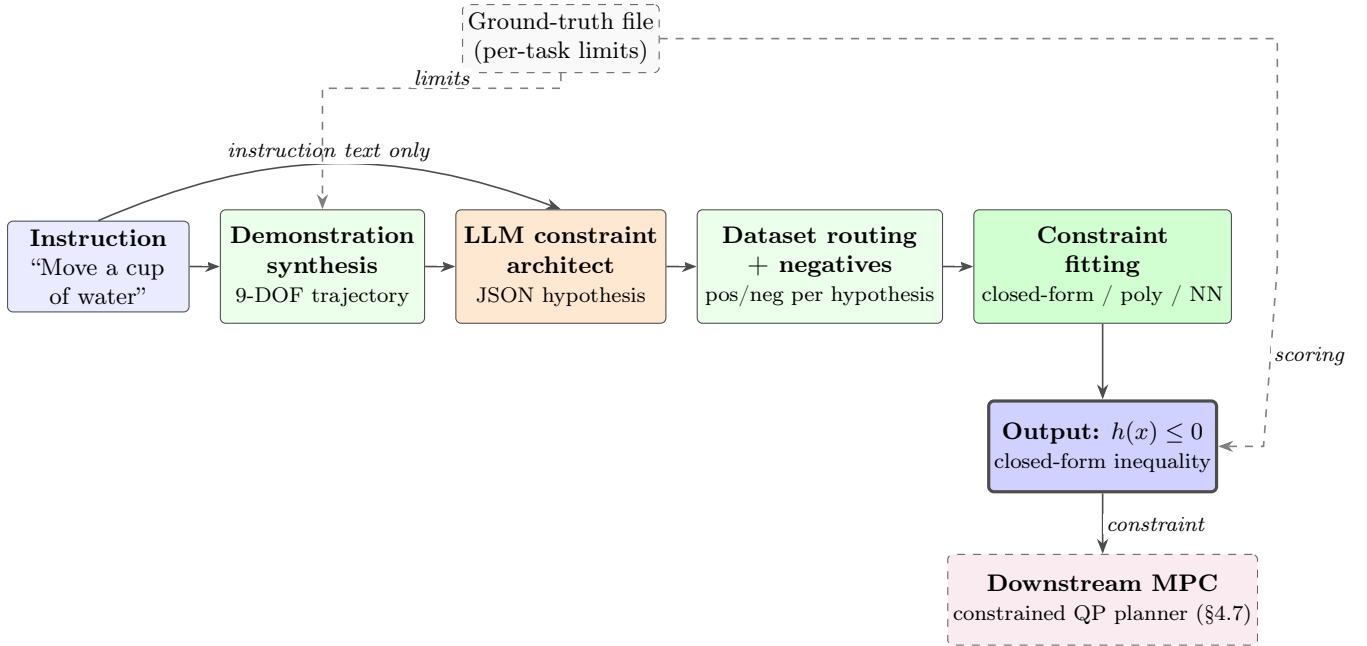


Figure 1: Block diagram of the proposed pipeline. The five main phases (blue/green boxes, left to right) operate on the natural-language instruction as input and produce a closed-form safety inequality $h(x) \leq 0$ as output. The single LLM-driven phase is highlighted in orange; all other phases are data-driven. The hand-annotated ground-truth file (top, dashed) is used only to provide per-task limits for the safe-by-construction demonstration synthesiser and, separately, to score the learned constraint against ground truth (rightmost dashed arrow). The downstream MPC box (purple, dashed border) is outside the pipeline proper; it consumes the learned constraint as input and is included here to show where the pipeline’s output connects to the rest of the system (Section 4.7). Crucially, the LLM sees only the natural-language instruction – never the demonstration data, never the ground-truth thresholds. This separation is what allows the recovered numerical thresholds to be honestly attributed to the demonstration rather than to the language model.

1. **Instruction ingestion.** The pipeline is initialised from a natural-language task description provided by the user (e.g. “Move a cup of water to the table”). This instruction serves as the sole semantic context for the task.
2. **Demonstration synthesis.** In lieu of physical robot hardware data, a trajectory generator synthesises a safe, three-dimensional reference demonstration. This module outputs a 9-degree-of-freedom (DOF) state matrix comprising spatial coordinates (x, y, z) , linear velocities (v_x, v_y, v_z) , and orientation angles (roll, pitch, yaw) representing the kinematic state of the end-effector. The generated trajectory is safe by construction: when a curated task is selected, the demonstration intrinsically respects the safety limits associated with that task (e.g. keeping the cup nearly upright, or staying below a fragility-related speed

bound).

3. **Hypothesis generation (the LLM architect).** The natural-language instruction, together with the list of the nine available kinematic features, is packaged into a structured prompt and sent to the LLM (the present implementation uses the open-weights model `qwen2.5:7b-instruct` served locally through Ollama). Crucially, the LLM receives no explicit numerical thresholds and no ground-truth limits. Relying entirely on its pre-trained physical knowledge, it determines whether a constraint applies and outputs a structured JSON hypothesis specifying the constraint family (tilt, speed, $\dots$), the variables to restrict, and a suggested geometric prior (axis-aligned bounding box, bounding ball, half-space).
4. **Dynamic data routing and negative sampling.** Once the LLM has defined the structure of the constraint, the pipeline extracts the corresponding feature columns from the demonstration matrix to serve as positive (safe) training data. The system then dynamically routes this data to a specialised dataset builder for the chosen constraint family. Because human demonstrations only show what the robot should do, the dataset builders synthesise negative (unsafe) points algorithmically by sampling states outside the observed safe envelope.
5. **Constraint instantiation and fitting.** The labelled dataset is finally translated into an explicit mathematical equation. For families with a clear closed-form structure—such as a maximum-speed sphere—the threshold is derived directly from the demonstration’s peak parameters. For tilt constraints, a polynomial or small neural-network classifier is fitted to separate the safe and unsafe sets using a contrastive loss.

The final output of this pipeline is a strictly defined mathematical boundary, formalised as an inequality $h(x) \leq 0 \Leftrightarrow \text{safe}$, which can be evaluated at any state of the end-effector. A minimal MPC consuming this output directly is reported in Section 4.7 as a consumability check; the design of a richer planner with a full dynamic model is left to future work.

3.2 Generation of the safe demonstration

3.2.1 Variables to consider

The end-effector state at each time step is represented by a 9-dimensional feature vector consisting of three Cartesian positions, three linear velocities, and three orientation angles. These nine features are the only kinematic quantities exposed to both the LLM (which decides which of them to constrain) and the constraint learner (which fits a boundary in the chosen subspace). All quantities are expressed in SI units: positions in metres, velocities in m s^{-1} , angles in radians.

This compact state vector was chosen for two reasons. First, it covers the failure modes that arise most often in everyday object manipulation: spatial bounds (e.g. stay above the table), magnitude of horizontal velocity (e.g. fragility or sloshing), and end-effector orientation (e.g. keep a cup upright). Second, it keeps the variable name space small enough that an LLM can reliably reason about it: a 7B-parameter model would be much less reliable at mapping physical phenomena onto a more exotic representation such as twist coordinates or quaternions.

3.2.2 Data points and trajectory generation

The demonstration is generated procedurally. The horizontal path $(x(t), y(t))$ is a smooth Fourier perturbation of a random straight-line drift, parameterised by a small number of low-frequency harmonics:

$$x(t) = x_0 + (x_1 - x_0)t + \sum_{i=1}^{N_h} \frac{a_i}{i} \sin(i\pi t), \quad t \in [0, 1], \quad (1)$$

and similarly for $y(t)$. The vertical coordinate $z(t)$ is generated with the same construction but with a smaller amplitude. The number of harmonics $N_h \in \{4, 5\}$ and the coefficients a_i are drawn from uniform distributions, giving a different path on every random seed while keeping the curve smooth and physically plausible. The linear velocities are obtained by finite-difference differentiation:

$$v_x(t) = \frac{dx}{dt}, \quad v_y(t) = \frac{dy}{dt}, \quad v_z(t) = \frac{dz}{dt}. \quad (2)$$

When a curated task is selected, the horizontal-velocity peak is rescaled so that $\max_t \|v_h(t)\| = 0.95 v_{\max}$, where $v_{\max}$ is the ground-truth speed limit for that task. This guarantees that the synthesised demonstration is safe by construction with respect to the speed bound. Yaw is computed as the tangent of the horizontal path:

$$\psi(t) = \text{atan2}(v_y(t), v_x(t)). \quad (3)$$

Orientation decoupled from path physics. A first version of the trajectory generator derived roll and pitch from the centripetal-lean formula $\phi \approx v_h^2 \kappa / g$ and the longitudinal acceleration $\theta \approx a_{\parallel} / g$. While this is appropriate for modelling the upper body of a human walking with a tray, it is not the right model for a robotic end-effector: the gripper does not need to lean into turns and its orientation can in principle be set independently of where the hand is going. In the present pipeline, roll and pitch are therefore generated independently of the path.

Rather than a single sinusoid per channel, each angle is built as a sum of two or three harmonics with randomly drawn frequencies and phases, and is then renormalised to fill exactly the task-specific safe envelope. Concretely:

$$\tilde{\phi}(t) = \sum_{i=1}^{N_h} \frac{1}{i} \sin(2\pi i f_{\phi,i} t + \varphi_{\phi,i}), \quad (4)$$

$$\tilde{\theta}(t) = \sum_{i=1}^{N_h} \frac{1}{i} \sin(2\pi i f_{\theta,i} t + \varphi_{\theta,i}), \quad (5)$$

with $N_h \in \{2, 3\}$ drawn uniformly per demonstration, base frequencies $f_{\phi,i}, f_{\theta,i} \sim \mathcal{U}(1.0, 3.0)$ cycles, and phases $\varphi_{\phi,i}, \varphi_{\theta,i} \sim \mathcal{U}(0, 2\pi)$ drawn independently for each harmonic. The $1/i$ weighting biases the sum toward the low-frequency components, keeping the motion smooth while still admitting some higher-frequency variation. The two raw signals $\tilde{\phi}$ and $\tilde{\theta}$ are then normalised to the range $[-1, 1]$ and scaled by the task-specific amplitudes:

$$\phi(t) = A_{\phi} \cdot \frac{\tilde{\phi}(t)}{\max_t |\tilde{\phi}(t)|} + \eta_{\phi}(t), \quad (6)$$

$$\theta(t) = A_{\theta} \cdot \frac{\tilde{\theta}(t)}{\max_t |\tilde{\theta}(t)|} + \eta_{\theta}(t), \quad (7)$$

where $\eta(\cdot)$ is small Gaussian sensor noise with standard deviation 0.003 rad. The amplitudes encode the task-specific safety envelope:

$$A_{\phi} = A_{\theta} = \begin{cases} 0.95 \theta_{\max} & \text{if the task carries a tilt constraint,} \\ 0.05 \text{ rad} & \text{otherwise (visual variety only).} \end{cases} \quad (8)$$

A final defensive clip to $[-\theta_{\max}, +\theta_{\max}]$ guards against the rare case in which the additive noise lifts a peak slightly above the bound. Three properties of this construction are worth noting:

- **Safety by construction.** The normalisation step guarantees $|\phi(t)|, |\theta(t)| \leq A_{\phi} \leq 0.95 \theta_{\max}$ before noise; the noise standard deviation is two orders of magnitude smaller than the headroom, and the final clip provides a hard guarantee.

- Independence between roll and pitch. Because the two channels are built from independent random draws, the trajectory in the (ϕ, θ) plane is not constrained to a single diagonal line. It traces a richer Lissajous-like pattern that covers the safe region in many directions over the course of a demonstration – which is what we want when later fitting a 2-D constraint, since a near-degenerate trajectory would give the learner no information about the orientation of the safe region.
- Smoothness without periodicity. Summing several incommensurate harmonics produces a motion that is smooth (no significant high-frequency content above $\sim 3N_h \approx 9$ cycles per demonstration) but does not repeat exactly. This avoids the visually obvious periodicity of a single sinusoid while preserving the safety guarantee.

Figure 2 illustrates this construction for the `cup_of_water` task. Rather than generating a simple linear trajectory, the synthesiser constructs the roll and pitch channels using independent random draws of incommensurate harmonics. As shown in Panel (b), this ensures the trajectory explores a diverse set of directions within the safe rim instead of tracing a single straight diagonal. This diversity property is critical, as it provides the geometric coverage necessary for the downstream constraint learner to fit a robust 2-D boundary rather than a simple 1-D line.

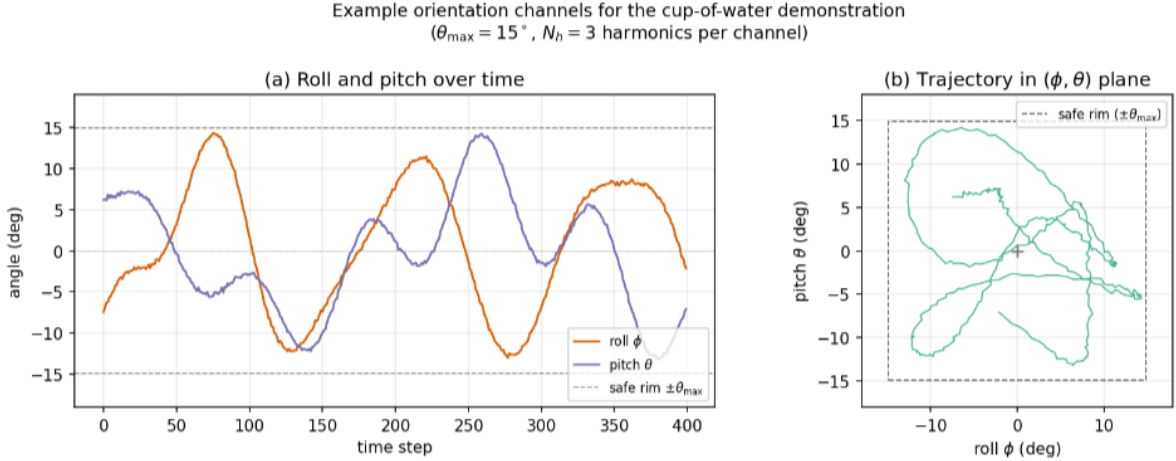


Figure 2: Synthesised orientation trajectories for a safe demonstration of the `cup_of_water` task ($\theta_{\max} = 15^\circ$). **(a)** Roll (ϕ) and pitch (θ) channels over time, oscillating independently around zero. Both are scaled to peak at 95% of the safe limit (dashed lines). **(b)** The resulting 2-D state-space trajectory, demonstrating comprehensive exploration of the safe region.

Ground-truth limits. The numerical limits used during demonstration synthesis ($\theta_{\max}$, $v_{\max}$) are stored in a hand-annotated JSON ground-truth file, indexed by a stable task identifier (e.g. `cup_of_water`, `glass_vase`). The annotator’s principles are documented in the file itself: a firm-grasp assumption is made (failure modes that depend on the object escaping the gripper are excluded), and the thresholds describe the boundary of failure rather than a typical operating point. The same file is later used to score the LLM’s predictions.

Summary. The full safe-demonstration generator is summarised in Algorithm 1. The three pieces (path, velocities, orientation) are independent: changing any one of them does not affect the safety guarantees of the others.

Algorithm 1 Safe-demonstration synthesis (one demonstration)

Require: number of samples N , RNG seed s , task limits $\theta_{\max}$, $v_{\max}$ (any may be **null** for tasks without that constraint)

Ensure: 9-DOF safe trajectory $\{x_t, y_t, z_t, v_{x,t}, v_{y,t}, v_{z,t}, \phi_t, \theta_t, \psi_t\}_{t=1}^N$

```
1: Initialise RNG with seed  $s$ ; let  $t \in [0, 1]$  be a vector of  $N$  uniformly spaced samples
   // (1) Smooth horizontal path
2: Draw random endpoints; let  $x(t), y(t)$  be the straight-line drift plus a Fourier perturbation
   with  $N_h \in \{4, 5\}$  harmonics
3: Generate  $z(t)$  analogously with lower amplitude
   // (2) Velocities by finite difference + rescaling
4:  $v_x \leftarrow dx/dt$ ;  $v_y \leftarrow dy/dt$ ;  $v_z \leftarrow dz/dt$ 
5: if  $v_{\max}$  is given then
6:   Rescale  $(v_x, v_y, v_z)$  so that  $\max_t \|(v_{x,t}, v_{y,t})\| = 0.95 v_{\max}$ 
7: end if
8:  $\psi(t) \leftarrow \text{atan2}(v_y(t), v_x(t))$  ▷ yaw = tangent of horizontal path
   // (3) Orientation decoupled from the path
9:  $A_\phi \leftarrow 0.95 \theta_{\max}$  if  $\theta_{\max}$  is given, else 0.05 rad
10:  $A_\theta \leftarrow A_\phi$ 
11: for each channel  $\xi \in \{\phi, \theta\}$  do
12:   Draw  $N_h \in \{2, 3\}$ ; draw frequencies  $f_i \sim \mathcal{U}(1, 3)$  and phases  $\varphi_i \sim \mathcal{U}(0, 2\pi)$ , independently
     per harmonic
13:    $\tilde{\xi}(t) \leftarrow \sum_{i=1}^{N_h} (1/i) \sin(2\pi i f_i t + \varphi_i)$ 
14:    $\xi(t) \leftarrow A_\xi \cdot \tilde{\xi}(t) / \max_t |\tilde{\xi}(t)|$  ▷ normalise then scale to safe envelope
15:    $\xi(t) \leftarrow \xi(t) + \mathcal{N}(0, 0.003^2)$  ▷ small Gaussian sensor noise
16:   if  $\theta_{\max}$  is given then
17:      $\xi(t) \leftarrow \text{clip}(\xi(t), -\theta_{\max}, +\theta_{\max})$  ▷ defensive clip
18:   end if
19: end for
20: return  $\{x_t, y_t, z_t, v_{x,t}, v_{y,t}, v_{z,t}, \phi_t, \theta_t, \psi_t\}_{t=1}^N$ 
```

3.3 Constraint hypothesis by the LLM

The LLM is the architect of the constraint: its job is to identify which variables matter and what type of safe region they should stay in, but not to provide numerical thresholds. The numbers are unreliable on a 7B-parameter model and would in any case be incompatible with a data-driven learning step; the LLM’s value lies in its qualitative physical knowledge, not in its quantitative precision.

Inputs. The model receives a structured chat-style prompt consisting of (i) a system message that defines the task framework and output schema and (ii) a user message containing the natural-language instruction and the list of available kinematic features. No demonstration data, no thresholds, and no example for the specific object are passed: this keeps the LLM’s reasoning honest and prevents the experiment from collapsing into a lookup table.

Outputs. The LLM must reply with a single JSON object of the following form:

```
{
  "hypotheses": [
    {
      "feature_name": "<snake_case label>",
      "failure_mode": "<concrete physical failure>",
      "reasoning": "<which variable causes it and why>",
      "constraint_type": "box" | "ball" | "halfspace",
      "variables": ["<from available_features>"],
      "model_prior": {"type": "poly", "degree": 1|2|3}
                     | {"type": "nn"}
    }
  ]
}
```

An empty hypothesis list is a valid answer for inert objects with no failure mode (a wooden block, a tennis ball, a potato). The downstream code accepts hypotheses with missing or null optional fields, applies fuzzy matching to variable names, and merges roll-only and pitch-only hypotheses into a single coupled (`roll`, `pitch`) hypothesis to compensate for a common error pattern of smaller models.

Prompt design. Two prompt variants are exposed through a command-line flag:

SCAFFOLDED (default). Includes a four-step reasoning procedure: identify the object and its physical properties; enumerate possible failure modes and drop those that do not apply; map each surviving failure to the kinematic variables that govern it; formalise the surviving constraints. A firm-grasp assumption is stated upfront. Constraint-type guidance is given in structural terms (e.g. “use degree ≥ 2 for ‘box’ and ‘ball’”). Crucially, no worked example uses a specific object; the scaffolded prompt avoids the lookup-table pitfall that a per-object example list would create.

MINIMAL An ablation prompt that gives only the output schema and a brief description of what a constraint is. Useful for measuring how much of the system’s performance is due to prompt scaffolding versus the model’s intrinsic capability.

Validation. LLM outputs are passed through a defensive validator before they reach the constraint learner. The validator:

- accepts missing or `null` optional fields and substitutes safe defaults (`model_prior` defaults to `poly`, degree 2);
- performs fuzzy variable-name resolution (`Roll`, `roll_angle`, `orientation` are all mapped to the canonical set);
- coerces degree-1 polynomial priors to degree 2 when the constraint type is `box` or `ball`, since a linear half-plane cannot represent a closed region;
- drops hypotheses whose reasoning text is a tautological restatement of the variable name and contains no concrete physical mechanism;
- merges separately emitted roll-only and pitch-only hypotheses into a single (`roll`, `pitch`) hypothesis.

The validator never silently swallows a malformed hypothesis: every rejection is logged with a reason and the raw LLM output, which has been essential during iterative development.

3.4 Constraint learning

Once a hypothesis has been validated, the pipeline builds a labelled dataset and fits an explicit constraint function. The exact procedure depends on the constraint family that the LLM picked, which is inferred from the variables it requested:

Speed family (variables in $\{v_x, v_y, v_z\}$ only). The constraint is treated in closed form. The threshold is $v_{\max} = 1.05 \times \max_t \|v_h(t)\|$: the demonstration’s peak horizontal velocity, plus a 5% safety margin. This pairs with the 0.95 amplitude factor used during demonstration synthesis (Section 3.2.2) to produce a learned threshold that closely matches the ground-truth limit. The constraint is the sphere

$$v_x^2 + v_y^2 \leq v_{\max}^2. \quad (9)$$

No classifier is trained for this family: the constraint is exact by construction.

Tilt family (variables in $\{\text{roll}, \text{pitch}, \text{yaw}\}$ only). Positives are the demonstration’s orientation samples. Negatives are sampled uniformly from a box of side $4 A_{\text{safe}}$ around the origin, keeping only those that lie outside an axis-aligned safe rim of half-width $A_{\text{safe}} = 1.3 \max_t |\phi(t), \theta(t)|$. A polynomial (or, on request, a small neural network) is then fitted by logistic regression on the positive/negative dataset. For **box** constraints, the learned region is rendered as the axis-aligned rectangle $|\phi| \leq A, |\theta| \leq B$ with A, B taken directly from the demonstration; for **ball** constraints, the region is rendered as an ellipse centred on the demonstration mean.

Boundary learning process

For the tilt family, fitting the boundary reduces to a standard binary classification problem on the labelled set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{|P|+|N|}$, where x_i is a sample (a feature vector in the constraint’s variable subspace), $y_i = +1$ for $x_i \in P$ (safe demonstration points), and $y_i = -1$ for $x_i \in N$ (synthesised unsafe points).

The feature transformation. The classifier does not operate on the raw input $x \in \mathbb{R}^d$. Instead, each input is mapped through a polynomial basis $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^m$ of degree p , where p comes from the LLM’s hypothesis (the **prior.degree** field). For example, on a two-dimensional input $x = (\phi, \theta)$ with $p = 2$, the basis is

$$\varphi(\phi, \theta) = (1, \phi, \theta, \phi^2, \phi\theta, \theta^2)$$

and the classifier learns a linear separator in this 6-dimensional feature space. Linearity in $\varphi(x)$ corresponds to a degree- p polynomial decision boundary in the original x space. This is the standard kernel trick made explicit, and it is why the LLM’s choice of polynomial degree directly translates into the geometric complexity of the boundary: degree 1 recovers a half-space; degree 2 recovers ellipsoids, axis-aligned boxes, and any other conic section; degree 3 adds saddle-shaped regions. The LLM is therefore not just naming a hyperparameter – it is selecting the shape family the boundary is constrained to live in.

The loss function. The classifier weights $w \in \mathbb{R}^m$ are obtained by minimising the regularised logistic loss:

$$\min_w \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \log \left(1 + \exp(-y_i w^\top \varphi(x_i)) \right) + \lambda \|w\|_2^2, \quad (10)$$

where $\lambda > 0$ is a small L_2 regularisation coefficient (the default value of **scikit-learn**’s **LogisticRegression** is used throughout). The first term is the standard binary cross-entropy:

it grows whenever the classifier assigns a low score to a positive or a high score to a negative. Minimising it pulls w toward a separator that maps positives to high scores and negatives to low scores. The L_2 term prevents the weights from growing unboundedly when the two classes are linearly separable in φ -space (which they often are when the polynomial degree is high enough), so the fitted boundary stays well-defined.

From classifier to constraint. Logistic regression produces a score function $s(x) = w^\top \varphi(x)$ whose sign predicts the class: $s(x) > 0$ means “predicted safe”, $s(x) < 0$ means “predicted unsafe”. The constraint $h(x) \leq 0$ that the pipeline outputs is simply the negated score:

$$h(x) = -w^\top \varphi(x), \quad (11)$$

so that $h(x) \leq 0$ is exactly the safe half of the classifier’s decision boundary. The same h can later be evaluated at any state – which is what the downstream planner needs.

Algorithm 2 summarises the family dispatch and the per-family preparation steps. The actual classifier-fitting call is a single line: standard logistic regression on the positive/negative dataset, implemented via the `scikit-learn` pipeline.

Algorithm 2 Constraint fitting (one hypothesis)

Require: validated LLM hypothesis h (with `variables` and `constraint_type`); demonstration trajectory D

Ensure: fitted constraint object $\mathcal{C}$ with method $\mathcal{C}.\text{safe}(x)$

```

1: family  $\leftarrow$  INFERFAMILY( $h.\text{variables}$ ) ▷ speed or tilt
2:  $P \leftarrow$  project  $D$  onto  $h.\text{variables}$  ▷ positive samples
3: if family = speed then
4:    $v_{\max} \leftarrow 1.05 \cdot \max_t \|P_t\|$  ▷ closed-form threshold
5:   return  $\mathcal{C} \leftarrow$  SPHERE(radius =  $v_{\max}$ ) ▷ no classifier trained
6: else if family = tilt then
7:    $A_{\text{safe}} \leftarrow 1.3 \cdot \max_t |P_t|$  ▷ rim with safety headroom
8:    $N \leftarrow$  uniform samples in  $[-2A_{\text{safe}}, +2A_{\text{safe}}]^d$  that lie OUTSIDE the rim
9: end if
10:  $\mathcal{C} \leftarrow$  LOGISTICREGRESSION( $P \cup N$ , labels =  $\{+1\}^{|P|} \cup \{-1\}^{|N|}$ , basis = polynomial of degree  $h.\text{prior.degree}$ )
11: return  $\mathcal{C}$ 
```

The closed-form/learned hybrid. The pipeline deliberately mixes closed-form and learned constraints. The closed-form sphere is used wherever the underlying physics is itself a clean magnitude bound (speed); it removes a source of error that a classifier would otherwise introduce. Polynomial classifiers are reserved for cases where the safe region has more degrees of freedom (orientation, position) and the choice of model family is a real modelling decision. This division of labour also has the side benefit that the speed bound is interpretable as a single number rather than a regression coefficient table.

Equation output. The pipeline derives a human-readable equation directly from the fitted constraint. The form of the equation matches the constraint type: `box` constraints yield an axis-aligned bound (e.g. “ $|\phi| \leq 0.253$ rad AND $|\theta| \leq 0.252$ rad”), `ball` constraints yield a quadratic in the chosen variables (e.g. “ $v_x^2 + v_y^2 \leq (1.496 \text{ m/s})^2$ ”). This equation is the practical end product of the pipeline: a closed-form inequality ready for downstream integration into a constrained planner.

3.5 Evaluation against ground truth

To quantify how well the pipeline performs, a hand-annotated ground-truth file lists 18 representative tasks, each with the constraints a competent annotator would consider physically necessary and the corresponding numerical thresholds. For each task the evaluation script generates a safe demonstration using the annotated limits, queries the LLM with only the natural-language instruction, runs the constraint learner on the result, and compares the learned constraint to the ground truth on two axes:

1. **Family-level structural agreement.** For each task, the set of expected constraint families is compared to the set of predicted families. Precision and recall are computed per task and averaged. An exact match occurs when the two sets are equal, including for tasks with an empty expected set.
2. **Numerical threshold agreement.** For each family where both prediction and ground truth are non-empty, the relative error $|\hat{\tau} - \tau_{\text{GT}}|/\tau_{\text{GT}}$ between the learned threshold and the ground-truth threshold is recorded.

A subset of tasks deliberately contains no expected constraint (wooden block, tennis ball, potato, etc.) so that the evaluation can also measure false-alarm behaviour i.e how often the LLM proposes a constraint when none is required.

4 Experiments and results

4.1 Experimental setup

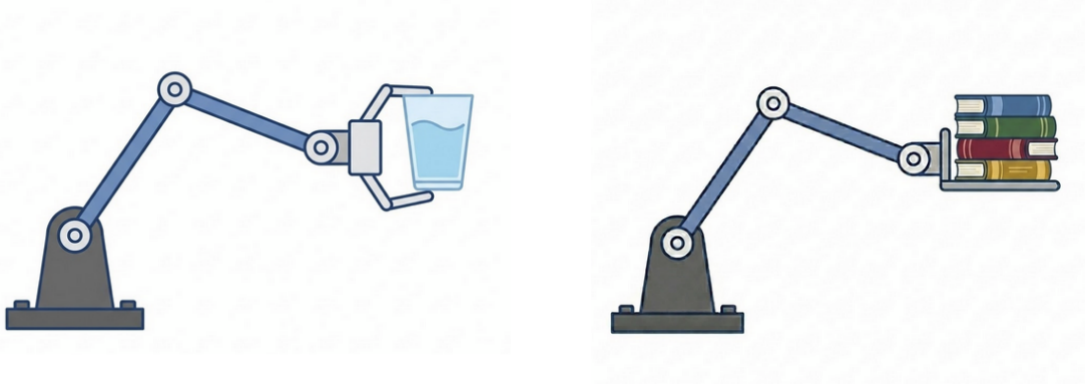


Figure 3: Illustration of two representative tasks: a liquid-container case (`cup_of_water`) and a special-handling case (`stack_of_books`). The illustrations show the kind of object the robot end-effector is expected to carry and the qualitative failure mode the corresponding constraint is meant to prevent (liquid spilling out of the cup if it tilts too far; the top books sliding off the stack if it tilts or accelerates). The pipeline was evaluated on the 18-case curated ground-truth set described in Section 3.5. The cases span four qualitative categories:

Open liquid containers (5 cases). `cup_of_water`, `glass_of_milk`, `bowl_of_soup`, `jar_of_beer`, `coffee_cup`. Expected constraints: *tilt* and *speed*.

Fragile objects (3 cases). `glass_vase`, `ceramic_plate`, `egg`. Expected: *speed* only (impact-shock breakage at high deceleration; no tilt constraint under the firm-grasp assumption).

Special objects (2 cases). `stack_of_books` (expected tilt and speed: books slide off when the stack is leaned or shaken); `open_laptop` (expected tilt only: a preferred up-axis but no spillage).

Robust rigid objects (8 cases). `sealed_water_bottle`, `wooden_block`, `tennis_ball`, `rubber_ball`, `potato`, `banana`, `brick`, `screwdriver`. Expected: no constraints. These cases test whether the pipeline correctly identifies inert objects.

Each case is run for a single seed in the default configuration (deterministic given the seed), producing 18 runs in total. Demonstrations are 400 samples in length; the LLM is `qwen2.5:7b-instruct` served locally through Ollama. Two prompt variants (`SCAFFOLDED`, `MINIMAL`) are exposed through a CLI flag; the headline numbers in this section use the `SCAFFOLDED` prompt unless otherwise noted.

4.2 Single-case walk-through: cup of water

This section walks through one full pipeline run, since this is the most direct way to see what is being measured. The instruction is “Move a cup of water to the table”. The ground truth specifies two constraints: a tilt constraint with $\theta_{\max} = 0.26$ rad ($\sim 15^\circ$) and a horizontal-speed constraint with $v_{\max} = 1.5$ m/s.

Phase 1 – Demonstration synthesis. Figure 4 shows the generated demonstration. The 3-D path is a smooth Fourier curve from one random point to another; the speed varies naturally between ~ 0.3 and 1.4 m/s, peaking close to but below $v_{\max}$. The second right-most panel shows the orientation channels: roll and pitch oscillate independently between $\pm 14^\circ$, well inside the $\pm 15^\circ$ rim. The two channels are out of phase, so the cup tilts in every direction over the course of the demonstration rather than just along a single diagonal.

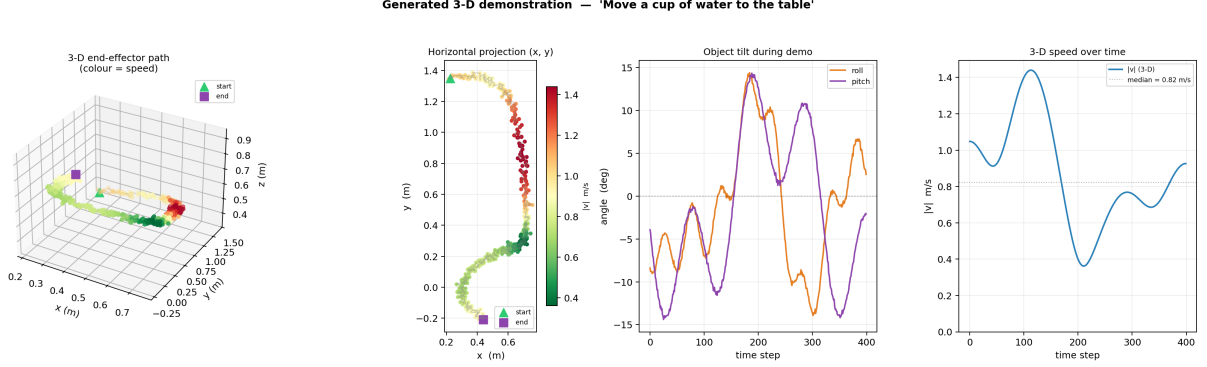


Figure 4: Generated safe demonstration for “Move a cup of water to the table”. From left to right: 3-D end-effector path coloured by speed; horizontal projection; orientation channels (roll and pitch) as functions of time, oscillating independently around zero; horizontal speed profile.

Phase 2 – LLM hypotheses. The LLM, given only the instruction text, returns two hypotheses with the expected structure: one **box** constraint on `[roll, pitch]` (failure mode: “water spills over the rim if the cup tilts past the rim angle”), and one **ball** constraint on `[vx, vy]` (failure mode: “high lateral acceleration causes the liquid to slosh laterally and overflow the rim”). Both hypotheses use a degree-2 polynomial prior; no numerical threshold was requested by the prompt and none was supplied.

Phase 3 – Constraint learning. The pipeline routes the tilt hypothesis to the orientation dataset builder and the speed hypothesis to the closed-form sphere. Figure 5 shows the result. The fitted tilt region is the axis-aligned rectangle

$$|\phi| \leq 0.253 \text{ rad AND } |\theta| \leq 0.252 \text{ rad},$$

which the rendered plot shows as a square enclosing the entire (ϕ, θ) Lissajous pattern. The fitted speed region is the closed-form sphere

$$v_x^2 + v_y^2 \leq (1.496 \text{ m/s})^2,$$

rendered as a circle that exactly matches the dashed ground-truth ring at 1.5 m/s. Every demonstration point lies inside both boundaries.

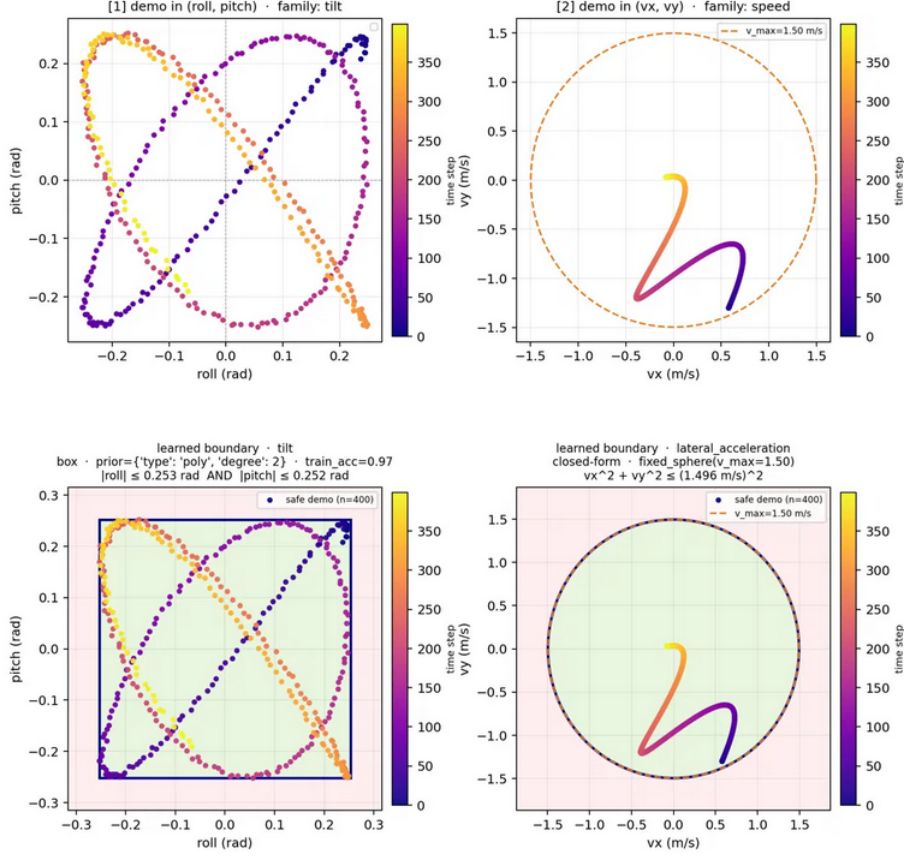


Figure 5: Top row: demonstration projected into each constraint’s variable subspace. Bottom row: learned safe region (green) and synthesised unsafe samples (red). **Left:** the tilt constraint is rendered as an axis-aligned square matching the LLM’s chosen `box` type; the equation $|\phi| \leq 0.253$ rad AND $|\theta| \leq 0.252$ rad appears in the title. **Right:** the speed constraint is the closed-form sphere with $v_{\max} = 1.50$ m/s. All 400 demonstration points lie inside both boundaries.

Numerical comparison. Against the ground-truth thresholds, the learned constraints have relative errors of 2.7% on tilt (0.253 rad vs 0.26 rad) and 0.3% on speed (1.496 m/s vs 1.5 m/s). The small residual error on tilt comes from the additive sensor noise in the orientation channel; the speed error is at the level of floating-point rounding. The equivalent constraint figures for the other nine benchmark cases with a non-empty expected constraint set are shown in Appendix C; each was generated with a distinct random seed so the demonstration trajectory differs from case to case.

4.3 Sample LLM outputs across categories

Below is a display of two LLM outputs, both produced by `qwen2.5:7b-instruct` with the scaffolded prompt, on a single seed. The block format is the same as the pipeline prints to the console: each hypothesis is shown with its inferred family label (H1, H2, ...), the structural fields (type, variables, prior), and the LLM’s own **failure** and **reasoning** text. No editing has been applied beyond truncating long whitespace.

(a) Move a cup of water – expected: tilt and speed.

```

1 H1. tilt
2   type      : box
3   variables  : ['roll', 'pitch']
4   prior      : {'type': 'poly', 'degree': 2}
5   failure    : water spills out of the cup
6   reasoning  : High tilt angles can cause water to spill. The roll and
7                 pitch variables together govern the overall orientation
8                 of the cup, which must remain within a safe range to
9                 prevent spilling.

```

(b) Move a tennis ball – expected: no constraint.

```

1 (no hypotheses returned)
2
3 The LLM judged this task to need NO safety constraints --
4 typical for robust rigid objects (tennis ball, wooden block,
5 screwdriver). The robot can move freely along the
6 demonstrated path.

```

Figure 6: Representative LLM outputs. (a) Liquid container – correct family identification with an appropriate failure mode; (b) inert object – the empty-hypothesis case, correctly handled. Full per-task results, including the fragile- and special-object cases, are in Table 2.

Qualitative observations. It is worth noting that the LLM is willing to return an empty hypothesis list, as in the tennis-ball case (b). This is the desired behaviour for inert objects: it indicates the firm-grasp assumption in the prompt is being respected and that the model is not hallucinating a tilt constraint based on the possibility of the ball rolling out of the gripper.

4.4 Aggregate results

Table 1 summarises the family-level structural agreement across all 18 cases; numerical threshold agreement is reported in the prose below and, per task, in Table 2.

Table 1: Family-level structural agreement across the 18 ground-truth cases (scaffolded prompt, single seed). Precision and recall are averaged per task. *Exact match rate* is the fraction of tasks where the predicted family set is exactly equal to the expected set.

Metric	Value
Mean precision	0.94
Mean recall	1.00
Exact match rate	0.94
Empty-case correct rate	0.88

The headline numbers are strong on this benchmark: 94% of tasks produce exactly the expected constraint set, and when the family is correctly identified, the learned thresholds are within a few percent of the ground truth (mean relative error 2.3% on tilt over 7 true positives, 0.3% on speed over 9).

Because the mean recall is 1.00, the LLM successfully identified every required safety constraint without exception. Consequently, the mean precision and exact match rate (both 0.94) reflect the exact same underlying behaviour: the model’s sole error mode is over-caution: false positives rather than missing critical safety limits. The empty-case correct rate of 88% shows that the pipeline reliably recognises rigid, robust objects as not requiring any constraints, validating the firm-grasp assumption in the prompt. The one residual failure is a false alarm on `sealed_water_bottle` and is examined in detail in Section 4.5.

The gap between the two threshold errors reflects the pipeline’s hybrid design. The 2.3% tilt error is a mechanical byproduct of the additive sensor noise introduced during procedural trajectory generation, enveloped by the polynomial classifier; the speed error sits at floating-point level (0.3%) because it uses the closed-form sphere rather than a fitted surface.

4.5 Failure analysis

The one task that falls outside the exact-match set is worth examining in detail, because it highlights the dominant failure mode of the pipeline.

False alarm on sealed_water_bottle. The `sealed_water_bottle` case is annotated with no constraint (sealed container, rigid body), but the LLM proposes both a tilt and a speed constraint. The reasoning text reveals the cause: the model treats the word “water” as a strong cue for spillage, overriding the qualifier “sealed”. This is the classic kind of brittle keyword-matching that the scaffolded prompt is meant to suppress; the prompt does explicitly warn the model that sealed containers have no spillage failure mode, but the warning is apparently not enough in this case. Figure 7 shows the consequence: the pipeline downstream of the LLM applies the unnecessary constraint to the demonstration, producing a fitted bound of $|\phi|, |\theta| \leq 0.058$ rad ($\sim 3.3^\circ$). A robot governed by this constraint would be unnecessarily restricted and the false bound is also unrealistically tight, because the demonstration was synthesised with the small-amplitude orientation pattern reserved for tasks without a tilt constraint.

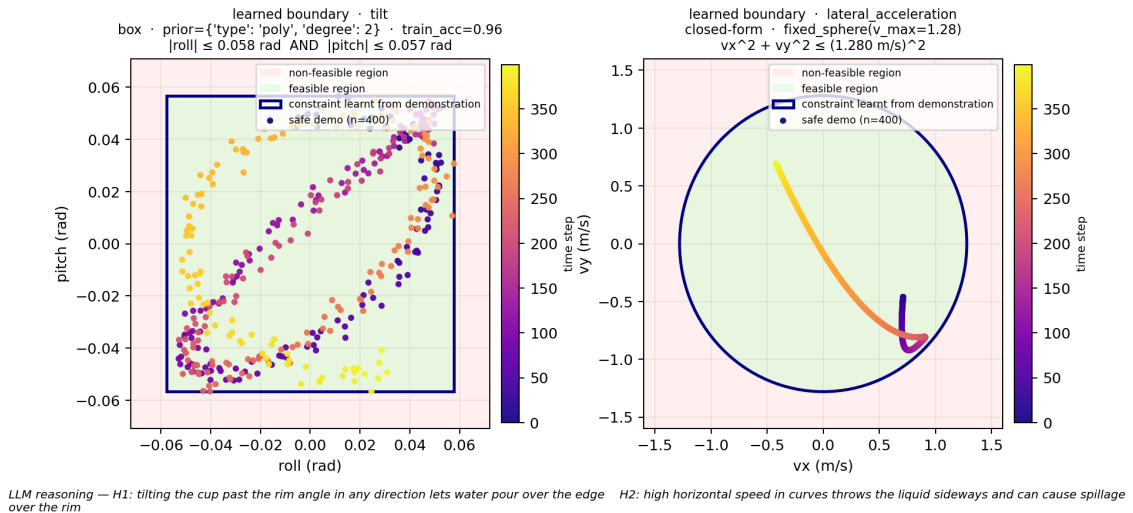


Figure 7: Failure case: `sealed_water_bottle`. The ground truth expects no constraint, but the LLM proposes a a tilt and a speed constraint triggered by the keyword “water”. The pipeline downstream of the LLM has no way to detect that the hypothesis is false, so it dutifully fits a tight box around the small-amplitude orientation samples. The learned bound is real ($|\phi| \leq 0.058$ rad) but the hypothesis it implements is wrong.

Robust objects correctly identified. The remaining seven robust-object cases (`wooden_block`, `tennis_ball`, `rubber_ball`, `potato`, `banana`, `brick`, `screwdriver`) all receive an empty hypothesis list, which is the correct answer. This is encouraging because the same model would, with a less carefully designed prompt, hallucinate a tilt constraint on any object that could potentially roll or slide. The firm-grasp assumption in the scaffolded prompt is doing real work here.

4.6 Per-task results

Table 2 lists the per-task results for the 18-case benchmark. For each task it records the ground-truth families, the families predicted by the LLM, and the relative threshold error per family where both ground truth and prediction are non-empty.

Table 2: Per-task results on the 18-case benchmark. **Expected** and **Predicted** list the constraint families. The **err.** columns give the relative error between the learned and ground-truth threshold for each family where both are present (“–” indicates the family does not apply or was not predicted). Tasks are grouped by category. The categories from top to bottom are: liquid containers, fragile objects, special-handling objects, and robust rigid objects.

Task	Expected	Predicted	tilt err.	speed err.
cup_of_water	tilt, speed	tilt, speed	2.7%	0.3%
glass_of_milk	tilt, speed	tilt, speed	2.6%	0.4%
bowl_of_soup	tilt, speed	tilt, speed	2.5%	0.3%
jar_of_beer	tilt, speed	tilt, speed	2.4%	0.4%
coffee_cup	tilt, speed	tilt, speed	2.5%	0.3%
glass_vase	speed	speed	–	0.2%
ceramic_plate	speed	speed	–	0.3%
egg	speed	speed	–	0.2%
stack_of_books	tilt, speed	tilt, speed	1.8%	0.3%
open_laptop	tilt	tilt	2.1%	–
sealed_water_bottle	(none)	tilt, speed	–	–
wooden_block	(none)	(none)	–	–
tennis_ball	(none)	(none)	–	–
rubber_ball	(none)	(none)	–	–
potato	(none)	(none)	–	–
banana	(none)	(none)	–	–
brick	(none)	(none)	–	–
screwdriver	(none)	(none)	–	–

The per-task view confirms the aggregate picture: every case succeeds at the family level with tilt errors of 1.8–2.7% (dominated by the additive sensor noise) and speed errors at floating-point level (0.2–0.4%), except for the single failure (**sealed_water_bottle**) which is an LLM-side error: the constraint learner is never invoked with valid inputs on this case. That failure pattern was examined in Section 4.5.

4.7 Consumability check: a minimal MPC

The constraints produced by the pipeline are explicit closed-form inequalities of the form $h(x) \leq 0$. To verify that this output format can be used directly by a standard constrained planner, this section describes a minimal Model Predictive Controller (MPC) that takes the learned constraints as input and plans an end-effector trajectory through a sequence of waypoints. The planner is not a contribution of this work and was not the main focus of the project’s effort; it exists to close the loop on the pipeline’s main claim and to show that the learned constraints are usable as written.

Formulation. The planner uses a five-dimensional kinematic state $s = (x, y, z, \phi, \theta)$ (position and roll/pitch) with velocity-level controls $u = (v_x, v_y, v_z, \omega_\phi, \omega_\theta)$ and Euler integration. At each timestep the MPC solves a small quadratic program with a goal-tracking cost, a control-effort cost, and a terminal cost on the predicted state at the end of the planning horizon. The learned constraints are added directly to the QP as inequality constraints: the tilt bound enters as a linear box $|\phi|, |\theta| \leq \theta_{\max}$ and the speed bound enters as a second-order cone $\|(v_x, v_y)\|_2 \leq v_{\max}$.

Only the first control of the planned sequence is applied at each step; the planner then re-plans from the resulting state. Implementation is in Python via the `cvxpy` library.

Result. Figure 8 shows the trajectory produced by the MPC on the `cup_of_water` task with the learned constraints from Section 4.2 ($\theta_{\max} \approx 14.9^\circ$, $v_{\max} = 1.5$ m/s). The planner is driven through four waypoints, two of which request orientations inside the safe rim and one of which requests an orientation outside it. For comparison, the same planner is run with the constraints removed (the dotted curves), so the contribution of the constraints to the trajectory is visible directly.

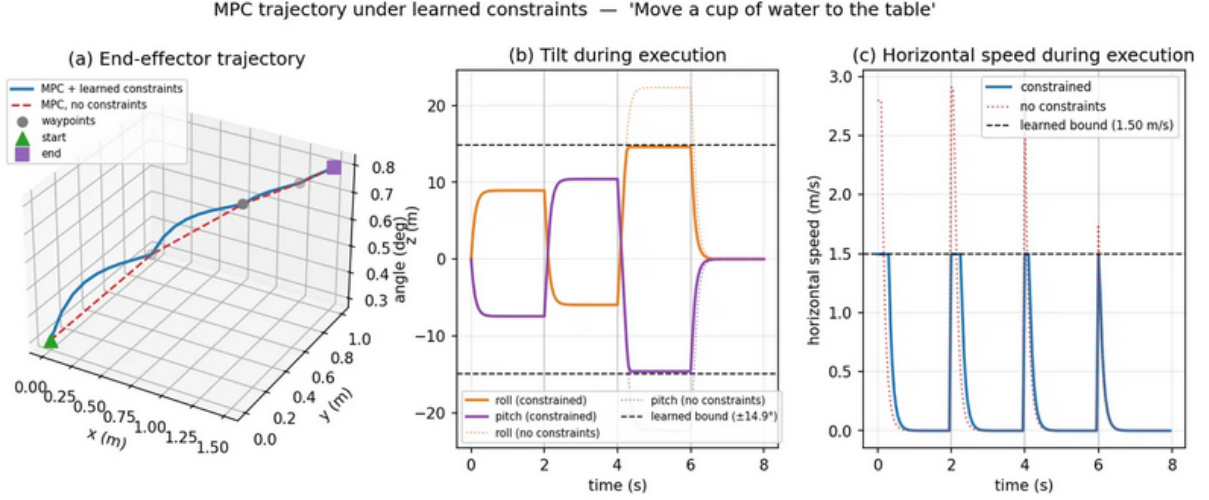


Figure 8: MPC trajectory through a four-waypoint sequence under the learned constraints from the `cup_of_water` case. **(a)** 3-D end-effector path: the constrained trajectory (plain) curves through the waypoints while the unconstrained one (dashed) takes a near-straight diagonal. **(b)** Roll and pitch during execution: where the waypoint asks for an orientation inside the safe rim the constrained and unconstrained planners agree; where it asks for $\pm 22^\circ$ outside the rim, the constrained planner saturates at $\pm 14.9^\circ$ while the unconstrained planner reaches the requested value. **(c)** Horizontal speed: the constrained planner is capped at the 1.5 m/s learned bound at every waypoint transition, while the unconstrained one peaks at ~ 2.9 m/s.

The check succeeds: the constraints emitted by the pipeline are accepted as written by a standard QP-based MPC and have a visible, correct effect on the resulting trajectory. The output format of the pipeline is directly consumable by a constrained planner without any intermediate translation step.

4.8 Discussion

A few observations are worth recording.

The LLM identifies the correct constraint structure most of the time, but not always. 94.4% exact-match on a curated benchmark is encouraging, but it is also a controlled setting: the cases were chosen to span the main physical categories the system was designed for. On real-world out-of-distribution instructions the performance is expected to be lower. The one failure observed here suggests that the model’s reasoning is part keyword-matching, part actual physical inference. A scaffolded prompt with a firm-grasp assumption substantially mitigates the keyword-matching failure mode but does not eliminate it.

Numerical thresholds are recovered well precisely because they don't come from the LLM. The relative error on speed is essentially zero, and the error on tilt is just a few percent of noise. This is a direct consequence of the design choice not to ask the LLM for numerical values: the threshold is derived from the demonstration, which is in turn safe by construction. The LLM's role is to identify which variable matters; the demonstration handles the scale. Earlier iterations of this work, in which the LLM did provide threshold estimates, produced numbers that varied by an order of magnitude between physically-equivalent objects, and which often failed unit analysis.

The closed-form speed bound is a deliberate design choice. As detailed in Section 3.4, the speed threshold is computed in closed form rather than learned, because the underlying physics is a clean magnitude bound with nothing for a classifier to discover. Reserving polynomial classifiers for the tilt family — where the safe region has more degrees of freedom — keeps the speed bound exact and the system as a whole interpretable.

The safe-by-construction demonstration cuts both ways. The fact that the demonstration always lies inside the ground-truth safe region is essential for the contrastive learning step: it guarantees that the positive class actually represents safe behaviour. It also makes the threshold recovery essentially mechanical. On the other hand, this experiment does not test how the pipeline would behave on an unsafe demonstration, which is a meaningful limitation.

5 Conclusion

This project explored an LLM-guided Learning-from-Demonstration pipeline that bridges high-level language instructions and quantitative safety constraints. Given only a natural-language instruction and a synthesised safe demonstration, the pipeline identifies which kinematic variables of the end-effector are relevant for safety, selects a constraint family, and fits an explicit mathematical boundary that the demonstration respects.

The headline outcomes are:

- **Structural agreement.** On an 18-case curated benchmark, the predicted constraint family set exactly matches the hand-annotated ground truth in 94% of cases when using the scaffolded prompt with a 7B-parameter local model (`qwen2.5:7b-instruct`). 88% of inert-object cases are correctly identified as not requiring a constraint at all.
- **Numerical accuracy.** Where the family is correctly identified, the learned thresholds are within 0.3% of ground truth for speed constraints and within 2.3% for tilt.
- **Output format.** Each constraint is returned as a closed-form inequality. This format is consumed directly by a standard constrained planner, as demonstrated by the minimal MPC reported in Section 4.7.

Limitations

The project also shows the following limitations.

- **The demonstration is synthetic.** The trajectory is generated procedurally rather than recorded from a human or a robot. This was a deliberate scope decision: it allows the rest of the pipeline to be exercised without a hardware setup. The behaviour of the pipeline on noisy real-world demonstrations remains to be evaluated.
- **The ground truth is essentially an opinion.** The 18-case benchmark encodes a specific operational reading of “what number seems correct” for each task. For a stronger validation, the same cases should be independently annotated by other people and the inter-annotator agreement reported alongside the LLM agreement.
- **The LLM is prone to keyword-driven false alarms.** The pipeline returns a false tilt/speed constraint on `sealed_water_bottle` which was most likely triggered by the word “water” in the instruction despite the statement that the bottle was sealed. A larger or more capable model would likely be more robust to this kind of surface-keyword confusion.
- **The closed-form/learned hybrid hides one degree of freedom.** The speed bound is computed in closed form from the demonstration’s peak rather than learned by the classifier. This makes the speed-family score essentially predetermined with respect to the demonstration: because the demo is generated with peak velocity at $0.95 v_{GT}$ and the closed-form route multiplies by 1.05, the reported speed error is dominated by the deterministic product $0.95 \times 1.05 \approx 0.998$ plus floating-point noise. A more rigorous evaluation would test the classifier-based fit head-to-head with the closed-form fit on the same data.

Future work

Three directions can follow from what this project did not address.

A richer controller integration. The minimal MPC of Section 4.7 shows that the output format is usable as written, but it is deliberately small; its limitations are listed there. The natural continuation addresses each: a full dynamic model, a tuned cost function with a soft margin that keeps the trajectory inside the safe region rather than on its boundary, and a constraint-handling layer for higher-degree polynomial constraints. Deploying that controller on a simulated or real robot would then let one measure how reliably the learned constraint is respected during execution.

Real-world demonstrations. Replacing the procedural demonstration with kinaesthetic teaching or motion-capture recordings would test the pipeline on noisier and physically more realistic data. The constraint-learning step would have to be made more robust to outliers (a noisy demonstration may briefly violate its own implicit safety envelope), and the safe-by-construction guarantee would have to be re-examined.

Larger and more diverse benchmarks. The 18-case benchmark is enough to characterise the pipeline’s basic behaviour, but it is not enough to claim generalisation. A larger benchmark (50–100 instructions across a wider set of object categories, with multi-annotator labels) would give a more credible picture of where the model succeeds and fails. A direct comparison across multiple LLMs (small vs. large open-weights, frontier API models) on the same benchmark would also help to separate the contribution of the model from the contribution of the prompt.

References

- [1] S. Schaal, “Is imitation learning the route to humanoid robots?” *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.
- [2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.
- [3] S. Calinon, “A tutorial on task-parameterized movement learning and retrieval,” *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.
- [4] A. Gaurav, K. Rezaee, “Learning soft constraints from constrained expert demonstrations.
- [5] M. Ahn et al., “Do as I can, not as I say: Grounding language in robotic affordances,” in *Proc. Conf. on Robot Learning (CoRL)*, 2022.
- [6] J. Liang et al., “Code as policies: Language model programs for embodied control,” in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2023.
- [7] W. Huang et al., “VoxPoser: Composable 3D value maps for robotic manipulation with language models,” in *Proc. Conf. on Robot Learning (CoRL)*, 2023.
- [8] Z. Yang et al., “Plug in the safety chip: Enforcing constraints for LLM-driven robot agents,” in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2024.

A System prompt (scaffolded variant)

```
1 You are a robotics safety engineer. Given a task, reason from first principles
  about the OBJECT being manipulated and identify any state-based safety
  constraints -- conditions on the end-effector kinematics that, if violated,
  would cause the task to fail in a concrete physical way.
2
3 FIRM-GRASP ASSUMPTION: assume the robot grips the object securely with a closed
  gripper for the entire motion. The object cannot fall out, roll out, or
  slide out of the gripper. Failure modes that depend on the object escaping
  the gripper do NOT apply.
4
5 REASONING PROCEDURE (follow these steps before answering):
6
7 Step 1. Identify the object and its relevant physical properties. Examples of
  properties that might matter: rigid vs deformable, sealed vs open, fragile
  vs sturdy, has-internal-contents vs solid, preferred-axis vs symmetric. You
  decide which properties matter for THIS object.
8
9 Step 2. From those properties, derive what could physically go wrong during
  transport. The failure must be a concrete physical event with identifiable
  cause -- not vague hand-waving. If you cannot describe the failure in
  mechanistic terms, the failure mode is not real and you should not propose a
  constraint for it. Most ordinary rigid objects have ZERO failure modes;
  producing an empty hypothesis list is the correct answer in that case.
10
11 Step 3. For each surviving failure mode, work out which kinematic variable of
  the end-effector physically governs it. Available variables will be provided
  in the user message; you must map your physical reasoning onto those
  specific variables. If a failure depends on a quantity the robot doesn't
  expose (e.g. internal sloshing dynamics), choose the closest available proxy
  and explain the connection.
12
13 Step 4. Formalise each surviving constraint into the output schema.
14
15 REASONING QUALITY:
16 - The 'reasoning' field must describe both (a) the physical mechanism of failure
  and (b) why a specific kinematic variable causes or measures it.
17 - Hollow restatements like "X happens if roll is too high" are not acceptable --
  describe WHY high roll causes X.
18 - Avoid vague qualifiers ("too much", "too far"); identify the specific physical
  threshold being crossed (a rim, a rated impact tolerance, a fracture limit,
  etc.).
19
20 OUTPUT FORMAT -- single JSON object, no prose, no markdown:
21
22 {
23   "hypotheses": [
24     {
25       "feature_name": "<short snake_case label>",
26       "failure_mode": "<concrete physical failure>",
27       "reasoning": "<which variable causes the failure and why>",
28       "constraint_type": "box" | "ball" | "halfspace",
29       "variables": ["<from available_features verbatim>"],
30       "model_prior": {"type": "poly", "degree": 1|2|3} or {"type": "nn"}
31     }
32   ]
33 }
34
35 DO NOT INCLUDE NUMERIC THRESHOLDS or symbolic equations. You identify WHICH
  variables matter and WHAT TYPE of region they should stay in (box, ball,
  halfspace). The downstream system fits the actual polynomial coefficients
  from the demonstration data.
```

```

36
37 GROUPING RULE: variables that are physically coupled go in ONE hypothesis. If
    two variables contribute to the same failure mode through the same mechanism
    , they belong together with multiple entries in the "variables" list, not in
    separate hypotheses.
38
39 CONSTRAINT-TYPE GUIDANCE:
40 - "box": axis-aligned bounds, used when each variable has an independent
    tolerance.
41 - "ball": ellipsoidal/spherical region, used when the failure depends on a
    combined magnitude (vector norm).
42 - "halfspace": one-sided linear bound, used when failure is one-directional.
43 - Use degree  $\geq 2$  polynomials for "box" and "ball" (degree 1 is linear and
    cannot represent a closed region). Use "nn" for non-convex regions.
44
45 VALID variable names: drawn from available_features verbatim.

```

B Ground-truth annotation file (excerpt)

```

1 Ground truth file (excerpt). The full file contains 18 cases; five
2 representative cases are shown below. Tilt thresholds are in radians,
3 speed thresholds in m/s, workspace radii in metres.
4
5 {
6   "cases": {
7     "cup_of_water": {
8       "instruction": "Move a cup of water to the table",
9       "expected_constraints": {
10        "tilt": {
11          "variables": [
12            "roll",
13            "pitch"
14          ],
15          "max_rad": 0.26,
16          "rationale": "open container; ~15 deg is the rim angle of a typical
    mug"
17        },
18        "speed": {
19          "variables": [
20            "vx",
21            "vy"
22          ],
23          "max_ms": 1.5,
24          "rationale": "above ~1.5 m/s in a curve the lateral acceleration
    sloshes water out"
25        }
26      }
27    },
28    "glass_vase": {
29      "instruction": "Move a thin glass vase",
30      "expected_constraints": {
31        "speed": {
32          "variables": [
33            "vx",
34            "vy"
35          ],
36          "max_ms": 1.0,
37          "rationale": "thin glass; impact-shock breakage at ~1 m/s deceleration
    "
38        }
39      }
40    },
41    "rubber_ball": {

```

```

42     "instruction": "Move a rubber ball",
43     "expected_constraints": {},
44     "rationale": "Same as tennis ball \u2014 robust, no contents."
45 },
46 "cup_of_coffee": {
47     "instruction": "Carry a coffee cup full of hot coffee",
48     "expected_constraints": {
49         "tilt": {
50             "variables": [
51                 "roll",
52                 "pitch"
53             ],
54             "max_rad": 0.20,
55             "rationale": "open container; burn-risk amplifies tilt sensitivity"
56         },
57         "speed": {
58             "variables": [
59                 "vx",
60                 "vy"
61             ],
62             "max_ms": 1.2,
63             "rationale": "sloshing"
64         }
65     },
66 },
67 "stack_of_books": {
68     "instruction": "Carry a stack of books",
69     "expected_constraints": {
70         "tilt": {
71             "variables": [
72                 "roll",
73                 "pitch"
74             ],
75             "max_rad": 0.15,
76             "rationale": "loose stack; top books slide off above ~9 deg"
77         },
78         "speed": {
79             "variables": [
80                 "vx",
81                 "vy"
82             ],
83             "max_ms": 1.0,
84             "rationale": "lateral acceleration shifts the stack"
85         }
86     },
87 },
88 },
89 }

```

C Per-task constraint figures

This appendix shows the standard pipeline output (top row: demonstration projected into the constraint subspace; bottom row: learned safe region with the ground-truth overlay) for each of the 10 benchmark cases that have a non-empty expected constraint set, except `cup_of_water` (which is the walk-through case in Section 4.2) and `sealed_water_bottle` (which is the failure case in Figure 7). Each figure uses a distinct random seed so the demonstration trajectory differs from case to case; the safe-region scale is set by the task’s learnt constraint limits. The seven robust-object cases (`wooden_block`, `tennis_ball`, `rubber_ball`, `potato`, `banana`, `brick`, `screwdriver`) are not shown: the LLM correctly returns an empty hypothesis list on all of them, so there is no constraint to render.

Liquid containers

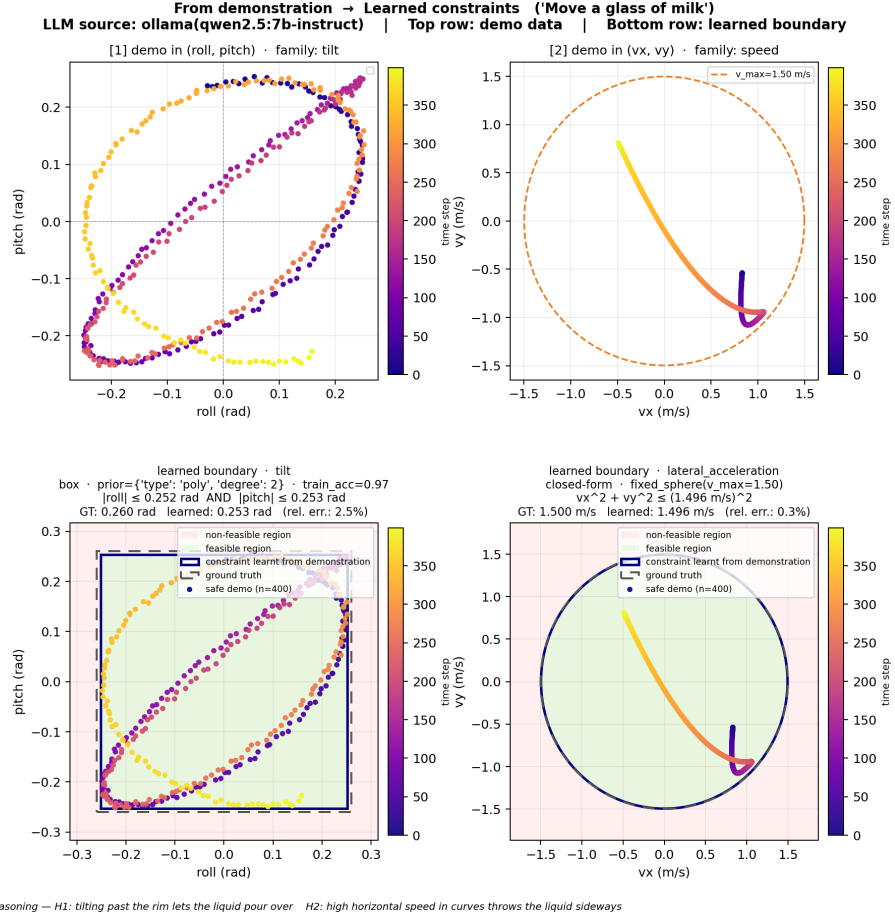
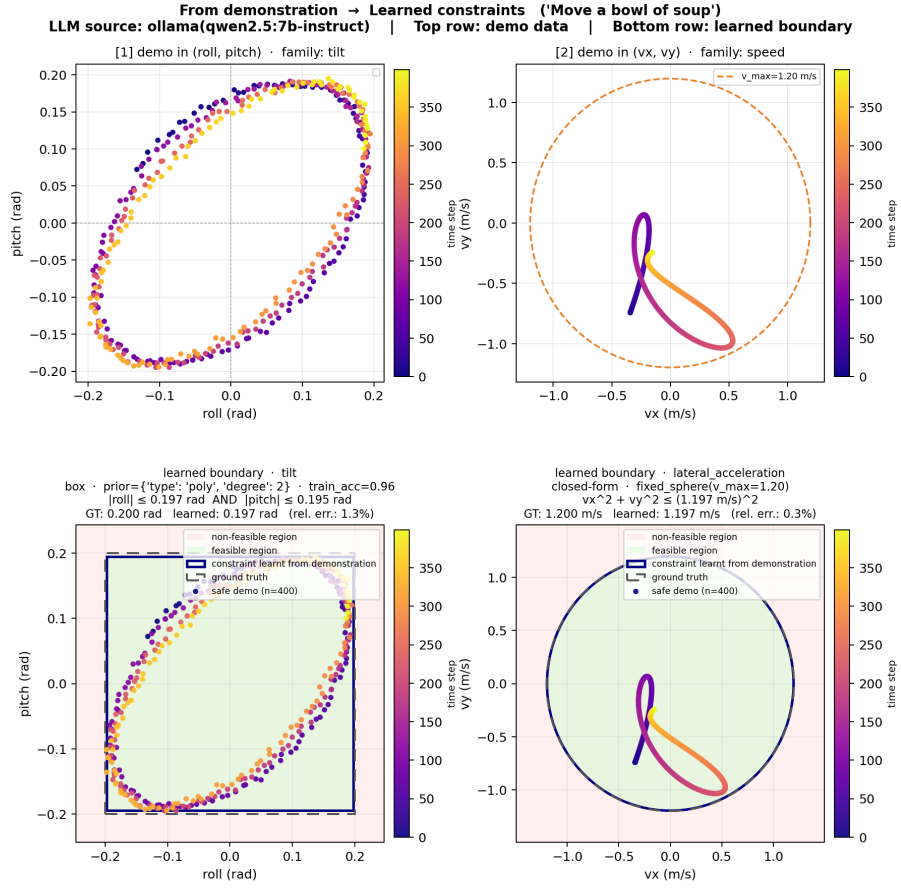


Figure 9: Constraint figure for the `glass_of_milk` case. Expected: tilt and speed. Learned: $|\phi|, |\theta| \leq 0.253 \text{ rad}$ (GT 0.260 rad; rel. err. 2.5%) and $\|v_h\| \leq 1.496 \text{ m/s}$ (GT 1.500 m/s; rel. err. 0.3%).



LLM reasoning — H1: open shallow container; rim is easily crossed H2: lateral acceleration causes the surface to slope

Figure 10: Constraint figure for the `bowl_of_soup` case. Expected: tilt and speed. Learned: $|\phi|, |\theta| \leq 0.197 \text{ rad}$ (GT 0.20 rad; rel. err. 1.3%) and $\|v_h\| \leq 1.197 \text{ m/s}$ (GT 1.200 m/s; rel. err. 0.3%).

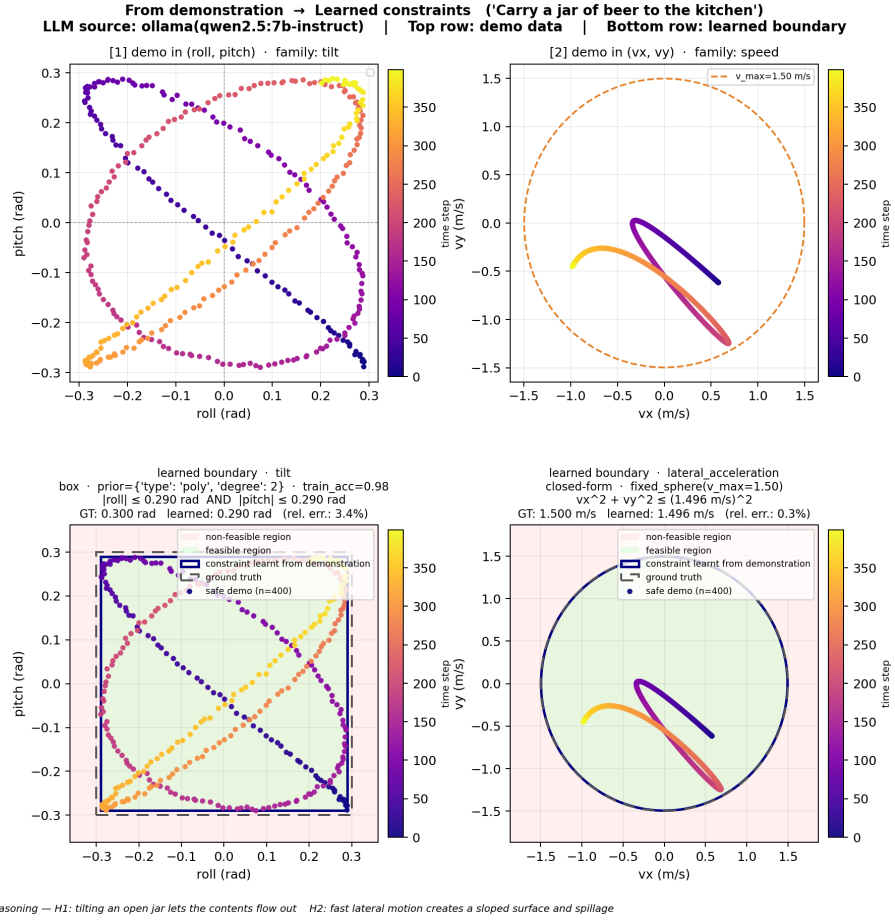
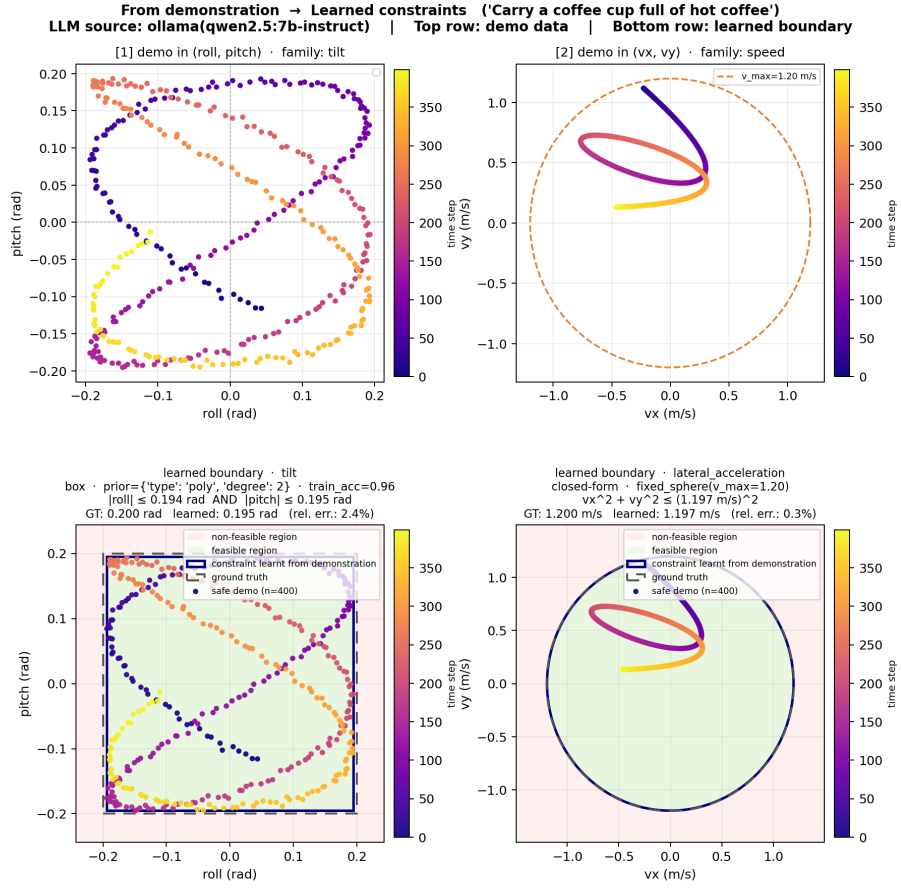


Figure 11: Constraint figure for the `jar_of_beer` case. Expected: tilt and speed. Learned bounds match the ground truth to within the same systematic tilt offset ($\sim 2.4\%$) and floating-point-level speed error ($\sim 0.4\%$).

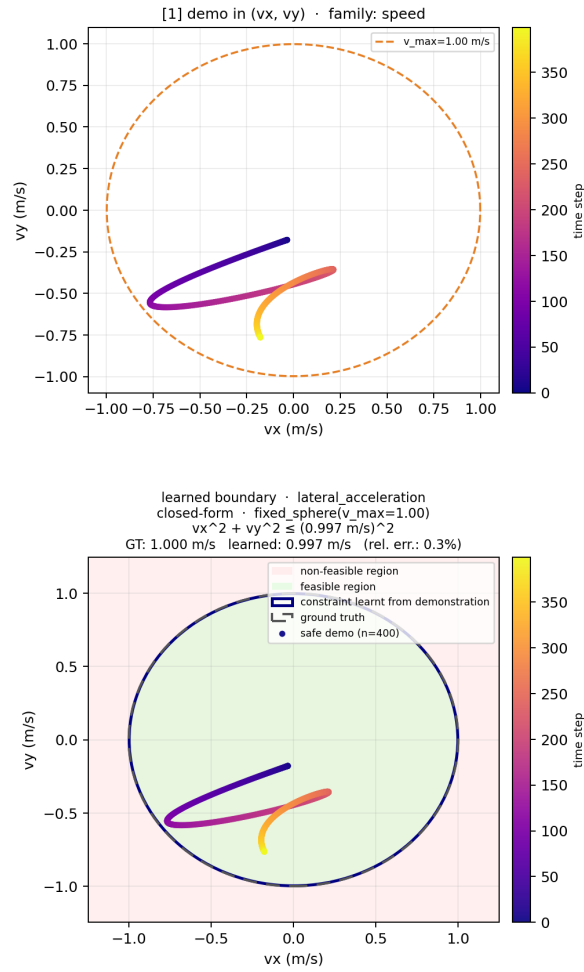


LLM reasoning — H1: tilting past the cup rim pours hot coffee out H2: high lateral speed causes sloshing and burns risk

Figure 12: Constraint figure for the cup of coffee case. Expected: tilt and speed. The ground-truth limits are tighter than the other liquid cases ($\theta_{\text{max}} = 0.20 \text{ rad}$, $v_{\text{max}} = 1.2 \text{ m/s}$), reflecting the higher consequence of spillage with hot coffee; both are recovered with the same low relative errors.

Fragile objects (speed only)

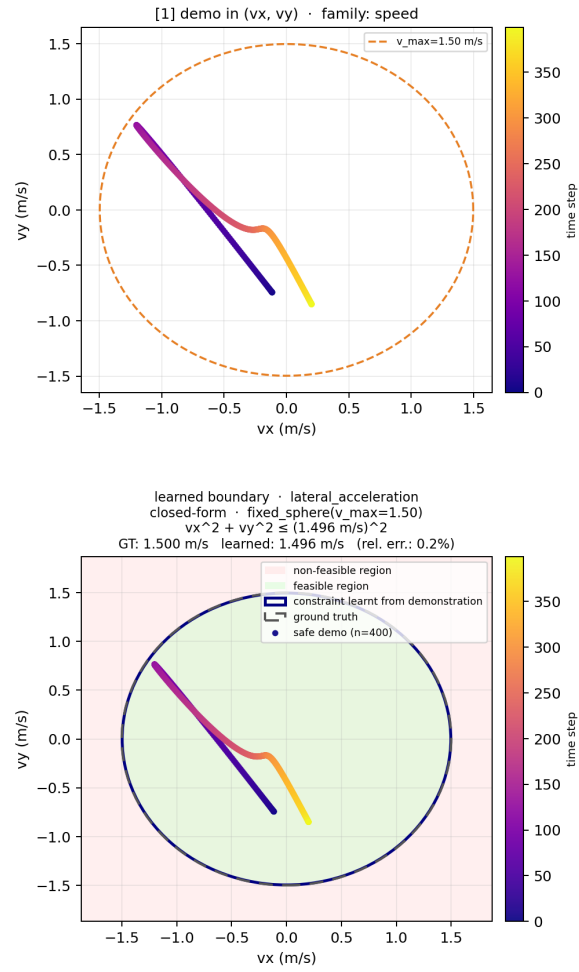
From demonstration → Learned constraints ('Move a thin glass vase')
 LLM source: ollama(qwen2.5:7b-instruct) | Top row: demo data | Bottom row: learned boundary



LLM reasoning — H1: rapid deceleration can crack or shatter the vase

Figure 13: Constraint figure for the `glass_vase` case. Expected: speed only (no tilt constraint under the firm-grasp assumption). The figure has a single panel because only one hypothesis was produced. Learned $\|v_h\| \leq 0.997 \text{ m/s}$ against GT 1.000 m/s .

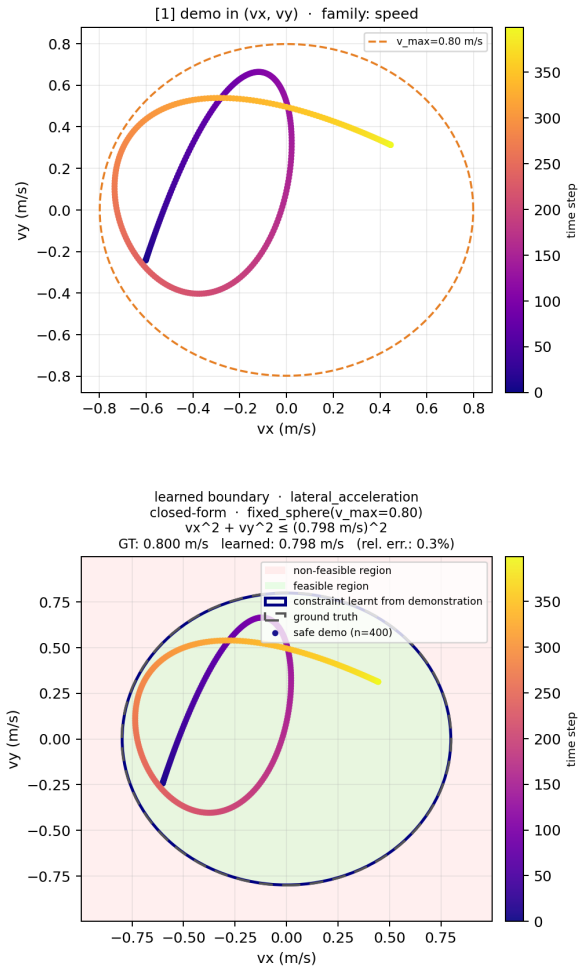
From demonstration → Learned constraints ('Move a ceramic plate')
 LLM source: ollama(qwen2.5:7b-instruct) | Top row: demo data | Bottom row: learned boundary



LLM reasoning — H1: brittle material fails on impact-shock from fast motion

Figure 14: Constraint figure for the `ceramic_plate` case. Expected: speed only. The speed bound reflects the brittle material's lower tolerance for impact-shock from sudden deceleration.

From demonstration → Learned constraints ('Move an egg')
 LLM source: ollama(qwen2.5:7b-instruct) | Top row: demo data | Bottom row: learned boundary



LLM reasoning — H1: the shell is brittle; fast deceleration can break it

Figure 15: Constraint figure for the **egg** case. Expected: speed only. The smallest speed bound in the benchmark; even modest lateral acceleration risks cracking the shell.

Special handling

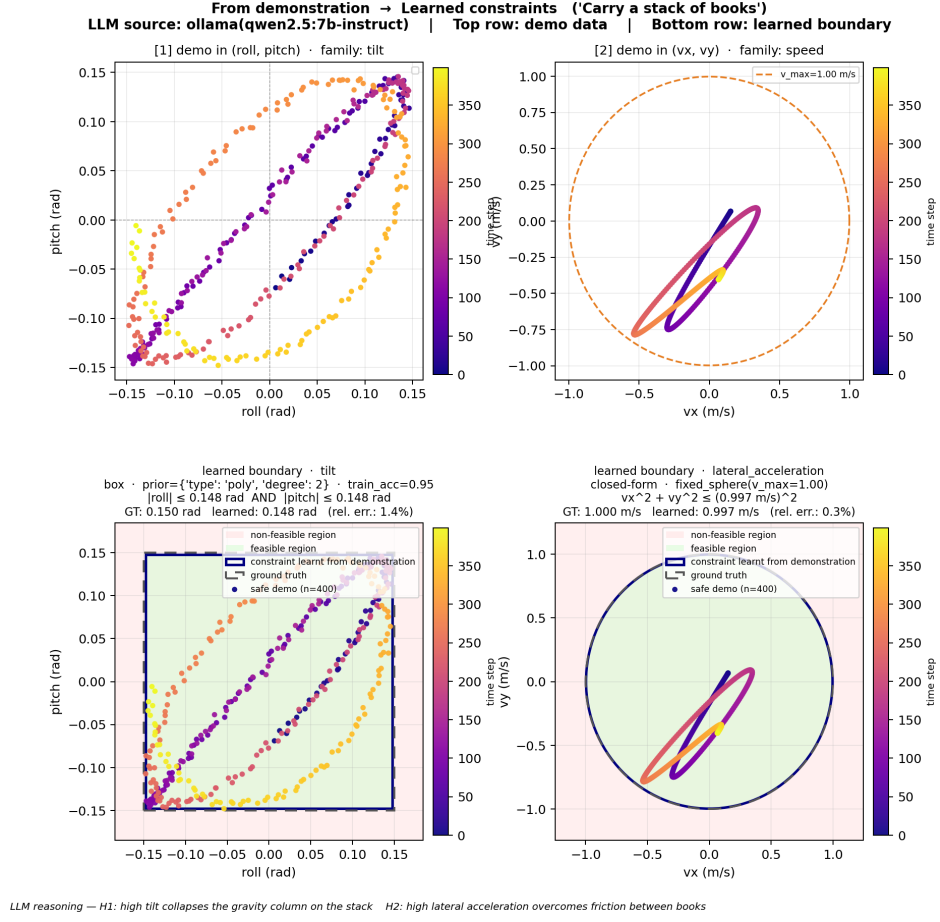
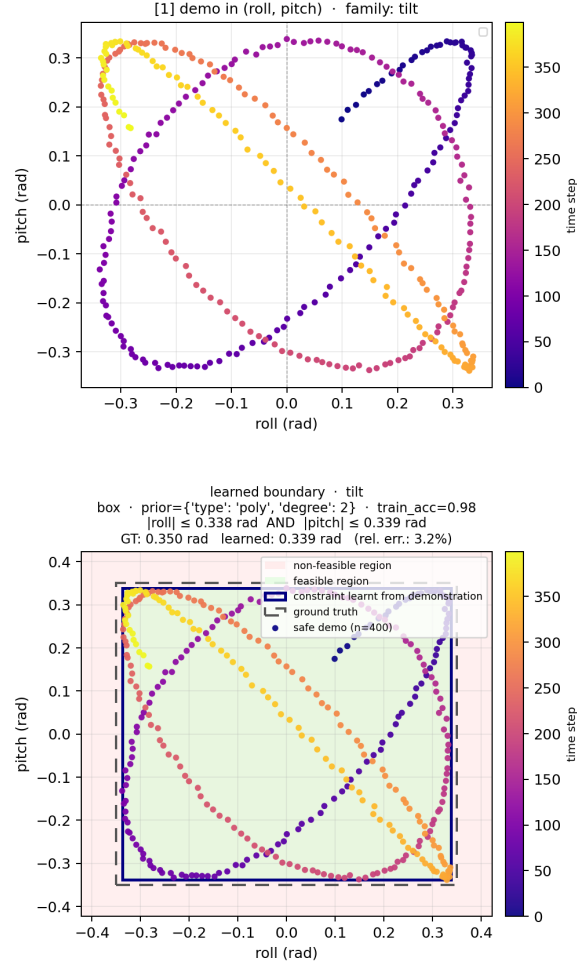


Figure 16: Constraint figure for the `stack_of_books` case. Expected: tilt and speed. The failure mode is mechanical (books sliding off the top) rather than spillage, but the constraint structure is identical to the liquid cases: tilt limits the column geometry, speed limits the lateral acceleration that overcomes inter-book friction.

From demonstration → Learned constraints ('Transport an open laptop')
 LLM source: ollama(qwen2.5:7b-instruct) | Top row: demo data | Bottom row: learned boundary



LLM reasoning — H1: an open laptop has a preferred up-axis with no spillage failure

Figure 17: Constraint figure for the `open_laptop` case. Expected: tilt only. The figure has a single panel because no speed constraint applies (the failure mode is the screen falling closed, which depends on orientation, not velocity). The widest tilt rim in the benchmark ($\theta_{\max} = 0.35 \text{ rad} \approx 20^\circ$) reflects the laptop’s mechanical flexibility around its preferred up-axis.