What your “bad news” actually implies (and why it’s not fatal)

If your **actors were trained with privileged state inputs** (cube position and goal position), then:

- Your policies are **not “vision-based”** in the publishable sense, even if an RGB-D stream existed in the observation.
- Any evaluation that **drops vision** (RGB-D blackout/noise) will likely show **little to no degradation**, because the policy can solve pick-and-place from the cube/goal positions alone.
- The *real* brittleness will show up when you degrade **the privileged object pose signal** (which, in a real system, is exactly what a vision perception pipeline provides).

This is actually a clean pivot: you can turn this into a solid short paper by reframing “vision dropout” as **perception/estimation dropout** (object pose estimate missing, stale, or noisy) and studying whether **force sensing adds redundancy**.

Even better: you can make your situation itself a contribution: **modality collapse / dominance**

When a low-dimensional privileged modality (object pose) is available during training, the policy often ignores high-dimensional vision. Therefore “vision robustness” can be illusory unless you explicitly enforce multi-modal usage.

That is a very defensible, conference-sized story.

New contribution you can claim *using your current M1–M4*

Contribution statement (tight and publishable)

1. **Diagnose modality dominance** in multi-modal RL manipulation: when privileged object pose is present, policies become insensitive to RGB-D corruption but highly sensitive to object-pose estimation failures.
2. **Quantify robustness** under perception outages using a controlled “pose-dropout/noise/delay” benchmark with **phase- and duration-conditioned dropouts**.
3. **Show redundancy from contact/force sensing**: adding gripper force/contact improves recovery under certain classes of perception failure (especially after initial contact / during transport and placement).

That is coherent even if your training was privileged.

Step 1 — Re-define “vision dropout” as “perception output dropout” (object pose dropout)

In real robots, the cube position you used is what you’d get from:

- RGB-D detection + pose estimation, or
- tracking + filtering.

So you can reframe your corruption tests as:

“Dropout/noise in the object pose estimate (perception module failure),” not raw camera failure.

Implement pose corruption at the observation wrapper (no retraining needed to start)

Let the policy input contain pobjp_{obj}pobj (cube position) and pgoalp_{goal}pgoal (goal position). Keep pgoalp_{goal}pgoal always available (reasonable; goals are typically known).

Create corruption modes for pobjp_{obj}pobj:

A) Hard dropout (missing measurement)

- Replace cube position with zeros or a sentinel value:
  - pobj←[0,0,0]p_{obj} \leftarrow [0,0,0]pobj←[0,0,0]
- **Optional but recommended:** add a 1-bit flag `obj_pose_valid` to observation
  If you cannot change obs dims now, skip it; but note in limitations.

B) Freeze / stale measurement (most realistic)

- Hold last valid measurement for LLL steps:
  - pobj(t)←pobj(tlast_valid)p_{obj}(t) \leftarrow p_{obj}(t_{last_valid})pobj (t)←pobj(tlast_valid)

This mimics camera freeze / tracker stuck.

C) Delay

- Use delayed measurement by kkk steps:
  - pobj(t)←pobj(t−k)p_{obj}(t) \leftarrow p_{obj}(t-k)pobj(t)←pobj(t−k)

D) Noise + drift (miscalibration / bad depth)

- Add Gaussian noise and slowly varying bias:
  - pobj←pobj+ϵ+bp_{obj} \leftarrow p_{obj} + \epsilon + bpobj←pobj+ϵ+b
  - ϵ∼N(0,σ2I)\epsilon \sim \mathcal{N}(0, \sigma^2 I)ϵ∼N(0,σ2I)

- $b(t)=0.99\,b(t-1)+0.01\,\eta$ $b(t)=0.99\,b(t-1) + 0.01\,\eta$ $b(t)=0.99b(t−1)+0.01η$, $\eta \sim \mathcal{N}(0,\sigma_b^2 I)$ $\eta \sim N(0,σb2I)$

Suggested numeric settings (good starting point):

- Noise $\sigma \in \{0, 5, 10, 20\}$ $\sigma∈{0,5,10,20}$ mm
- Drift $\sigma_b = 1$ $\sigma b=1$ mm per step equivalent (small persistent bias)

Step 2 — Make your M1–M4 "meaningful" under this new framing

You said you already have M1–M4. Under the new framing, you want them to correspond to:

- **M1:** privileged pose + proprio (baseline)
- **M2:** privileged pose + proprio + force/contact (redundancy)
- **M3:** *robust-trained* version of M1 (trained with pose corruption)
- **M4:** *robust-trained* version of M2 (trained with pose corruption)

If your current M3/M4 were trained with *vision* dropout (RGB-D), they may be redundant

Because pose is privileged and dominates, "vision dropout training" often changes nothing.

Recommendation (specific, minimal change):

- Treat your existing M1 and M2 as "pretrained checkpoints."
- Produce **new** M3 and M4 by **fine-tuning**:
  - Continue PPO training from M1 → M3 with **object pose corruption during training**
  - Continue PPO training from M2 → M4 with **object pose corruption during training**
- This is fast because the task is already learned; you're training robustness, not competence.

Fine-tune budget (practical):

- 10–30% of the original training timesteps is usually enough to see robustness gains.
- Use a curriculum (below) so you don't destroy the policy early.

Step 3 — Training-time corruption curriculum (for M3/M4)

You want the policy to keep high clean success **and** become robust.

Curriculum schedule (concrete)

Let pstartp_{start}pstart be probability to begin a corruption event at any step.
Let LLL be duration in steps (20 Hz control assumed).

Start simple; ramp up:

Phase 0 (stabilize): first 5% of fine-tuning

- pstart=0.00p_{start}=0.00pstart=0.00

Phase 1: next 35%

- pstart=0.01p_{start}=0.01pstart=0.01
- L∈{5,10}L \in \{5, 10\}L∈{5,10} steps (0.25–0.5 s)
- corruption type: **freeze** only (stale measurement)

Phase 2: next 40%

- pstart=0.02p_{start}=0.02pstart=0.02
- L∈{5,10,20,40}L \in \{5,10,20,40\}L∈{5,10,20,40} (0.25–2 s)
- corruption types mix:
  - 50% freeze
  - 25% hard dropout
  - 25% noise (σ=10\sigma=10σ=10 mm)

Phase 3: final 20%

- pstart=0.03p_{start}=0.03pstart=0.03
- L∈{10,20,40,80}L \in \{10,20,40,80\}L∈{10,20,40,80} (0.5–4 s)
- include drift + delay occasionally (e.g., 10% of events)

This curriculum is *exactly* the "robustness training baseline" reviewers expect.

Reward

Do **not** change reward now unless you must. Changing reward midstream makes comparisons harder to defend.

In the paper, be explicit:

- "All models share identical reward; only observations/training corruptions differ."

If you *do* change anything for robustness, make it a tiny regularizer and ablate it. But best is: **same reward across M1–M4**.

Step 4 — Evaluation suite (this becomes your main technical content)

You can reuse the evaluation framework I suggested earlier almost 1:1, just replacing "vision dropout" with "object pose dropout."

4.1 Baseline diagnostic: modality sensitivity / collapse

This is the key "you accidentally trained privileged" pivot.

Run each policy under these test-time ablations:

- **Ablate RGB-D** (if RGB-D exists in obs): blackout/noise
- **Ablate object pose** (hard dropout / freeze)
- **Ablate force** for force policies: set forces to zero + flags off

Report success rate and failure modes.

Expected publishable finding:

- Vision ablation barely changes success (policy ignores it).
- Pose ablation kills success (policy depends on it).
- Force helps mainly when pose is stale/missing *after contact*.

That's a clean story and turns "bad news" into a result.

4.2 Duration study (what duration hurts most)

Use your earlier duration set, but applied to pose corruption events:

Durations $L \in \{0,5,10,20,40,80\}$ steps
At 20 Hz: {0, 0.25, 0.5, 1, 2, 4 s}

For each LLL, measure:

- Success rate
- Grasp success
- Place success conditional on grasp

- Drop rate (object dropped)
- Timeout rate

Then report:

- **Critical duration** L50L_{50}L50: duration where success drops to 50% of clean
- **Robustness area** (AUC): average success across durations (one-number robustness score)

4.3 Timing/phase study (this is where the paper becomes "real")

A pure "duration-only" curve is usually noisy. Phase-based is much stronger.

Define phases using privileged signals (allowed, since it's evaluation tooling):

- **Phase A (Reach):** from episode start until EE within e.g. 6 cm of object
- **Phase B (Grasp):** first contact → grasp confirmed
- **Phase C (Transport):** grasp confirmed → object within 6 cm of goal
- **Phase D (Place/Release):** near goal → release and settle

Then trigger pose corruption at the start of each phase, with durations LLL.

This will let you claim something specific and non-obvious, e.g.:

- "Pose outages during Reach are catastrophic for all models."
- "Force sensing improves robustness primarily during Grasp/Transport/Place."
- "Robust training shifts the 'critical phase' boundary."

This phase × duration heatmap is an excellent main figure.

4.4 Add two realistic corruption modes (reviewers like this)

Instead of only "drop to zero," include:

- **Freeze-last** (stale pose)
- **Noisy pose** (jitter/drift)

These correspond to real system failures:

- tracking stuck
- misdetections / depth noise

Often the force sensor helps more in *freeze-last* than in *hard dropout*, which is an interesting nuance.

Step 5 — What you can claim about force sensing (without overclaiming)

Be very explicit about the causal mechanism:

What force sensing can help with

- Grasp confirmation (knowing you have the object)
- Slip/instability detection (maintain grip without precise pose updates)
- Contact-rich placement (knowing when object contacts the surface/bin)

What it cannot help with

- Global object localization before contact (if pose is missing early)

This makes your conclusions credible.

Step 6 — "Short conference paper" structure (optimized for your situation)

Title (lean into the pivot)

Option A: *"Perception Outage Robustness in RL Pick-and-Place: Diagnosing Modality Dominance and the Role of Force Feedback"*
Option B: *"When Privileged Inputs Dominate: Robust Manipulation Under Object-Pose Dropouts with Force Sensing"*

Abstract (what you say now)

- Problem: learned manipulation policies rely on perception outputs that can fail.
- Observation: policies trained with privileged object pose can ignore vision and collapse under pose outages.
- Method: controlled pose-dropout/noise/delay benchmark + force sensing + robustness fine-tuning.
- Results: force improves robustness after contact; dropout training improves across phases; identify critical outage timing/duration.

Sections (minimal but complete)

1. **Introduction**
   - Sensor/perception outages are real and under-studied in RL manipulation
2. **Setup**
   - IsaacLab task, policy inputs, training algorithm
   - Explicitly acknowledge privileged training as "perfect perception output"
3. **Robustness Benchmark**
   - Define pose corruption types, schedules, phase-based triggers
4. **Methods (M1–M4)**
   - M1/M2: clean training
   - M3/M4: robustness fine-tuning (pose corruption curriculum)
5. **Results**
   - Modality sensitivity diagnostic (vision vs pose vs force ablations)
   - Duration curves + AUC/L50
   - Phase × duration heatmaps
   - Failure mode breakdown
6. **Discussion & limitations**
   - Sim contact realism; no full end-to-end vision; implications for real perception pipelines

Must-have figures/tables

- **Table 1:** Observations per model (pose/proprio/force/vision)
- **Figure 1:** Modality ablation bar chart (drop pose vs drop vision vs drop force)
- **Figure 2:** Success vs dropout duration (curves for M1–M4)
- **Figure 3:** Phase × duration heatmap (per model or side-by-side)
- **Table 2:** AUC robustness score + L50L_{50}L50 per model + CI

Step 7 — How to write the "privileged training" part so it doesn't get rejected

Don't hide it. Frame it as a deliberate proxy:

- Call cube position input: **"object pose estimate (oracle)"**
- State clearly: "We model perception output directly rather than training end-to-end from pixels; this isolates robustness to estimation outages."

Then your novelty is the **outage benchmark + multi-modal redundancy**, not end-to-end vision.

If reviewers push: you can say your future work is distilling to vision-only policies (see below).

Optional upgrade (stronger paper, still uses your existing policies): teacher–student distillation

If you have bandwidth, this is the best way to recover the *original* "vision-based policy" claim without discarding your work:

- Use M1/M2 as **teachers** (they are strong because they see oracle pose).
- Collect rollouts: (ovision+proprio(+force),ateacher)(o_{vision+proprio(+force)}, a_{teacher})(ovision+proprio(+force),ateacher).
- Train a student policy on **vision+proprio** (and optionally force) via behavior cloning.
- Then optionally fine-tune student with RL (PPO) without oracle pose.

This turns privileged policies into a *feature*:

"Learning from privileged information improves sample efficiency and robustness."

But if you want the fastest path to publication, the **pose-outage robustness paper** above is already coherent.

Bottom line plan (no questions, actionable)

1. **Run the modality sensitivity diagnostic** now:
    - vision blackout vs pose dropout vs force dropout
    - this tells you whether your current M3/M4 are actually different
2. **Implement pose corruption wrapper** (hard, freeze, noise, delay).
3. **Evaluate M1–M4** on:
    - duration sweep
    - phase-based triggers
4. If M3/M4 are redundant, **fine-tune** from M1/M2 with the corruption curriculum to create meaningful robust versions.
5. Write the paper around:
    - **modality dominance diagnosis**
    - **perception outage benchmark**
    - **force as redundancy + phase/duration sensitivity**

This will read like a legitimate, self-contained short conference paper, and it does not require you to pretend you trained end-to-end from RGB-D.