Feasibility (and what will *actually* make this publishable)

What you described is **feasible** in IsaacLab/Isaac Sim: you can train an RL pick-and-place policy using **RGB-D + proprio**, then evaluate it under **vision modality dropout**, then train a second policy that additionally observes **gripper force / contact signals**, and compare robustness.

However, there are two "gotchas" that will determine whether your paper reads like a solid contribution versus an obvious demo:

1. **Force sensing will not help if vision dropout happens "too early."**
   If the camera drops out **before the robot has localized the object**, force doesn't provide object position—so both models will fail similarly.
   → To make the force sensor matter, your evaluation should include **phase-aware dropout** (e.g., during grasp closure / after grasp) and/or a task where the **goal is known** but the **object must be acquired**.
2. **If you don't train with any modality corruption, the force-sensor model may still ignore force.**
   Even if force is present in the observation, the easiest signal is usually vision. If vision is always perfect during training, the network can learn a "vision-only" solution and treat force as noise.
   → For a publication-quality robustness claim, you should include **at least one robust-training baseline** (vision dropout/noise during training), otherwise reviewers will say "you could have just trained with dropout."

Recommended change (be specific)

Keep your original two-model story, but **add one extra baseline** so your conclusion is defensible:

- **M1 (baseline):** RGB-D + proprio, trained with clean vision
- **M2 (your proposed):** RGB-D + proprio + force, trained with clean vision
- **M3 (recommended baseline):** RGB-D + proprio, trained with *vision corruption/dropout*
- **M4 (best-method, optional but strong):** RGB-D + proprio + force, trained with *vision corruption/dropout*

This lets you separate:

- "force helps" vs.
- "dropout training helps" vs.
- "force + dropout training is best."

If you only do M1 vs M2, you risk a weak or inconsistent effect.

The paper's core claim (make it crisp)

A clean publication framing:

Claim: Adding gripper contact/force sensing improves robustness of vision-based RL manipulation under intermittent vision failures; robustness depends strongly on *when* and *for how long* vision is missing.

Your contribution becomes:

1. A reproducible IsaacLab benchmark: RGB-D pick-and-place with controlled vision failures
2. A multi-modal policy with force/contact input
3. A systematic study of **dropout duration + dropout timing** sensitivity

---

Experimental design that won't collapse under review

Task: simple but not trivial

Use a single object pick-and-place with randomized object pose:

- Robot: **Franka Panda** (common, good controllers, easy)
- Object: cube or cylinder (start with cube)
- Table scene: uncluttered (for "simple")
- Initial object pose: random in a rectangle
- Goal: a bin/target region at a known pose
  Important: provide goal pose as a **command input** (low-dimensional), not only visually. This makes "vision dropout after grasp" meaningful.

Episode structure

- Control frequency: **20 Hz**
- Episode length: **250 steps** (12.5 s)
- Reset randomization each episode

Success definition (write it clearly)

Success at end of episode or terminal condition if all true:

- Object position within goal radius:

  ‖pobj−pgoal‖<3 cm\|p_{obj}-p_{goal}\| < 3\text{ cm}‖pobj−pgoal‖<3 cm

- Object height near goal surface (not hovering): e.g. ∣zobj−zgoal∣<2 cm∣z_{obj} - z_{goal}∣ < 2\text{ cm}∣zobj−zgoal∣<2 cm
- Object linear speed small (stable): ∥vobj∥<0.05 m/s\∣v_{obj}\∣ < 0.05\text{ m/s}∥vobj ∥<0.05 m/s
- Gripper open (released), optional but recommended

Observations (what each model sees)

Common proprio + command (both models)

Include:

- Joint positions qqq (7)
- Joint velocities q˙\dot qq˙ (7)
- Gripper width / finger positions (1–2)
- End-effector pose (position + orientation) OR (position + quaternion) (7)
  (This is still "proprio-derived," not external perception.)
- Previous action (helps stability) (action_dim)
- **Goal pose command** in robot base frame: pgoalp_{goal}pgoal (3) (and optionally yaw)

This matters because with vision dropout, the robot still needs to know where "place" is.

Vision modality (RGB-D)

- One camera (choose one and justify):
  - **Wrist-mounted** camera: best for grasp; more sensitive to occlusion
  - **Static overhead**: best for global localization; less realistic for many setups
- Resolution: **84×84** or **96×96** (start small; render cost dominates)
- Inputs:
  - RGB normalized to [0,1]
  - Depth clipped to [0, dmaxd_{max}dmax] and normalized (e.g., dmax=1.5d_{max}=1.5dmax=1.5 m)
- Optional: 2-frame stack (gives motion cues)

Force / contact modality (only M2/M4)

In simulation you can get this from PhysX contacts at the fingertips or gripper links.

Use a compact, learnable signal:

- Normal contact force magnitude per finger: FL,FRF_L, F_RFL,FR (2)
- Tangential magnitude per finger (optional): TL,TRT_L, T_RTL,TR (2)
- Binary contact indicators cL,cRc_L, c_RcL,cR (2)

- Optional: estimated slip proxy (relative tangential velocity at contact)

Recommended final force feature vector (8 dims):

oforce=[FL,FR,TL,TR,cL,cR,ΔF,Fsum]o_{force}=[F_L,F_R,T_L,T_R,c_L,c_R,\Delta F, F_{sum}]oforce=[FL,FR,TL,TR,cL,cR,ΔF,Fsum]

Normalize forces by Fnorm=20 NF_{norm}=20\text{ N}Fnorm=20 N (or your chosen limit).

Also add realistic noise:

- Force noise: N(0,0.2 N)\mathcal{N}(0,0.2\text{ N})N(0,0.2 N)
- Low-pass filter: Ft=0.8Ft−1+0.2FtrawF_t = 0.8F_{t-1}+0.2F^{raw}_tFt=0.8Ft−1 +0.2Ftraw

---

Actions (keep it stable)

Use an operational-space / Cartesian delta action (easier than direct torques):

Action vector:

- Δx,Δy,Δz\Delta x, \Delta y, \Delta zΔx,Δy,Δz (m per step, clipped e.g. ±0.03 m)
- Δr,Δp,Δy\Delta r, \Delta p, \Delta yΔr,Δp,Δy (rad per step, clipped e.g. ±0.15 rad)
- Gripper command g∈[−1,1]g\in[-1,1]g∈[−1,1] (close/open)

Then IsaacLab controller converts to joint targets/torques.

Policy architecture (publishable and robust)

You want something reviewers recognize as sensible, not fragile:

Encoder + fusion

- **Vision encoder:** small CNN (or ResNet-like) on 4 channels (RGBD)
  - Output 256-d embedding
- **Proprio/goal encoder:** 2-layer MLP → 128-d
- **Force encoder:** 2-layer MLP → 64-d (only for force models)
- Fuse by concatenation → 256–384 dim
- Optional but strong: **GRU/LSTM (128)** after fusion
  This is *very* relevant to “dropout duration” studies.

Actor-Critic

- PPO with Gaussian policy (continuous control)
- **Asymmetric critic** (recommended): critic can see privileged state (object pose) to improve learning stability, while actor only sees sensors. This is widely accepted in sim RL.

Training algorithm and settings (practical defaults)

Use PPO (common in IsaacGym/IsaacLab workflows):

- $\gamma=0.99$ \gamma=0.99
- GAE $\lambda=0.95$ \lambda=0.95
- PPO clip $\epsilon=0.2$ \epsilon=0.2
- Learning rate: 3×10^{-4} 3×10−43\times 10^{-4}
- Entropy coef: 0.001 (or 0.0 if too stochastic)
- Value loss coef: 0.5
- Max grad norm: 1.0
- Rollout length: 32–64 steps
- Batch size: as large as GPU allows (vision limits env count)
- Envs: start with 32–128 (vision rendering is the bottleneck)

Train until plateau, but expect:

- M1 (clean vision) reaches >80% faster
- robust variants may need 1.5–3× steps

Reward design (explicit, weighted, and consistent)

You asked for a specific reward/penalty system and weights. The key is: **stage-gated shaping** so the agent doesn't "cheat" by moving toward the goal before grasping.

Below is a reward that works well for pick-and-place and is easy to justify.

Definitions (computed from simulator state, not observations)

- $d_{eo} = \|p_{ee}-p_{obj}\|$ deo=‖pee−pobj‖d_{eo} = \|p_{ee}-p_{obj}\| end-effector to object distance
- $d_{og} = \|p_{obj}-p_{goal}\|$ dog=‖pobj−pgoal‖d_{og} = \|p_{obj}-p_{goal}\| object to goal distance
- $h = z_{obj}$ h=zobjh = z_{obj}

- hlift=0.10h_{lift}=0.10hlift=0.10 m (lift threshold)
- aaa = action vector
- Igrasp∈{0,1}I_{grasp}\in\{0,1\}Igrasp∈{0,1}: object grasped (contact + gripper width condition)
- Ilift∈{0,1}I_{lift}\in\{0,1\}Ilift∈{0,1}: h>htable+hlifth>h_{table}+h_{lift}h>htable +hlift
- Isucc∈{0,1}I_{succ}\in\{0,1\}Isucc∈{0,1}: success condition
- Idrop∈{0,1}I_{drop}\in\{0,1\}Idrop∈{0,1}: object fell off / dropped (e.g., z<ztable−0.02z<z_{table}-0.02z<ztable−0.02 or out of bounds)
- Icol∈{0,1}I_{col}\in\{0,1\}Icol∈{0,1}: undesired collision (arm-table, self-collision)

Shaping terms

Use smooth exponentials:

- Reach:

  rreach=exp⁡(−10 deo)r_{reach}=\exp(-10\, d_{eo})rreach=exp(−10deo)

- Goal transport:

  rgoal=exp⁡(−6 dog)r_{goal}=\exp(-6\, d_{og})rgoal=exp(−6dog)

- Lift progress (linear, clipped):

  rlift=clip(h−(htable+0.02)hlift,0,1)r_{lift}=\text{clip}\left(\frac{h-(h_{table}+0.02)}{h_{lift}},0,1\right)rlift=clip(hlifth−(htable+0.02),0,1)

Full per-step reward (M1 / M3: vision+proprio)

rt=
1.0 rreach+Igrasp·1.5+Igrasp·2.0 rlift+Ilift·3.0 rgoal−0.01−0.002 ‖a‖2−1.0 Icol−5.0 Idrop\begin{aligned} r_t =\;& 1.0\,r_{reach} \\ &+ I_{grasp}\cdot 1.5 \\ &+ I_{grasp}\cdot 2.0\,r_{lift} \\ &+ I_{lift}\cdot 3.0\,r_{goal} \\ &- 0.01 \\ &- 0.002\,\|a\|^2 \\ &- 1.0\,I_{col} \\ &- 5.0\,I_{drop} \end{aligned}rt=1.0rreach+Igrasp·1.5+Igrasp·2.0rlift+Ilift·3.0rgoal−0.01−0.002‖a‖2−1.0Icol −5.0Idrop

Terminal success bonus (and terminate episode):

rT+=10.0 Isuccr_T \mathrel{+}= 10.0\,I_{succ}rT+=10.0Isucc

Force-aware reward addition (M2 / M4: +force)

You have two options:

Option A (recommended): Keep reward identical across models

This isolates the effect of adding force to observation. It's cleaner scientifically.

Option B (also publishable): add a small grasp-quality regularizer

This encourages using force without dominating learning.

Let Fsum=FL+FRF_{sum}=F_L+F_RFsum=FL+FR (normal forces), and define a desired safe band:

- Fmin=2F_{min}=2Fmin=2 N (too weak → slip likely)
- Fmax=15F_{max}=15Fmax=15 N (too strong → "crushing" / unstable contact in sim)

Penalty only when grasped:

rforce=−0.05 Igrasp[max(0,Fmin−Fsum)+max(0,Fsum−Fmax)]r_{force} = -0.05\,I_{grasp}\left[\max(0, F_{min}-F_{sum}) + \max(0, F_{sum}-F_{max})\right]rforce =−0.05Igrasp[max(0,Fmin−Fsum)+max(0,Fsum−Fmax)]

Add:

rt+=rforcer_t \mathrel{+}= r_{force}rt+=rforce

This is small relative to the lift/goal rewards and mainly shapes contact behavior.

Publication advice: If you use Option B, you must also report Option A as an ablation, otherwise reviewers will say "it's reward shaping, not sensing."

Vision modality dropout: how to do it so the results are interpretable

You need two separate things:

1. **Dropout mechanism definition** (exactly how you corrupt observations)
2. **Evaluation protocol** (when dropout happens, how long, how often)

Dropout types (you should include at least 2)

Define "vision" = {RGB, Depth} jointly (main), and optionally separate them.

1. **Hard dropout (blackout):**
   - RGB set to 0
   - Depth set to dmaxd_{max}dmax (or 0) consistently
2. **Noise corruption:**

    - RGB: add Gaussian noise + blur + brightness jitter
    - Depth: Gaussian noise + missing pixels (set to dmaxd_{max}dmax)
      Keep it bounded so it resembles sensor failure, not random TV static.

Important recommendation: include a "camera health flag" baseline
Real systems often know if a camera stream is missing. You can evaluate both:

- **Unannounced dropout:** no flag (harder)
- **Announced dropout:** add a 1-bit "vision_valid" to observations (more realistic)

This can be a strong discussion point.

Dropout schedule generator (concrete)

Do **event-based dropout**:

At each timestep:

- If not currently in dropout, start dropout with probability pstartp_{start}pstart
- If started, sample duration LLL steps and apply corruption for LLL steps

Parameters:

- pstart=0.02p_{start}=0.02pstart=0.02 (roughly one dropout every ~50 steps on average)
- Durations L∈{0,5,10,20,40,80}L\in\{0,5,10,20,40,80\}L∈{0,5,10,20,40,80} steps
  At 20 Hz: {0, 0.25s, 0.5s, 1s, 2s, 4s}

You can also sample LLL from a distribution (geometric or log-uniform) for smoother curves.

The "dropout duration affects success most" study (make it rigorous)

If you only vary duration, you'll get messy results because the **timing** matters more than length. A 0.5s dropout during grasp closure can be worse than a 2s dropout during transport.

Do this as a 2D sensitivity map

Evaluate success as a function of:

- **dropout onset time** (or phase)
- **dropout duration**

Two recommended variants:

Variant 1 (time-based grid)

- Onset bins: t/T∈{0–0.2,0.2–0.4,0.4–0.6,0.6–0.8,0.8–1.0}t/T \in \{0–0.2, 0.2–0.4, 0.4–0.6, 0.6–0.8, 0.8–1.0\}t/T∈{0–0.2,0.2–0.4,0.4–0.6,0.6–0.8,0.8–1.0}
- Durations: as above
- Report a heatmap of success rate

Variant 2 (phase-based, best for manipulation)

Define phases using privileged signals:

- Phase A: Reach (before first contact)
- Phase B: Grasp closure (first contact → grasp confirmed)
- Phase C: Lift/transport (grasp confirmed → near goal)
- Phase D: Place/release (near goal → released)

Then apply dropout starting at the phase boundary, varying duration.

This will almost certainly show:

- Vision dropout is most critical in Phase A/B
- Force helps most in Phase B/C/D
- Duration sensitivity differs by phase

That is a publishable result.

Quantify "most affected"

Don't just eyeball curves—compute:

- S(L)S(L)S(L): success vs duration
- "critical duration" L50L_{50}L50: duration where success drops to 50% of clean-vision performance
- Fit a simple decay model per phase:

  S(L)=S∞+(S0−S∞)exp⁡(−αL)S(L)=S_{\infty} + (S_0-S_{\infty})\exp(-\alpha L)S(L)=S∞+(S0−S∞)exp(−αL)

  Compare α\alphaα between models/phases.

Training plan (what you should actually run)

Phase 1: Get M1 to >80% success (clean vision)

- Moderate domain randomization (don’t overdo initially):
  - Object XY position
  - Object yaw
  - Mass/friction within small ranges
  - Camera jitter (small)
  - Lighting randomization (small)
- No dropout during training

Phase 2: Evaluate M1 under dropout (should drop)

Report:

- clean success (target >80%)
- dropout success vs L and vs phase
- noise vs hard-dropout

Phase 3: Train M2 (+force), same conditions as M1

- Same reward, same training steps budget, same seeds count
- Evaluate under dropout the same way

Phase 4 (recommended for publication): train robust baselines

Train M3 and optionally M4 with **vision corruption during training**:

- Start corruption only after N million steps (curriculum), or ramp probability:
  - pstartp_{start}pstart linearly 0 → 0.02
  - duration cap 0.25s → 2s over training
- Keep corruption mild enough that learning still happens

This yields a compelling story:

- Clean-vision policies fail under intermittent sensor failures
- Robust-training helps
- Force sensing reduces brittleness further, especially in grasp/placement phases

---

Domain randomization (keep it realistic and targeted)

If your goal is robustness to **vision dropout**, you still need basic visual generalization:

Randomize:

- Textures (table/object)

- Lighting intensity/direction
- Camera extrinsics jitter (a few mm, <1°)
- Depth noise model (Gaussian + missing pixels)
- Object friction/mass moderately

But do **not** randomize so aggressively that your clean-vision baseline can't learn—reviewers want controlled experiments.

Evaluation protocol (what to report so reviewers trust you)

Metrics (minimum)

- Success rate (%)
- Grasp success rate (%)
- Place success given grasp (%)
- Mean time-to-success (steps)
- Failure mode breakdown (drop, miss-grasp, collision, timeout)

Statistical practice (do this)

- ≥ **3 random seeds** per method (5 is better)
- ≥ **500–2000 evaluation episodes** per condition (dropout durations multiply conditions; you can reduce by using confidence intervals smartly)
- Report **95% binomial confidence intervals** on success rates

Compute fairness

Keep:

- same training steps
- same network size except added force input encoder
- same evaluation seeds/episodes

How to structure the paper (a complete writing guide)

Title ideas

- "Robust Vision-Based RL Manipulation Under Intermittent Sensor Failures via Gripper Force Sensing"
- "On the Sensitivity of Vision-Based Pick-and-Place Policies to Modality Dropout and the Role of Force Feedback"

Abstract (template)

1–2 sentences motivation (vision failures happen).
1 sentence method (train RGB-D+proprio RL; evaluate dropout; add force).
1 sentence results (force improves robustness, especially in grasp/place phases; quantify duration sensitivity).
1 sentence contribution (benchmark + protocol + findings).

1. Introduction

- Real robots lose frames / have glare / occlusions.
- Vision-only manipulation policies can be brittle.
- Force sensing is complementary: local contact info, grasp stability.
- Contributions list (bullet points).

2. Related Work

Organize into 3 buckets:

- Vision-based RL for manipulation
- Multi-modal sensing (vision + tactile/force)
- Robustness methods: modality dropout, domain randomization, sensor failure

3. Method

3.1 Environment (IsaacLab)

- Robot, object, camera placement, control frequency
- Observation definitions (RGB-D, proprio, force)
- Action space + controller

3.2 Policy architecture

- CNN for RGB-D, MLP for proprio/force, fusion, actor-critic
- Optional recurrence (if used)

3.3 Reward function

Include the exact equation and weights (use the ones above).
Explain stage-gating rationale.

3.4 Vision dropout modeling

- Hard dropout + noise dropout definitions
- Event-based dropout schedule
- Phase-based protocol

4. Experiments

- Training details (PPO hyperparameters, steps, env count, seeds)
- Baselines (M1–M4)
- Evaluation suite (durations, phases, announced/unannounced)

5. Results

Must include:

- Table: clean-vision success for all models
- Plot: success vs dropout duration (overall)
- Heatmap: success vs (phase or onset time) × duration
- Ablation: force-only effect vs dropout-training effect
- Failure mode analysis (qualitative + quantitative)

6. Discussion

- Where force helps (grasp confirmation, slip mitigation, placement contact)
- Where it doesn't (object localization under full blackout pre-grasp)
- Practical implications: sensor redundancy, runtime detection of sensor failure

7. Limitations

Be honest:

- Sim contact models differ from real tactile sensors
- Rendering constraints limit env count
- Only single-object pick-and-place

8. Reproducibility

Include:

- Isaac Sim / IsaacLab version
- Full hyperparameters
- Random seeds
- Evaluation scripts for dropout schedules

Practical "don't fail in week 3" implementation tips (IsaacLab + vision RL)

- **Vision rendering is the bottleneck.** Keep image small (84×84), reduce camera count.

- Use **frame skipping** cautiously (can destabilize grasp).
- Normalize everything (proprio, forces, depth).
- Start with **depth-only** debug runs if RGB training is unstable; then add RGB.
- Add **action smoothing** (either in controller or via penalty) to avoid oscillations.

Expected outcomes (so you can sanity-check)

If you implement the above:

- **M1 (vision+proprio, clean training):**
  - Clean success: >80% achievable
  - Under dropout:
    - Huge drop if dropout hits Phase A/B
    - Moderate drop if dropout hits Phase C/D
- **M2 (+force, clean training):**
  - Clean success: similar to M1
  - Under dropout:
    - Improvement mainly in Phase B/C/D
    - Little/no improvement in Phase A
- **M3 (vision+proprio + dropout training):**
  - Clean success: slightly lower or similar
  - Under dropout: improved across many durations
- **M4 (+force + dropout training):**
  - Best robustness, especially long dropouts during grasp/placement

That pattern is coherent and defensible.

If you only do *one* thing to upgrade this to "publication-ready"

Do the **phase-based dropout** analysis (not just random duration). That is where your "duration affects most" question becomes a real result instead of a noisy curve.

If you want, I can also draft:

- a full **Methods** section text (ready-to-paste),
- an **Experiment table layout** (conditions × durations × phases),
- and pseudocode for the dropout scheduler and phase detector (in a paper-friendly format).