InstructFlow: Adaptive Symbolic Constraint-Guided Code Generation for Long-Horizon Planning

Haotian Chi[1], Zeyu Feng[2], Yueming Lyu[2], Chengqi Zheng[2], Linbo Luo[3], Yew-Soon Ong[2,4], Ivor Tsang[2,4], Hechang Chen[1*], Yi Chang[1*], Haiyan Yin[2*]

[1]Jilin University, [2]A*STAR, Centre for Frontier AI Research, [3]Xidian University, [4]Nangyang Technological University

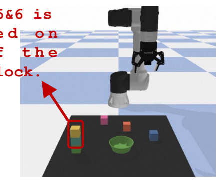
Paper Link Demo

cht2915192
chiht19@gmail.com

Motivations

- LLMs have become a prevalent approach for **robotic code generation**.
- However, LLMs frequently hallucinate valid-looking but **physically infeasible** code or **fail to recover** when execution errors occur.

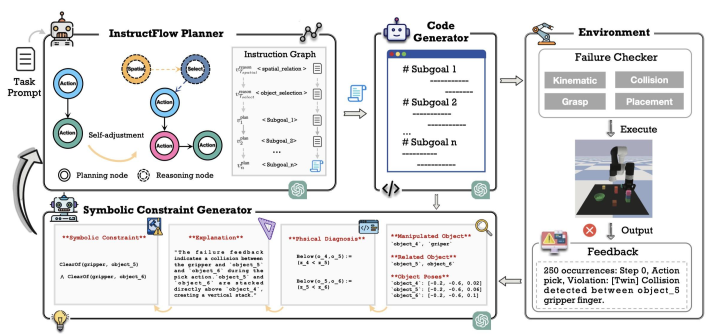
Unstack: Place a green block into a green bowl



[Error Message]: "250 occurences: Step 0, Action pick, Violation: [Twin] Collision detected between object_5 object gripper finger" **and** "250 occurences: Step 0, Action pick, Violation: [Twin] Collision detected between object_6 object gripper finger"

InstructFlow

✓ We propose **InstructFlow**, a **multi-agent** framework that establishes a **symbolic, feedback-driven flow** of information for code generation in robotic manipulation tasks.



InstructFlow-Guided Code Generation — InstructFlow Planner / Code Generator

Instruction Graph

$$\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t), \quad \mathcal{V}_t = \mathcal{V}_{\text{plan}} \cup \mathcal{V}_{\text{reason}}$$

$$v^{\text{plan}} : (\text{goal}, \text{state}) \to \text{subgoal}_{\mathcal{A}}, \quad \mathcal{A} \in \{\text{pick}, \text{place}, \dots\}.$$

$\mathcal{T}_{\text{spatial}} : S \to \texttt{Rel}(\text{Objects}, \text{Adjacency})$ (spatial relation inference)

$\mathcal{T}_{\text{density}} : S \to \texttt{Rel}(\text{Objects}, \text{Density})$ (local clutter estimation)

$\mathcal{T}_{\text{select}} : (G, S, \Phi) \to \texttt{Select}(\text{Objects})$ (target object selection)

$\mathcal{T}_{\text{order}} : (G, S, \Phi) \to \texttt{Order}(\text{Actions})$ (plan logic inference)

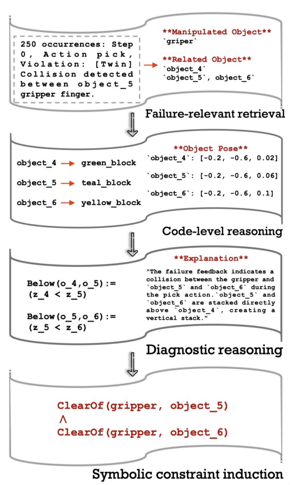
$\mathcal{T}_{\text{param}} : (G, \Phi) \to \texttt{Refine}(\text{ParamDomain})$ (parameter range refinement)

$$\underbrace{\texttt{instr}^{(t)}}_{\text{Task Prompt}} = \text{Encode}\left(\{v^{\text{reason}}_{\mathcal{T}_j}\}_{j=1}^{|v^{\text{reason}(t)}|},\ v^{\text{plan}}\right), \quad \underbrace{\texttt{code}^{(t)}}_{\text{Generated Code}} = \text{LLM}\left(\texttt{instr}^{(t)}\right)$$

Symbolic Constraint Induction from Failures — Symbolic Constraint Generator

- We formalize the symbolic constraint φ as a conjunction over two complementary modalities of failure correction.

$$\phi := \bigwedge_{c \in \mathcal{C}(\mathcal{E},\mathcal{R},\mathcal{F},\mathcal{B})} c, \quad \text{where} \quad \mathcal{C}(\mathcal{E},\mathcal{R},\mathcal{F},\mathcal{B}) = \underbrace{\{R_i(e_{a_i}, e_{b_i})\}}_{\text{Relational Constraints}} \cup \underbrace{\{f_j(\Theta_j) \oplus \tau_j\}}_{\text{Physical Constraints}}.$$



Failure Diagnosis Workflow

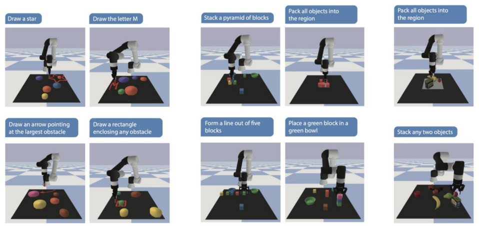
Task	Symbolic Constraints
Pyramid	StableOn(?block_top, ?block_bottom); ProximitySafe(?block_bottom); PathClear(?gripper, ?block); Before(place(block_top), place(?block_bottom)); Distance(?block_bottom) ∈[0, 0.04] Aligned(?block_bottom) ∧ On(block_top, ?block_bottom); Order(Place(block_bottom1), Place(block_bottom2), Place(block_top)); Contact(block_top, ?block_bottom) ∧ CloseTogether(?block_bottom); CenterOfMass(block$_t$op) ∈ SupportArea(?block_bottom)
Line	StableOn(?block, ?table); Alignment(?block); Alignment_tolerance ∈[-0.01, 0.01]
Packing (Blocks)	ProximitySafe(?block, ?boundary); PlacementFeasible(?block, square_region); WithinBounds(?block, region_center, 2*blcok_size); ProximitySafe(block, region_center) ∧ WithinDistance(?block, region_center, ?max_distance)
Unstack (Blocks)	StableOn(?block, ?bowl); ProximitySafe(?gripper, ?block); Cleafof(?gripper,?obstacle); PlacementFeasible(green_block, green_bowl); Inside(green_block, green_bowl); Aligned(block_center, bowl_center); NotStacked(green_block, ?obstacle) ∧ offset_z > 0.03
Packing (YCB)	ProximitySafe(?gripper, ?object); GraspFeasible(?grasp, ?object_pose); ProximitySafe(object, table_center); Distance(?object, table_center) < 0.06; PlacementFeasible(?object, center, threshold) ∧ threshold = 0.06; Graspable(?grasp, ?object); CollisionFree(?gripper, ?object)
Stacking (YCB)	OnTop(object_a, object_b); ClearOf(object_a, surface); ClearSurface(object_b); PlacementFeasible(object_a, object_b); StableOn(object_a, object_b); AlignedForStacking(object_a, object_b); Graspable(?object, ?grasp) ∧ CollisionFree(?object)

Table 4: Inducted constraints for each task across Arrange-Blocks and Arrange-YCB domains

$$\phi_{\text{pick}} := \texttt{ProximitySafe}(?object, ?neighbor) \wedge \texttt{PathClear}(?gripper, ?object),$$
$$\phi_{\text{place}} := \texttt{Dist}(?pose, ?neighbor) \geq \delta_{\text{safe}} \wedge \texttt{StableOn}(?object, ?surface).$$

Main Results

- Empirical results validate the InstructFlow's ability to handle **long-horizon, constraint-sensitive** scenarios with **improved success rates** and **sample efficiency**.



	Drawing				Arrange Blocks				Arrange YCB	
	Star	Arrow	Letters	Enclosed	Pyramid	Line	Packing	Unstack	Packing	Stacking
LLM[3]	40%	40%	80%	50%	0%	40%	30%	0%	0%	10%
CaP	10%	0%	40%	30%	20%	20%	20%	10%	30%	10%
PRoC3S	90%	**80%**	80%	90%	60%	70%	50%	60%	30%	40%
InstructFlow (Ours)	**100%**	**80%**	**100%**	**100%**	**90%**	**100%**	**90%**	**90%**	**60%**	**70%**

Table 1: Task success rates (%) across drawing, block arrangement, and YCB manipulation domains. Bold indicates top-performing results.

	Drawing				Arrange Blocks				Arrange YCB	
	Star	Arrow	Letters	Enclosed	Pyramid	Line	Packing	Unstack	Packing	Stacking
Ours	**100%**	**80%**	**100%**	**100%**	**90%**	**100%**	**90%**	**90%**	**60%**	**70%**
Ours w/o Planner Agent	90%	80%	80%	100%	50%	90%	50%	40%	40%	40%
Ours w/o Constraint Agent	100%	80%	100%	80%	40%	100%	60%	60%	30%	40%

Table 2: Ablation study results (% task success) highlighting the contributions of the InstructFlow Planner and Symbolic Constraint Generator.

Case Study

- A code snippet illustrating how InstructFlow repairs the Unstack plan by intuitively injecting a targeted object removal routine automatically derived from InstructFlow's structural reasoning.

```
def gen_plan(initial: RavenBelief,
             x_clear1: float, y_clear1: float,
+            x_clear2: float, y_clear2: float,
             x_offset: float, y_offset: float):
    plan = []

    # subgoal 1: Pick object_6 (the yellow block)
    block6 = initial.objects["object_6"]
    plan.append(Action("pick", block6.pose.point))

    # subgoal 2: Place object_6 at a clear location
    plan.append(Action("place", [x_clear1, y_clear1, block6.pose.point[2]]))
+
    # subgoal 3: Pick object_5 (the teal block)
    block5 = initial.objects["object_5"]
    plan.append(Action("pick", block5.pose.point))

    # subgoal 4: Place object_5 at a clear location
    plan.append(Action("place", [x_clear2, y_clear2, block5.pose.point[2]]))

    # subgoal 5: Pick object_4 (the green block)
    block4 = initial.objects["object_4"]
    plan.append(Action("pick", block4.pose.point))

    # subgoal 6: Place object_4 into object_7 (the green bowl)
    bowl7 = initial.objects["object_7"]
    x, y, z = bowl7.pose.point
    plan.append(Action("place", [x + x_offset, y + y_offset, z]))

    return plan
```

- InstructFlow can repair code respects both **spatial constraints** and **temporal dependencies.**

Conclusion

InstructFlow supports **targeted code repair**, avoids full-plan regeneration, and significantly **enhances robustness** in manipulation tasks.