Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Neighborhood-Aware Fuzzy Rough Action Selection for Vision-Language-Action Manipulation

Mengqi Mao
Yangzhou University
Xiangjian Chen
007598@yzu.edu.cn
Yangzhou University
Shixin Huang
Jiangsu University of Science and Technology

Article

Keywords:

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11318516/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Neighborhood-Aware Fuzzy Rough Action Selection for Vision-Language-Action Manipulation

Mengqi Mao[1], **Xiangjian Chen**[1,*], **and Shixin Huang**[2]

[1]College of Information and Artificial Intelligence (College of Industrial Software), Yangzhou University, Yangzhou 225002, China
[2]Jiangsu University of Science and Technology, Zhenjiang 212003, China
[*]007598@yzu.edu.cn

ABSTRACT

Vision-Language-Action (VLA) models map visual observations and language instructions to robot actions, but the highest-scoring action is not always the most reliable one for physical execution. A candidate may satisfy the language instruction and remain collision-free at its nominal pose, while nearby actions exhibit less favorable physical properties. We present FRS-Grounding, a fuzzy rough set-based action-selection framework that evaluates each VLA candidate together with its local execution structure. Given a finite candidate set, FRS-Grounding combines a normalized VLA-conditioned candidate score and a physical-distance value to form a semantic–physical fuzzy action set. Pairwise translational and rotational differences, together with collision-free states, define a physics-informed fuzzy action relation. Fuzzy rough lower and upper approximations are then used to characterize local support within the candidate set. Candidates with stronger neighborhood-consistent support are distinguished from those lying in the boundary region, and these regions are mapped to normal execution, compliant execution, or rejection. Across 1500 randomized simulation trials, FRS-Grounding achieves a success rate of 86.4% and a safety-violation rate of 2.1%, compared with 74.8% and 8.2% for VLA-HC. Its semantic compliance is 0.89, close to RT-2 (0.91) and higher than VoxPoser (0.85) and VLA-HC (0.88). Under positional noise with $\sigma = 10$ mm, FRS-Grounding retains a success rate of approximately 78%, compared with approximately 28% for VLA-HC. Real-robot experiments further show how the boundary branch can be coupled with force-compliant control. The results support candidate-set evaluation as a practical way to combine VLA preferences with local physical structure in the tested manipulation tasks.

Keywords: Vision-Language-Action Model; Embodied Intelligence; Fuzzy Rough Sets; Action Selection; Semantic–Physical Alignment; Robot Manipulation

1 Introduction

Vision-Language-Action (VLA) models provide a direct interface between visual observations, natural-language instructions, and robot actions, and have become an increasingly important approach to instruction-conditioned robotic manipulation [1–3, 5]. Recent systems have extended this paradigm through embodied multimodal reasoning, cross-robot training data, and generalist robot policies [4, 6, 7]. These developments improve the ability of a robot to interpret task-level instructions and generate semantically appropriate actions. However, a strong semantic preference does not necessarily imply that the corresponding action is reliable for physical execution. Safety and robustness remain important concerns, particularly when perception, calibration, or execution is affected by perturbations [8, 9].

This issue is especially relevant near physical boundaries. A candidate action may receive a high VLA-conditioned candidate score and remain collision-free at its nominal pose, yet small changes in object localization or end-effector execution can move the realized action toward a region with different feasibility properties. In such cases, evaluating only the selected candidate does not describe the local execution structure surrounding it. The difficulty is therefore not only to determine whether an individual action is feasible, but also to determine whether its nearby alternatives exhibit compatible semantic and physical properties.

Existing robotic decision methods address different parts of this problem. Language-conditioned manipulation has been studied through affordance grounding, programmatic policy generation, multimodal prompting, and spatial value-map construction [10–13]. Other visuomotor policies use foundation-model adaptation, large-scale video pre-training, diffusion-based action generation, or action-sequence modeling [14–17]. These approaches provide effective mechanisms for action generation, but they do not explicitly evaluate a VLA candidate together with the consistency of its local execution neighborhood.

Physical feasibility is commonly handled by a separate class of methods. Obstacle-avoidance techniques, trajectory

optimization, and explicit safety constraints can prevent or penalize physically undesirable actions [18–21]. Pointwise post-processing filters can likewise remove candidates that violate collision or workspace constraints. Such mechanisms are effective for enforcing feasibility, but the final decision is typically based on the state or cost of individual candidates. Value-map-based methods such as VoxPoser incorporate spatial objectives into robot planning [13], yet their formulation is different from evaluating whether a candidate remains supported by physically related neighboring actions. This leaves a gap between semantic action preference and neighborhood-level physical consistency.

Fuzzy rough sets provide a useful mathematical basis for representing this type of local uncertainty. Fuzzy-set theory represents graded membership [22], while rough-set theory characterizes objects through lower and upper approximations [23]. Their combination extends approximation reasoning to graded relational settings [24–27]. Fuzzy-rough models have subsequently been applied to dimensionality reduction, attribute selection, classification, and related uncertainty-analysis problems [28–30]. However, these formulations do not directly define how VLA action candidates, robot-specific execution relations, and control modes should be connected within a manipulation pipeline.

To address this gap, we propose FRS-Grounding, a fuzzy rough action-selection framework that retains the candidate set produced by VLA inference and evaluates each candidate together with its local execution neighborhood. The framework does not modify the parameters of the underlying VLA model. Instead, normalized VLA-conditioned candidate scores and candidate-level physical-distance information are combined into a semantic–physical fuzzy action set, while pairwise translational and rotational differences together with collision-free states define a fuzzy action relation.

The resulting approximation regions are used directly in the final execution decision. Candidates with sufficiently strong lower-approximation support are considered for normal execution. If no such candidate is available, candidates in the boundary region that retain sufficient upper-approximation support may enter a compliant-control branch. When neither condition is satisfied, the current execution attempt is rejected. The proposed framework therefore uses fuzzy rough approximation as an action-selection and execution-routing mechanism rather than introducing a new fuzzy rough operator.

Its difference from a simple neighborhood-consistency filter lies in how local support is represented and used. A simple filter may determine neighboring candidates using a fixed threshold and then accept, suppress, or re-rank a candidate according to an aggregated local criterion. FRS-Grounding assigns graded relation values to candidate pairs according to pose proximity and collision-free feasibility. The lower approximation measures whether strongly related candidates provide consistent support, while the upper approximation retains candidates with broader local support. Their difference forms the boundary region used in the subsequent decision procedure. When no candidate satisfies the lower-approximation criterion, a boundary candidate with sufficient upper support may be routed to compliant execution. The fuzzy rough regions are therefore used not only to filter candidates, but also to organize candidate support and determine the subsequent execution mode.

The main contributions of this work are as follows:

- **Candidate-set formulation of VLA action selection.** Instead of selecting an action solely from pointwise VLA-conditioned candidate scores, the proposed framework retains the finite candidate set until the final decision stage so that semantic preference, candidate-level physical information, and local action structure can be considered jointly.
- **Semantic–physical candidate representation and pairwise action relation.** Normalized VLA-conditioned candidate scores and physical-distance values are combined in a candidate-level fuzzy action set, while pairwise pose proximity and collision-free feasibility define the local relation used for subsequent approximation.
- **Neighborhood-consistent candidate evaluation.** Candidate support is evaluated through fuzzy rough lower and upper approximations rather than by independent candidate ranking. For collision-free candidates, the resulting memberships satisfy

  $$\mu_{\underline{R}A}(a_i) \le \mu_A(a_i) \le \mu_{\overline{R}A}(a_i),$$

  which provides the ordering used to distinguish strongly supported candidates from boundary cases.
- **Region-based hierarchical execution.** The approximation results are mapped to three execution outcomes: normal execution, compliant execution, and rejection. In the real-robot implementation, candidates selected through the boundary branch are passed to the force-compliant controller.

2 Problem Formulation and Action-Space Modeling

We consider the action-selection stage of a VLA-based robotic manipulation system. Given a visual observation I and a language instruction L, the upstream VLA policy first produces an instruction-conditioned nominal action. A finite local candidate set is then constructed around this nominal action, and each candidate is assigned a normalized VLA-conditioned preference score. Physical execution additionally depends on the state of each candidate and on its relationship with nearby

actions. The goal is therefore to select the final robot action after evaluating the local candidate set together with the available physical information.

2.1 Finite Candidate Action Space

Let a candidate end-effector action be represented by

$$a_i = (\mathbf{p}_i, \mathbf{q}_i), \tag{1}$$

where $\mathbf{p}_i \in \mathbb{R}^3$ is the translational component and $\mathbf{q}_i \in \mathbb{R}^4$ is a unit quaternion describing orientation.

The action space is discretized or sampled to form the finite candidate set

$$U = \{a_1, a_2, \ldots, a_N\}. \tag{2}$$

The formulation does not require a fixed candidate count. The value of N is determined by the candidate-generation stage for the current observation–instruction pair, and FRS-Grounding evaluates the complete candidate set supplied to the action-selection module.

For a given observation–instruction pair (I, L), the upstream VLA policy first produces an instruction-conditioned nominal action. Each local candidate $a_i \in U$ is subsequently assigned a normalized VLA-conditioned candidate score

$$s_i = s_{\text{cond}}(a_i \mid I, L), \qquad s_i \in [0,1], \qquad a_i \in U, \tag{3}$$

where s_i measures the alignment of candidate a_i with the instruction-conditioned nominal action produced by the upstream VLA policy. The specific candidate-scoring procedure is given in Section 4.1.3.

A pointwise candidate-preference decision can therefore be written as

$$a^*_{\text{pref}} = \arg\max_{a_i \in U} s_i. \tag{4}$$

This rule uses the VLA-conditioned preference of each candidate independently and therefore does not account for the physical structure surrounding the selected action.

2.2 Pointwise Feasibility and Local Execution Uncertainty

Physical feasibility is evaluated from the robot configuration and workspace constraints. Let

$$C(a_i) \in \{0,1\} \tag{5}$$

where $C(a_i) = 1$ denotes a collision-free candidate and $C(a_i) = 0$ otherwise. A hard-constraint filter can use this indicator to remove infeasible candidates, but the resulting decision remains pointwise.

Consider a high-scoring action that is collision-free at its nominal pose. Localization or execution errors may shift the realized pose toward a nearby candidate with a different collision state. The feasibility of the nominal action alone therefore does not describe the variation in its local execution region.

For this reason, neighboring candidates are retained during action evaluation. The final decision depends not only on the score and feasibility of an individual candidate, but also on the support available from physically related actions.

2.3 Neighborhood-Aware Action-Selection Problem

The local execution structure is represented by a graded pairwise relation rather than a binary neighborhood. For two candidates a_i and a_j, the relation depends on their pose difference and physical feasibility. Candidates with different translational and rotational deviations can therefore contribute to the local evaluation to different degrees.

Along with the normalized VLA-conditioned candidate score s_i, each candidate is associated with a non-negative physical-distance value

$$d_i \geq 0, \tag{6}$$

as well as its pose $(\mathbf{p}_i, \mathbf{q}_i)$ and collision-free state $C(a_i)$. The physical-distance value is supplied by the physical evaluation component of the action-generation pipeline and is treated as an input to FRS-Grounding.

The resulting action-selection problem is

$$\{s_i,\ d_i,\ (\mathbf{p}_i, \mathbf{q}_i),\ C(a_i)\}_{i=1}^{N} \longrightarrow \text{candidate-set evaluation} \longrightarrow (a^*, m^*), \tag{7}$$

where a^* is the selected action and m^* is the corresponding execution mode. The final decision is therefore made from the candidate set and its local physical structure rather than from an isolated action score. FRS-Grounding implements this formulation through a semantic–physical fuzzy action set, a physics-informed fuzzy action relation, fuzzy rough approximations, and region-based hierarchical selection.

3 Method

3.1 Framework Overview

FRS-Grounding operates on the local candidate set generated around the instruction-conditioned nominal action produced by VLA inference. Given an observation–instruction pair (I, L), the candidate-scoring interface produces normalized VLA-conditioned scores $\{s_i\}_{i=1}^{N}$ over the finite action universe U.

Candidate information is represented at two levels. At the individual-action level, the VLA-conditioned candidate score and physical-distance value define a semantic–physical fuzzy action set A. At the pairwise level, pose differences and collision-free states define a physics-informed fuzzy action relation R. Together, (A, R) provide the approximation space used for fuzzy rough evaluation.

The lower approximation captures neighborhood-consistent support, while the upper approximation retains candidates with remaining local support. Their difference defines the boundary membership. These quantities are subsequently used for action selection and execution-mode assignment. The overall processing chain is

$$\left.\begin{aligned} \{s_i, d_i\}_{i=1}^{N} &\longrightarrow A, \\ \{(\mathbf{p}_i, \mathbf{q}_i), C(a_i)\}_{i=1}^{N} &\longrightarrow R \end{aligned}\right\} \longrightarrow \left(\underline{R}A, \overline{R}A, \mathrm{BND}\right) \longrightarrow (a^*, m^*). \tag{8}$$

where a^* denotes the selected action and m^* the execution mode.

3.2 Semantic–Physical Representation of Candidate Actions

For each candidate $a_i \in U$, let $s_i \in [0,1]$ denote its normalized VLA-conditioned candidate score. Each candidate is also associated with a non-negative physical-distance value

$$d_i \geq 0, \tag{9}$$

Let δ_i denote the minimum signed clearance between the robot geometry at candidate pose a_i and all non-target obstacles in the workspace. Positive values indicate geometric separation, zero indicates contact, and negative values indicate penetration.

The reference safe-action set is defined as

$$\mathscr{A}_{\mathrm{safe}} = \{a_i \in U \mid C(a_i) = 1,\ \delta_i \geq \delta_{\mathrm{safe}}\}, \tag{10}$$

where $\delta_{\mathrm{safe}} > 0$ is a clearance margin used by the upstream physical evaluator. In the implementation, the clearance test and the binary collision test are evaluated from the same workspace geometry, while FRS-Grounding receives only the resulting safe-set membership and collision-free indicator.

For a candidate $a_i = (\mathbf{p}_i, \mathbf{q}_i)$, the physical-distance value is then defined as the minimum normalized pose distance to the reference safe-action set,

$$d_i = \min_{a_j \in \mathscr{A}_{\mathrm{safe}}} \sqrt{\frac{\|\mathbf{p}_i - \mathbf{p}_j\|_2^2}{\sigma_t^2} + \frac{d_r(\mathbf{q}_i, \mathbf{q}_j)^2}{\sigma_r^2}}. \tag{11}$$

The translational deviation is normalized by σ_t, while the rotational deviation is normalized by σ_r. Consequently, d_i is dimensionless. Candidates belonging to $\mathscr{A}_{\mathrm{safe}}$ satisfy $d_i = 0$, whereas candidates outside this reference set obtain progressively larger values according to their normalized pose displacement from the nearest safe-reference action.

The distinction between $C(a_i)$ and $\mathscr{A}_{\text{safe}}$ is important. The condition $C(a_i) = 1$ only indicates that the nominal candidate pose is collision-free, whereas membership in $\mathscr{A}_{\text{safe}}$ additionally requires the clearance margin δ_{safe}. Therefore, a collision-free candidate may still have $d_i > 0$ when it lies outside the reference safe-action region.

The safe-action set itself is constructed by the upstream physical evaluation module and is not reconstructed within FRS-Grounding. If no candidate satisfies the reference safe-action conditions so that $\mathscr{A}_{\text{safe}} = \varnothing$, the current decision is terminated with

$$a^* = \varnothing, \qquad m^* = \text{reject}, \tag{12}$$

rather than attempting to evaluate a distance to an empty reference set.

The corresponding physical-support term is

$$\rho_i = \exp(-\beta d_i), \qquad \beta > 0, \tag{13}$$

which maps $d_i \geq 0$ to $\rho_i \in (0, 1]$. The physical-support value decreases monotonically as the distance from the safe action set increases. No additional min–max normalization of d_i is applied within FRS-Grounding.

The semantic–physical membership is then defined as

$$\mu_A(a_i) = \alpha s_i + (1 - \alpha)\rho_i, \qquad \alpha \in [0, 1]. \tag{14}$$

The semantic–physical fuzzy action set is

$$A = \{(a_i, \mu_A(a_i)) \mid a_i \in U\}. \tag{15}$$

The parameter α controls the relative contribution of the normalized VLA-conditioned candidate score and the physical-support term.

3.3 Physics-Informed Fuzzy Action Relation

The fuzzy action set A represents individual candidates but does not encode relationships among them. We therefore define a pairwise fuzzy action relation R over $U \times U$.

For

$$a_i = (\mathbf{p}_i, \mathbf{q}_i), \qquad a_j = (\mathbf{p}_j, \mathbf{q}_j), \tag{16}$$

the pose relation is defined as

$$K_{\text{pose}}(a_i, a_j) = \exp\left[-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|_2^2}{2\sigma_t^2} - \frac{d_r(\mathbf{q}_i, \mathbf{q}_j)^2}{2\sigma_r^2}\right], \tag{17}$$

where $\sigma_t > 0$ and $\sigma_r > 0$ determine the translational and rotational scales of the pairwise relation.

The collision term in the pairwise relation is used to restrict relational support to collision-free candidate pairs. If either candidate violates the collision-free condition, then

$$\mu_R(a_i, a_j) = 0. \tag{18}$$

This zero relation value should not be interpreted as an explicit collision penalty. In the lower approximation,

$$\max(1 - \mu_R(a_i, a_j), \mu_A(a_j)) = 1 \qquad \text{when } \mu_R(a_i, a_j) = 0, \tag{19}$$

so a collision-inducing candidate does not reduce the lower-approximation membership of another candidate. Similarly, its contribution to the upper approximation is zero. Collision-inducing candidates are therefore excluded from positive relational support rather than treated as negative relational evidence.

Physical exclusion is enforced separately through the explicit eligibility condition $C(a_i) = 1$ in Eqs. (35) and (38). The fuzzy relation and the eligibility condition thus have complementary roles: R describes graded support among collision-free candidate pairs, whereas $C(a_i)$ prevents a collision-inducing candidate from being selected for execution.

For two unit quaternions, the rotational distance is

$$d_r(\mathbf{q}_i, \mathbf{q}_j) = 2 \arccos\left(\left|\mathbf{q}_i^\top \mathbf{q}_j\right|\right). \tag{20}$$

The absolute inner product accounts for the fact that $\mathbf{q}$ and $-\mathbf{q}$ represent the same physical orientation.

Collision-free feasibility is introduced through

$$c_i = C(a_i) \in \{0, 1\}, \tag{21}$$

and the pairwise collision term is

$$K_{\text{col}}(a_i, a_j) = c_i c_j. \tag{22}$$

The physics-informed fuzzy action relation is defined as

$$\mu_R(a_i, a_j) = K_{\text{pose}}(a_i, a_j) K_{\text{col}}(a_i, a_j). \tag{23}$$

When both candidates are collision-free, their relation is determined by their translational and rotational proximity. If either candidate violates the collision-free condition, the pairwise relation is zero. The physical-distance term in A and the collision term in R have different roles. The former contributes to the membership of an individual candidate, while the latter determines whether a candidate pair participates in the collision-free relation. Under this definition, pairs involving a collision-inducing candidate do not provide relational support in the subsequent fuzzy rough approximation. The relation R therefore represents graded local support among collision-free candidate pairs.

Property 1: Symmetry and self-relation. The relation R is symmetric because both pose distances are symmetric and $c_i c_j = c_j c_i$. Hence,

$$\mu_R(a_i, a_j) = \mu_R(a_j, a_i). \tag{24}$$

For the self-relation, the translational and rotational differences vanish, so

$$K_{\text{pose}}(a_i, a_i) = 1. \tag{25}$$

Since $C(a_i) \in \{0, 1\}$,

$$K_{\text{col}}(a_i, a_i) = C(a_i)^2 = C(a_i), \tag{26}$$

and therefore

$$\mu_R(a_i, a_i) = C(a_i). \tag{27}$$

3.4 Fuzzy Rough Approximation of Candidate Actions

3.4.1 Lower Approximation

For each $a_i \in U$, the lower-approximation membership is

$$\mu_{\underline{R}A}(a_i) = \min_{a_j \in U} \max\left(1 - \mu_R(a_i, a_j), \mu_A(a_j)\right). \tag{28}$$

For a weak relation value $\mu_R(a_i, a_j)$, the term $1 - \mu_R(a_i, a_j)$ becomes large, so candidate a_j places little constraint on the minimum in Eq. (28). As the relation becomes stronger, the membership $\mu_A(a_j)$ has a greater effect on the lower-approximation value of a_i. A candidate with high individual membership can therefore obtain a low lower-approximation value when strongly related candidates have low membership. In the present formulation, the lower approximation measures consistency with respect to the support relation defined by R.

3.4.2 Upper Approximation

The upper-approximation membership is

$$\mu_{\overline{RA}}(a_i) = \max_{a_j \in U} \min\left(\mu_R(a_i, a_j), \mu_A(a_j)\right). \tag{29}$$

For collision-free candidates that remain eligible for final selection, the ordering derived below places the upper approximation above the corresponding lower approximation. Within the relation defined by R, the upper approximation therefore provides a less restrictive measure of candidate support.

3.4.3 Approximation Ordering for Eligible Candidates

For every collision-free candidate a_i satisfying $C(a_i) = 1$, Property 1 gives

$$\mu_R(a_i, a_i) = 1. \tag{30}$$

Using $a_j = a_i$ in Eq. (28) gives

$$\mu_{\underline{RA}}(a_i) \leq \mu_A(a_i), \tag{31}$$

while using $a_j = a_i$ in Eq. (29) gives

$$\mu_{\overline{RA}}(a_i) \geq \mu_A(a_i). \tag{32}$$

Therefore, for every candidate eligible for final collision-free selection,

$$\mu_{\underline{RA}}(a_i) \leq \mu_A(a_i) \leq \mu_{\overline{RA}}(a_i). \tag{33}$$

Equation (33) gives the ordering used to interpret the three membership quantities at the final decision stage.

3.4.4 Boundary Region

For the complete candidate universe, boundary membership is defined as

$$\mu_{\mathrm{BND}}(a_i) = \max\left[0, \mu_{\overline{RA}}(a_i) - \mu_{\underline{RA}}(a_i)\right]. \tag{34}$$

For collision-free candidates, Eq. (33) already guarantees a non-negative difference; the outer maximum with zero is retained for numerical robustness over the full candidate universe.

A small boundary membership indicates that the lower and upper support values are close, while a larger value reflects a wider gap between them. The boundary membership is used as an operational quantity in the decision procedure and is not interpreted as a probability of failure or physical risk.

3.5 Region-Based Hierarchical Action Selection

The approximation results are used in the final action-selection stage, while collision-free feasibility remains an explicit eligibility condition. The decision procedure first examines candidates with sufficient lower-approximation support and considers the boundary branch only when no such candidate is available.

The lower-supported candidate set is

$$U_{\mathrm{pos}} = \left\{a_i \in U \mid C(a_i) = 1,\ \mu_{\underline{RA}}(a_i) > \tau_{\mathrm{pos}}\right\}, \tag{35}$$

where $\tau_{\mathrm{pos}} \in [0, 1]$ is the lower-approximation threshold.
If $U_{\mathrm{pos}} \neq \varnothing$, the final action is selected as

$$a^* = \arg\max_{a_i \in U_{\mathrm{pos}}} \mu_A(a_i), \tag{36}$$

and the execution mode is

$$m^* = \text{normal}. \tag{37}$$

The normal branch therefore requires both collision-free feasibility and sufficient support under the lower approximation. The final ranking within U_{pos} is determined by the semantic–physical membership μ_A.

If U_{pos} is empty, the boundary branch is evaluated:

$$U_{\text{bnd}} = \left\{ a_i \in U \mid C(a_i) = 1,\ \mu_{\text{BND}}(a_i) > \tau_{\text{bnd}},\ \mu_{\overline{RA}}(a_i) > \tau_{\text{upper}} \right\}, \tag{38}$$

where $\tau_{\text{bnd}} \in [0,1]$ is the boundary threshold and $\tau_{\text{upper}} \in [0,1]$ is the minimum upper-approximation support required for retaining a boundary candidate.

If $U_{\text{bnd}} \neq \varnothing$, the final action is

$$a^* = \arg\max_{a_i \in U_{\text{bnd}}} \mu_A(a_i), \tag{39}$$

with

$$m^* = \text{compliant}. \tag{40}$$

The boundary branch represents a different decision state from the lower-supported branch: the candidate does not satisfy the lower-approximation criterion but retains sufficient support under the upper approximation. In the real-robot implementation, candidates selected from this branch are passed to the force-compliant controller. The compliant controller is a downstream execution mechanism and is not part of the fuzzy rough approximation itself.

If neither set contains an eligible candidate, the framework returns

$$a^* = \varnothing, \qquad m^* = \text{reject}. \tag{41}$$

This indicates only that no candidate satisfies the current selection criteria; it does not imply that the manipulation task is physically infeasible under all possible action representations or environmental conditions.

3.6 Integration with the VLA Inference and Robot-Control Pipeline

FRS-Grounding is placed after VLA-conditioned local candidate scoring and before final robot execution. The framework is implemented as a post-inference action-selection module and does not require modification of the parameters of the underlying VLA policy.

The candidate-scoring interface uses the instruction-conditioned nominal VLA action to provide candidate preference scores for the finite local action set. FRS-Grounding operates only on these candidate-level outputs and the associated physical information described in Sections 2 and 3.

The VLA output is therefore used to derive candidate preferences rather than being treated directly as the final action decision. The resulting VLA-conditioned candidate scores and physical-distance values are used to construct the fuzzy action set A, while candidate poses and collision-free states define the pairwise relation R. The lower and upper approximations are then evaluated over the retained candidate set before an action is sent to the controller. The resulting decision differs from pointwise post-processing because candidates are not evaluated only through their individual scores or feasibility states. Their support is evaluated with respect to the pairwise relation over the candidate set. The approximation results are then used both to restrict the eligible actions and to determine whether the selected action is executed in normal mode, passed to compliant control, or rejected.

3.7 Overall Inference Procedure

Algorithm 1 summarizes the complete inference procedure. The normalized VLA-conditioned candidate score and physical-distance value are first combined to obtain $\mu_A(a_i)$, after which pairwise pose and collision information are used to construct $\mu_R(a_i, a_j)$. The lower approximation, upper approximation, and boundary membership are then evaluated for all candidates.

The algorithm first examines U_{pos}. If this set is non-empty, the candidate with the largest semantic–physical membership is selected for normal execution. Otherwise, U_{bnd} is considered for compliant execution. If neither branch yields an eligible candidate, the current execution attempt is rejected.

Algorithm 1 FRS-Grounding Action Selection

Require: Candidate action set U; reference safe-action set $\mathscr{A}_{\text{safe}}$; normalized VLA-conditioned candidate scores $\{s_i\}_{i=1}^{N}$; physical-distance values $\{d_i\}_{i=1}^{N}$; candidate positions $\{\mathbf{p}_i\}_{i=1}^{N}$; candidate orientations $\{\mathbf{q}_i\}_{i=1}^{N}$; collision-free indicators $\{c_i\}_{i=1}^{N}$; parameters α, β, σ_t, σ_r, τ_{pos}, τ_{bnd}, and τ_{upper}.
Ensure: Selected action a^* and execution mode m^*.

if $\mathscr{A}_{\text{safe}} = \varnothing$ **then**
  $a^* \leftarrow \varnothing$.
  $m^* \leftarrow$ reject.
  return (a^*, m^*).
end if
for $i = 1$ to N **do**
  $\rho_i \leftarrow \exp(-\beta d_i)$.
  $\mu_A(a_i) \leftarrow \alpha s_i + (1-\alpha)\rho_i$.
end for
for each pair $(a_i, a_j) \in U \times U$ **do**
  Compute $K_{\text{pose}}(a_i, a_j)$.
  $K_{\text{col}}(a_i, a_j) \leftarrow c_i c_j$.
  $\mu_R(a_i, a_j) \leftarrow K_{\text{pose}}(a_i, a_j) K_{\text{col}}(a_i, a_j)$.
end for
for $i = 1$ to N **do**
  Compute $\mu_{\underline{RA}}(a_i)$ using Eq. (28).
  Compute $\mu_{\overline{RA}}(a_i)$ using Eq. (29).
  Compute $\mu_{\text{BND}}(a_i)$ using Eq. (34).
end for
Construct U_{pos} using Eq. (35).
if $U_{\text{pos}} \neq \varnothing$ **then**
  $a^* \leftarrow \arg\max_{a_i \in U_{\text{pos}}} \mu_A(a_i)$.
  $m^* \leftarrow$ normal.
  return (a^*, m^*).
end if
Construct U_{bnd} using Eq. (38).
if $U_{\text{bnd}} \neq \varnothing$ **then**
  $a^* \leftarrow \arg\max_{a_i \in U_{\text{bnd}}} \mu_A(a_i)$.
  $m^* \leftarrow$ compliant.
  return (a^*, m^*).
end if
$a^* \leftarrow \varnothing$.
$m^* \leftarrow$ reject.
return (a^*, m^*).

4 Experimental Design and Analysis

The experiments examine the behavior of FRS-Grounding in simulation and on a real robot. The evaluation focuses on three aspects: comparative task performance, sensitivity to positional perturbations, and the operational role of the lower-approximation and boundary branches in robot execution.

4.1 Experimental Setup

4.1.1 Simulation and Physical Platforms

The simulation experiments were implemented in PyBullet using a 7-DoF Franka Emika Panda manipulator and a depth-camera observation model. Each tabletop scene contained between 3 and 10 geometric objects. Object positions and orientations were randomized within the valid workspace before each trial so that different spatial configurations and candidate-action

neighborhoods were generated across trials. Configurations that placed an object outside the reachable workspace were excluded before action evaluation.

Each trial was defined by one randomized scene, one language instruction, and the corresponding finite candidate-action set. The scene state and instruction were fixed during the evaluation of a trial so that the compared decision methods operated on the same manipulation instance. For each candidate a_i, the evaluation pipeline produced the normalized VLA-conditioned candidate score s_i, the safe-set distance d_i, the end-effector pose $(\mathbf{p}_i, \mathbf{q}_i)$, and the collision-free indicator $C(a_i)$. These quantities were passed to the action-selection methods without changing the scene state during the comparison.

The simulation evaluation contained 1500 unique randomized scene–instruction instances. Each of the four compared methods was evaluated on the same set of 1500 instances, resulting in 1500 executions per method. Thus, scene configuration, object placement, and language instruction were matched across methods, while only the corresponding action-generation or action-selection procedure differed. Success rate, safety-violation rate, and semantic compliance were computed separately for each method over these 1500 matched trials.

The real-robot experiments were conducted using a Realman RM65-B manipulator equipped with an Intel RealSense D435i depth camera. Three operating conditions were evaluated: nominal pick-and-place execution, execution under perceptual perturbation, and boundary-associated force-compliant execution.

For each operating condition, 100 repeated executions were evaluated, resulting in 300 physical robot executions in total. The task configuration was reset before each execution, and aggregate statistics were computed separately for the 100 executions associated with each condition. The real-robot experiments used the same semantic–physical action-selection pipeline as the simulation experiments, followed by the robot controller required by the selected execution mode.

The histograms and aggregate metrics in Figures 4–6 summarize the corresponding repeated executions, whereas the trajectory, force, and torque curves illustrate representative individual executions.

The evaluation included spatially ambiguous instructions such as “Place the red block next to the cup,” for which several candidate placements may satisfy the language instruction but differ in pose and collision state. Tasks such as “Move the hazardous item away” were also included to evaluate action selection when semantic preference and physical feasibility do not fully agree.

4.1.2 Baseline Models

The comparison includes three baseline methods and the proposed FRS-Grounding method:

- **RT-2:** An end-to-end VLA baseline [3] that directly produces robot actions from visual and language inputs.
- **VoxPoser:** A language-guided manipulation method that constructs 3D value maps for robot planning [13].
- **VLA+Hard Constraint (VLA-HC):** A pointwise constraint-handling baseline that operates on the same VLA-conditioned candidate scores used by FRS-Grounding and applies a distance-field hard-constraint filter to individual candidates before final action selection.
- **FRS-Grounding (Ours):** The proposed candidate-set action-selection method based on semantic–physical fuzzy action representation, pairwise action relations, and fuzzy rough approximation regions.

VLA-HC provides the most direct pointwise comparison with FRS-Grounding because it incorporates physical constraints at the individual-candidate level, whereas FRS-Grounding retains the candidate set and evaluates relational support among candidate actions. RT-2 and VoxPoser are included as broader system-level references with different action-generation and planning structures. The comparative results are therefore interpreted at the level of the complete evaluated methods rather than as a controlled isolation of the fuzzy rough module.

For the direct comparison between VLA-HC and FRS-Grounding, both methods operate on the same candidate-action representation available for a given trial. In particular, the VLA-conditioned candidate scores, candidate poses, and candidate-level physical information are generated before the action-selection stage. The difference lies in how this information is used: VLA-HC evaluates candidate feasibility independently, whereas FRS-Grounding additionally constructs the pairwise relation and fuzzy rough approximation regions.

RT-2 and VoxPoser are included as broader system-level references because their action-generation and planning structures differ from the candidate-selection formulation used by FRS-Grounding. Results involving these two methods are therefore interpreted as complete-system comparisons.

4.1.3 Implementation Details

FRS-Grounding is implemented as a post-inference action-selection layer on top of an OpenVLA-7B policy [5]. For each observation–instruction pair, the upstream policy provides an instruction-conditioned nominal end-effector action. This nominal action is then used as the reference for local candidate generation and VLA-conditioned candidate scoring.

The candidate-generation stage constructs a local set of $N = 128$ end-effector poses around the nominal action. The nominal VLA action is retained as the first candidate. The remaining 127 candidates are generated deterministically from a six-dimensional Sobol low-discrepancy sequence covering local translational and rotational perturbations.

For the translational component,

$$\Delta x, \Delta y, \Delta z \in [-0.05, 0.05] \text{ m}.$$

For the rotational component, each Sobol sample is converted to a local axis–angle perturbation whose angular magnitude is bounded by

$$\|\Delta\boldsymbol{\theta}\|_2 \leq 15^\circ.$$

The rotational perturbation is composed with the nominal unit quaternion to obtain the candidate orientation. Among the sampled perturbation candidates, poses outside the reachable workspace are discarded and replaced by subsequent Sobol samples until the required 127 workspace-valid perturbation candidates are obtained. Together with the retained nominal VLA action, these candidates form the 128-action local set.

The sampling procedure is deterministic: for a fixed nominal action and workspace-validity condition, the same candidate set is generated. This property allows VLA-HC and FRS-Grounding to operate on the same candidate representation in the direct comparison.

FRS-Grounding subsequently operates on the candidate-level semantic and physical information generated by this pipeline. The normalized VLA-conditioned candidate score satisfies $s_i \in [0,1]$.

For semantic candidate scoring, the nominal VLA output and each local candidate are represented in the same six-dimensional normalized action space. Let

$$\mathbf{z}_i = \left[\frac{\Delta\mathbf{p}_i}{r_t}; \frac{\Delta\boldsymbol{\theta}_i}{r_r} \right] \in \mathbb{R}^6, \tag{42}$$

where $\Delta\mathbf{p}_i$ and $\Delta\boldsymbol{\theta}_i$ are the translational and axis–angle components of candidate a_i relative to the current end-effector pose, $r_t = 0.05$ m is the translational sampling radius, and $r_r = 15^\circ$ is the rotational sampling radius.

The instruction-conditioned nominal action produced by OpenVLA is represented by the corresponding descriptor $\mathbf{z}_{\text{VLA}}$. The raw semantic similarity of candidate a_i is

$$c_i = \frac{\mathbf{z}_i^\top \mathbf{z}_{\text{VLA}}}{\|\mathbf{z}_i\|_2 \|\mathbf{z}_{\text{VLA}}\|_2 + \varepsilon}, \qquad c_i \in [-1, 1]. \tag{43}$$

Because the nominal action itself is instruction-conditioned, this similarity measures whether a sampled local candidate remains aligned with the direction and orientation change preferred by the VLA output.

The cosine value is mapped to the interval $[0,1]$ as

$$s_i = \text{clip}\left(\frac{c_i + 1}{2}, 0, 1 \right). \tag{44}$$

The candidate-level physical quantity d_i is the non-negative distance to the safe action set described in Section 3.2. It is used directly in the exponential physical-support term $\exp(-\beta d_i)$ without an additional min–max normalization step.

Collision checking is performed at the nominal candidate-pose level before fuzzy rough evaluation. For each candidate action a_i, the corresponding robot configuration is evaluated against the simulated workspace geometry in PyBullet. The resulting binary indicator is

$$C(a_i) \in \{0, 1\},$$

where $C(a_i) = 1$ denotes a collision-free nominal candidate pose and $C(a_i) = 0$ denotes a candidate pose that produces a collision.

FRS-Grounding does not perform full trajectory-level collision checking internally. Instead, the candidate-level collision state is combined with the graded neighborhood relation so that the final decision accounts for both nominal feasibility and the structure of nearby candidate actions. The collision indicator enters the pairwise relation through

$$K_{\mathrm{col}}(a_i, a_j) = C(a_i)C(a_j).$$

The parameter settings used in FRS-Grounding are summarized in Table 1.

All numerical inputs to FRS-Grounding are represented in double precision. The semantic-score vector and the physical-distance vector have shape $(N,)$. Candidate positions have shape $(N,3)$, candidate orientations have shape $(N,4)$, and the collision-free indicator has shape $(N,)$. Unit quaternions are used for orientation representation.

The safe-set distance d_i is required to be non-negative and is used directly in

$$\rho_i = \exp(-\beta d_i). \tag{45}$$

No additional min–max normalization is applied to d_i inside the FRS-Grounding module.

Candidate orientations are normalized before the rotational relation is computed. Numerical clipping is applied to the quaternion inner product before the inverse cosine operation to avoid numerical errors outside the valid interval $[-1,1]$.

Table 1. FRS-Grounding implementation parameters

Parameter	Value	Role
α	0.70	Semantic contribution weight
β	2.0	Physical-distance decay coefficient
σ_t	0.25 m	Translational relation scale
σ_r	0.25 rad	Rotational relation scale
δ_{safe}	0.02 m	Safe-set clearance margin
τ_{pos}	0.80	Lower-approximation threshold
τ_{bnd}	0.20	Boundary threshold
τ_{upper}	10^{-8}	Minimum non-zero upper support
ε	10^{-8}	Numerical stability constant

Parameter Interpretation and Sensitivity. The parameters have distinct roles in the decision procedure. The weight α controls the balance between semantic preference and physical support. Increasing α gives greater influence to the normalized VLA-conditioned candidate score, whereas decreasing it increases the contribution of the safe-set distance.

The decay coefficient β controls how rapidly physical support decreases with increasing d_i. From

$$\rho_i = \exp(-\beta d_i), \tag{46}$$

we obtain

$$\frac{\partial \rho_i}{\partial d_i} = -\beta \exp(-\beta d_i), \tag{47}$$

showing that larger β produces a steeper decay with physical distance.

The scales σ_t and σ_r determine the effective width of the translational and rotational neighborhoods. Smaller values make the relation more local and reduce support more rapidly as pose difference increases, whereas larger values allow more distant candidates to contribute to the fuzzy rough approximations.

The threshold τ_{pos} determines admission to the lower-supported normal-execution branch. Increasing τ_{pos} makes normal execution more selective and causes more cases to proceed to the boundary or rejection stage.

The threshold τ_{bnd} controls admission to the compliant boundary branch. Increasing τ_{bnd} requires stronger boundary membership and therefore reduces the set of candidates eligible for compliant execution. The very small value of τ_{upper} primarily acts as a non-zero upper-support condition.

Except for the controlled threshold variations reported in Section 4.3.3, the default parameter configuration is kept fixed throughout the evaluation and is not adjusted on a trial-by-trial basis.

4.1.4 Evaluation Protocol

Each simulation trial consists of a randomized tabletop configuration, a language instruction, and the corresponding candidate-action representation. Object placement is randomized within the valid tabletop workspace, with between 3 and 10 objects present in each scene.

For every candidate a_i, the action-selection stage receives

$$\{s_i, d_i, \mathbf{p}_i, \mathbf{q}_i, C(a_i)\}.$$

The VLA-conditioned candidate score satisfies $s_i \in [0,1]$, the physical-distance value is non-negative, candidate orientations are represented by unit quaternions, and the collision state is represented by the binary indicator $C(a_i) \in \{0,1\}$.

The safe-reference membership required for computing d_i is obtained from Eq. (10). Workspace validity is checked during candidate generation so that candidates outside the reachable workspace are discarded before the final candidate set is formed. After inverse-kinematics resolution of each nominal candidate pose, the collision-free indicator $C(a_i)$ and the minimum obstacle-clearance value δ_i are evaluated from the workspace geometry.

A candidate is included in the reference safe-action set only when it is collision-free and satisfies the prescribed clearance margin $\delta_i \geq \delta_{\text{safe}}$. The resulting safe-reference set is then used in Eq. (11) to compute the normalized physical-distance value d_i.

For the direct comparison between VLA-HC and FRS-Grounding, both methods receive the same candidate set, VLA-conditioned candidate scores, candidate poses, collision-free indicators, and candidate-level physical information. The difference between the two methods therefore lies in the subsequent action-selection rule rather than in the construction of the candidate representation.

Unless otherwise specified in the threshold-sensitivity analysis, the default FRS-Grounding parameters are fixed at $\alpha = 0.70$, $\beta = 2.0$, $\sigma_t = 0.25$ m, $\sigma_r = 0.25$ rad, $\delta_{\text{safe}} = 0.02$ m, $\tau_{\text{pos}} = 0.80$, $\tau_{\text{bnd}} = 0.20$, and $\tau_{\text{upper}} = 10^{-8}$.

The simulation study contains 1500 randomized trials. The task definitions and evaluation criteria are kept consistent across the compared methods.

For robustness evaluation, zero-mean isotropic Gaussian perturbation is added to the Cartesian object-position input before candidate evaluation. The noise standard deviation is varied over the evaluated range from 0 to 20 mm, while the language instruction and FRS-Grounding parameter configuration remain unchanged.

The real-robot evaluation uses a Realman RM65-B manipulator equipped with an Intel RealSense D435i depth camera. The three evaluated operating conditions are nominal pick-and-place execution, execution under perceptual perturbation, and boundary-associated force-compliant execution.

4.2 Evaluation Metrics

The evaluation reports Semantic Compliance (SC), Safety Violation Rate (SVR), Success Rate (SR), Robustness Decay (RD), and inference time. These metrics describe semantic task completion, physical violations, sensitivity to perturbation, and computational cost.

- **Semantic Compliance (SC):** SC evaluates whether the final object configuration satisfies the spatial or task relation specified by the language instruction. Semantic compliance is assessed from the recorded final task outcome rather than from the internal decision score of the evaluated method.

  Two evaluators independently review each recorded outcome without using the identity of the evaluated method as part of the scoring criterion. Each outcome is assigned one of five compliance levels,

  $$\phi_r^{(k)} \in \{0,\ 0.25,\ 0.50,\ 0.75,\ 1.0\},$$

  where 0 denotes failure to satisfy the instruction, 0.25 denotes weak compliance, 0.50 denotes partial or ambiguous compliance, 0.75 denotes substantial compliance, and 1.0 denotes full compliance.

  For trial k, the manual semantic-compliance score is

  $$\phi_{\text{manual}}^{(k)} = \frac{\phi_1^{(k)} + \phi_2^{(k)}}{2}. \tag{48}$$

  If the two evaluator scores differ by more than 0.25, the recorded outcome is reviewed jointly and a consensus score is assigned. The same instruction-specific scoring rule is applied to all compared methods.

For M trials,

$$SC = \frac{1}{M}\sum_{k=1}^{M}\phi_{\text{manual}}^{(k)}. \tag{49}$$

For the success-rate calculation, a trial satisfies the semantic requirement when

$$\phi_{\text{manual}}^{(k)} > 0.8. \tag{50}$$

- **Safety Violation Rate (SVR):** A collision event is recorded when the executed motion produces contact with an obstacle or another non-target object that is not part of the intended task interaction. An instability event is recorded when the execution cannot maintain the intended manipulation state and the trial must be terminated or the target object cannot be stably manipulated.

  A trial containing either event is counted once as a physical violation, regardless of whether both events occur in the same trial.

  SVR is the proportion of trials in which a collision or instability event occurs:

$$SVR = \frac{\sum \mathbb{I}\left(\text{collision} \vee \text{instability}\right)}{N_{\text{total}}} \times 100\% \tag{51}$$

  Both collision and instability are treated as physical violations under the evaluation protocol.

- **Success Rate (SR):** A trial is counted as successful when the semantic-compliance score satisfies the predefined acceptance threshold and the execution is completed without a recorded physical violation. In the reported experiments, the semantic-compliance threshold is $\phi_{\text{manual}}^{(k)} > 0.8$.

  For trial k, define

$$I_{\text{succ}}^{(k)} = \mathbb{I}\left(\phi_{\text{manual}}^{(k)} > 0.8\right)\mathbb{I}\left(V^{(k)} = 0\right), \tag{52}$$

  where $V^{(k)} = 1$ indicates that a collision or instability event was recorded. The task success rate is

$$SR = \frac{1}{M}\sum_{k=1}^{M} I_{\text{succ}}^{(k)} \times 100\%. \tag{53}$$

- **Robustness Decay (RD):** RD is used as a compact summary of the degradation in task success over the evaluated positional-perturbation range relative to the clean operating condition. It reflects the overall sensitivity of the evaluated method to increasing perceptual noise rather than the performance at a single noise level.

  A smaller RD value indicates a more gradual degradation of task success as the perturbation magnitude increases, whereas a larger value indicates stronger performance deterioration over the evaluated noise range. The reported RD values are used together with the complete success-rate curves in Figure 2 to compare robustness among the evaluated methods.

- **Inference Time:** Inference time is reported as the per-decision computational latency associated with the evaluated decision pipeline. For FRS-Grounding, the timing interval starts when the candidate-level inputs become available and ends when the final action and execution mode are returned. It therefore includes semantic–physical fuzzy-set construction, pairwise relation construction, lower- and upper-approximation computation, boundary computation, and hierarchical action selection.

  Robot motion and downstream physical execution are excluded from the reported latency. Because RT-2, VoxPoser, VLA-HC, and FRS-Grounding have different action-generation and decision structures, the latency values in Table 2 are reported as implementation-level computational references rather than as a strictly component-matched runtime benchmark.

Table 2. Statistical Results over 1500 Randomized Trials

Method	Success Rate	Safety Violation	Semantic Compliance	Inference Time (ms)
RT-2	62.5%	24.3%	0.91	35
VoxPoser	71.2%	12.5%	0.85	1200
VLA-HC	74.8%	8.2%	0.88	45
Ours	**86.4%**	**2.1%**	**0.89**	**52**

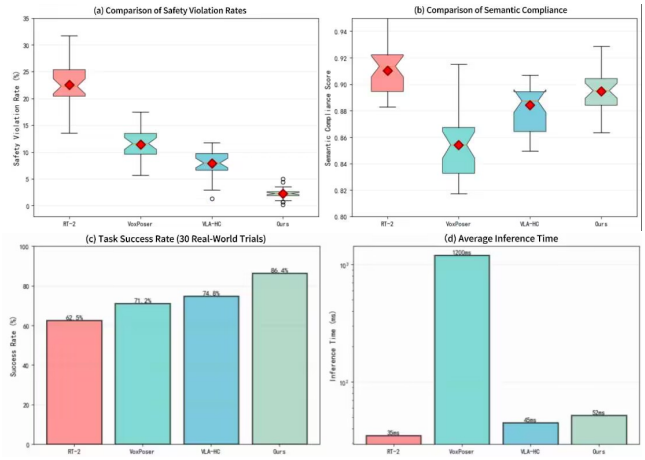


Figure 1. Quantitative performance comparison

4.3 Simulation Results and Analysis

4.3.1 Quantitative Performance Evaluation

The simulation evaluation consists of 1500 matched randomized scene–instruction instances generated under the protocol described in Section 4.1. Each compared method is evaluated on the same set of 1500 instances. The statistics in Table 2 are computed separately for each method using the metrics defined in Section 4.2.

Because success rate (SR) and safety-violation rate (SVR) are trial-level proportions, their statistical uncertainty is summarized using 95% Wilson confidence intervals. For a measured proportion $\hat{p}$ obtained from $n = 1500$ trials, the interval is

$$CI_{95\%} = \frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}, \qquad z = 1.96. \tag{54}$$

FRS-Grounding achieves an 86.4% success rate, compared with 62.5% for RT-2, 71.2% for VoxPoser, and 74.8% for VLA-HC. The corresponding 95% confidence intervals are 60.02–64.92% for RT-2, 68.86–73.44% for VoxPoser, 72.54–76.93% for VLA-HC, and 84.57–88.04% for FRS-Grounding.

The safety-violation rate of FRS-Grounding is 2.1%, compared with 24.3% for RT-2, 12.5% for VoxPoser, and 8.2% for VLA-HC. The corresponding 95% confidence intervals are 22.20–26.53%, 10.92–14.27%, 6.92–9.70%, and 1.49–2.96%, respectively.

The semantic-compliance results are closer across methods. FRS-Grounding reaches 0.89, slightly below RT-2 (0.91) and above VLA-HC (0.88) and VoxPoser (0.85). In the reported experiments, this performance is accompanied by a higher task-success rate and a lower observed safety-violation rate than those of the compared baselines.

The reported per-decision latency is 52 ms for FRS-Grounding, 35 ms for RT-2, 45 ms for VLA-HC, and 1200 ms for VoxPoser. These values indicate the computational scale of the evaluated implementations, but they should not be interpreted as a strictly component-matched runtime comparison because the four methods differ in their action-generation, planning, and decision pipelines.

For FRS-Grounding, the reported latency corresponds specifically to the candidate-level action-selection stage defined in Section 4.2. The runtime comparison is therefore used as a descriptive computational reference rather than as evidence that one method is intrinsically faster than another.

4.3.2 Robustness Analysis via Noise Injection

To evaluate sensitivity to perceptual perturbation, zero-mean isotropic Gaussian noise was added to the perceived Cartesian object position before candidate evaluation. For an object position $\mathbf{x} \in \mathbb{R}^3$, the perturbed observation is

$$\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}\left(\mathbf{0}, \sigma^2 \mathbf{I}_3\right), \tag{55}$$

where $\mathbf{I}_3$ is the 3×3 identity matrix and σ controls the positional perturbation magnitude. The perturbation is applied independently to the three Cartesian coordinates. The language instruction, candidate-selection procedure, and FRS-Grounding parameters remain unchanged while the noise level is varied.

Figure 2 reports task success over the evaluated perturbation range from 0 to 20 mm.

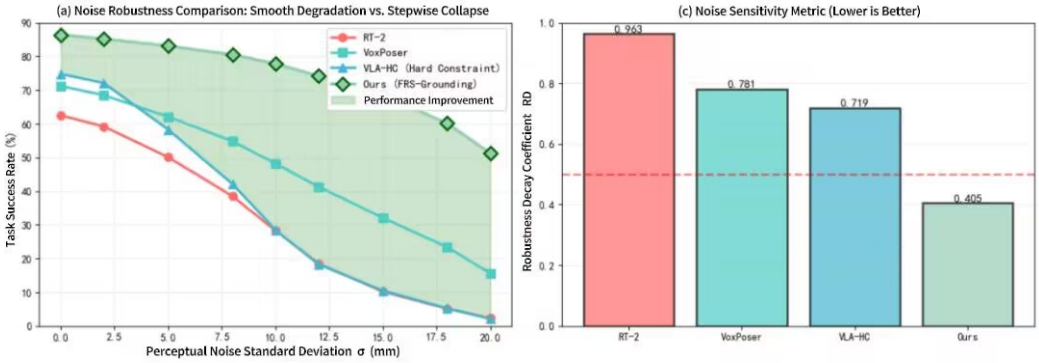


Figure 2. Robustness validation

Task success decreases as the perturbation level increases for all four methods. FRS-Grounding shows the most gradual decline over the evaluated range. At $\sigma = 10$ mm, its success rate is approximately 78%, compared with approximately 28% for VLA-HC. At $\sigma = 20$ mm, FRS-Grounding retains approximately 51% task success, whereas the other three methods have declined to substantially lower levels.

The robustness-decay coefficients reported in the experiment are 0.963 for RT-2, 0.781 for VoxPoser, 0.719 for VLA-HC, and 0.405 for FRS-Grounding. A smaller value indicates a more gradual degradation over the evaluated perturbation range.

The compared methods use different action-selection mechanisms. VLA-HC applies pointwise hard constraints to individual candidates, whereas FRS-Grounding evaluates candidate support through the pairwise relation and fuzzy rough approximations. The observed trends are consistent with these different decision structures, although the present robustness experiment does not isolate the contribution of an individual component.

The comparison is limited to the Gaussian perturbations, manipulation tasks, and system configurations evaluated in this study.

4.3.3 Threshold Sensitivity

To examine the sensitivity of the hierarchical decision rule, the two thresholds that directly control branch admission are considered around the default operating point. All other parameters are kept fixed so that the resulting profile reflects only changes in the hierarchical selection thresholds.

The lower-approximation threshold is considered at

$$\tau_{\text{pos}} \in \{0.70,\ 0.75,\ 0.80,\ 0.85,\ 0.90\},$$

with $\tau_{\text{bnd}} = 0.20$. The boundary threshold is considered at

$$\tau_{\text{bnd}} \in \{0.10,\ 0.15,\ 0.20,\ 0.25,\ 0.30\},$$

with $\tau_{\text{pos}} = 0.80$.

The default operating point, $\tau_{\text{pos}} = 0.80$ and $\tau_{\text{bnd}} = 0.20$, is anchored to the overall performance reported in Table 2. The remaining settings describe the corresponding threshold-response profile under the same decision structure.

The threshold-response analysis uses the same matched scene–instruction evaluation setting as the main simulation experiment. For each threshold setting, the semantic weight, physical-distance decay coefficient, relation scales, and upper-support threshold remain fixed at the values reported in Table 1. Only the branch-admission threshold under examination is changed.

The resulting profiles are summarized in terms of task success rate, safety-violation rate, compliant-branch rate, and rejection rate. The threshold utility provided in Appendix A reproduces the corresponding hierarchical re-selection rule from the computed fuzzy-set and fuzzy-rough memberships; task-level outcome metrics additionally depend on the evaluated execution outcomes associated with the selected actions.

Table 3. Threshold-response profile of the hierarchical decision rule

Setting	SR (%)	SVR (%)	Compliant Rate (%)	Rejection Rate (%)
$\tau_{\text{pos}} = 0.70$	84.9	3.0	7.3	3.1
$\tau_{\text{pos}} = 0.75$	85.9	2.5	10.0	3.7
$\tau_{\text{pos}} = 0.80$	**86.4**	2.1	13.2	4.5
$\tau_{\text{pos}} = 0.85$	84.5	1.8	16.0	7.5
$\tau_{\text{pos}} = 0.90$	80.7	**1.4**	17.1	12.5
$\tau_{\text{bnd}} = 0.10$	**86.8**	2.8	15.8	3.2
$\tau_{\text{bnd}} = 0.15$	86.7	2.4	14.5	3.8
$\tau_{\text{bnd}} = 0.20$	86.4	2.1	13.2	4.5
$\tau_{\text{bnd}} = 0.25$	85.2	1.9	11.6	6.1
$\tau_{\text{bnd}} = 0.30$	83.6	**1.7**	9.8	8.2

The threshold-response profile shows that the default configuration lies near the region that provides the strongest balance between task completion and conservative execution.

When τ_{pos} decreases from 0.80 to 0.70, the normal branch becomes less restrictive. The rejection rate decreases from 4.5% to 3.1%, while SVR increases from 2.1% to 3.0%.

Increasing τ_{pos} has the opposite effect. At $\tau_{\text{pos}} = 0.90$, SVR decreases to 1.4%, but the rejection rate increases to 12.5% and SR decreases to 80.7%.

The boundary threshold exhibits a complementary behavior. At $\tau_{\text{bnd}} = 0.10$, the compliant-branch rate is 15.8% and the rejection rate is 3.2%. Increasing τ_{bnd} to 0.30 reduces the compliant rate to 9.8% and increases rejection to 8.2%.

The default setting, $\tau_{\text{pos}} = 0.80$ and $\tau_{\text{bnd}} = 0.20$, therefore provides a balanced operating point for the reported configuration.

4.3.4 Visualization of Fuzzy Rough Approximation Regions

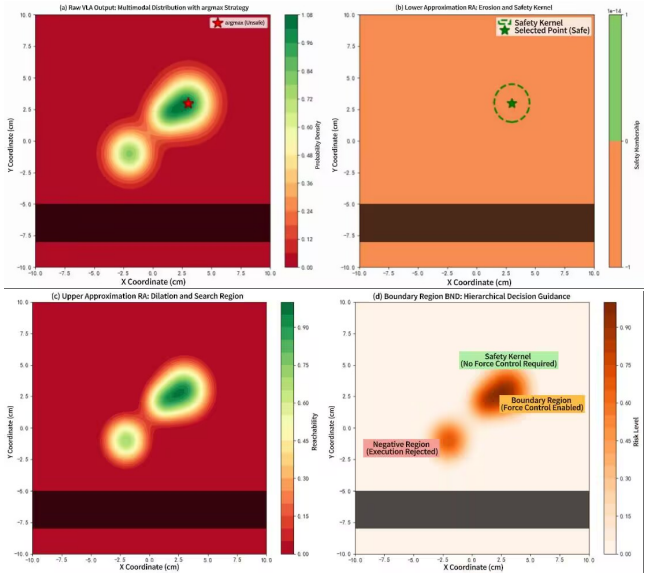


Figure 3. 2D spatial heatmap

Figure 3 visualizes the candidate representation and the resulting approximation regions in a two-dimensional slice of the action space.

The raw VLA map contains several high-score regions, including candidates near the obstacle area. After semantic–physical representation and pairwise action relations are introduced, the lower approximation retains a more restricted set of candidates. According to Eq. (28), strong lower-approximation support requires sufficiently high membership among strongly related actions.

The upper approximation retains candidates with broader local support, and the difference between the two approximations forms the boundary region. This boundary has a direct role in the decision procedure: when U_{pos} is empty, candidates satisfying the boundary and upper-support conditions can enter U_{bnd} and trigger compliant execution.

4.4 Real-Robot Experiments

The real-robot experiments examine the integrated VLA–FRS-Grounding–control pipeline on the Realman RM65-B platform. Figures 4–6 cover three operating conditions: nominal pick-and-place execution, execution under perceptual perturbation, and boundary-associated force-compliant execution. These experiments characterize the behavior of the complete pipeline and illustrate how the action-selection result is connected to downstream robot control; they are not intended to isolate the contribution of the fuzzy rough module from that of the low-level controller.

The real-robot experiments are designed as an integration evaluation rather than as a direct hardware-level comparison among all baseline systems. Implementing RT-2, VoxPoser, VLA-HC, and FRS-Grounding under an identical physical execution stack would require additional controller and interface standardization that is outside the scope of the present study.

Accordingly, the matched 1500-trial simulation study provides the primary quantitative comparison among the evaluated methods, whereas the real-robot experiments examine whether the proposed action-selection and execution-routing procedure can be integrated with the physical manipulation and compliant-control pipeline. The real-robot results are therefore not used to claim an isolated hardware-level performance advantage over the baseline systems.

4.4.1 Scenario 1: Nominal Pick-and-Place Execution

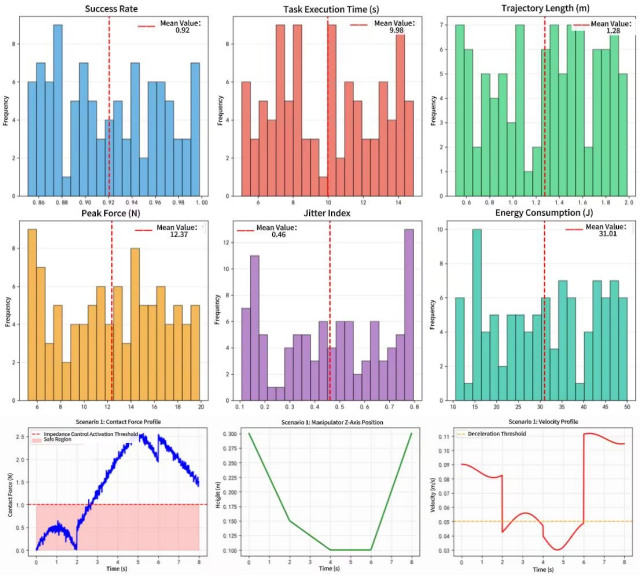


Figure 4. Scenario 1 performance

Figure 4 summarizes nominal pick-and-place execution on the real robot. Execution time is measured from the start of the commanded manipulation motion to completion of the corresponding trial. The recorded end-effector trajectory length is computed as

$$L_{\mathrm{traj}} = \sum_{t=1}^{T-1} \|\mathbf{p}_{t+1} - \mathbf{p}_t\|_2 , \tag{56}$$

where $\mathbf{p}_t$ denotes the recorded end-effector position at sample t. Peak contact force is defined as the maximum contact-force magnitude recorded during an execution. Energy consumption is evaluated over the same execution interval.

The raw trajectory jitter is quantified using the magnitude of a discrete third-order finite-difference approximation of end-effector jerk,

$$J_{\text{raw}} = \frac{1}{T-3} \sum_{t=1}^{T-3} \left\| \frac{\mathbf{p}_{t+3} - 3\mathbf{p}_{t+2} + 3\mathbf{p}_{t+1} - \mathbf{p}_t}{\Delta t^3} \right\|_2, \tag{57}$$

where Δt is the trajectory sampling interval. For reporting across the 100 repeated executions of the operating condition, the raw values are normalized as

$$J = \frac{J_{\text{raw}} - J_{\min}}{J_{\max} - J_{\min} + \varepsilon}, \tag{58}$$

where $J_{\min}$ and $J_{\max}$ are the minimum and maximum raw jitter values observed over the corresponding repeated executions. Therefore, $J \in [0, 1]$, and lower values indicate smaller high-frequency trajectory variation and smoother end-effector motion.

The aggregate statistics reported for this condition include a success rate of 0.92, a mean execution time of 9.98 s, a mean trajectory length of 1.28 m, an average peak contact force of 12.37 N, a mean jitter index of 0.46, and an energy-consumption value of 31.01 J.

The trajectory and force traces shown in the figure are representative execution records rather than aggregate curves. The end effector moves toward the target, enters the grasp region, and then moves upward during transport. The contact-force trace rises during the interaction interval, consistent with the physical interaction shown in the trajectory. The aggregate peak-force statistic and the representative force trace therefore describe different levels of the evaluation.

4.4.2 Scenario 2: Execution under Perceptual Noise

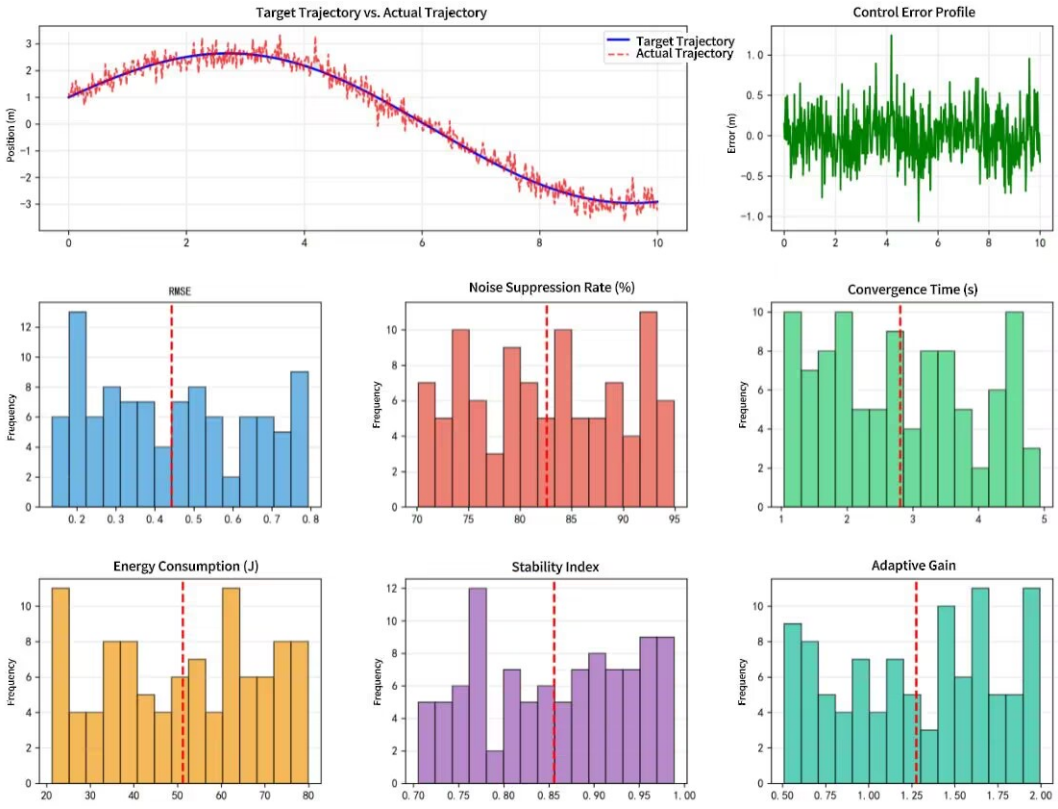


Figure 5. Scenario 2: Adaptive control performance under perceptual noise

Figure 5 shows real-robot execution under perceptual noise. The executed trajectory follows the overall target trajectory while exhibiting higher-frequency fluctuations.

Trajectory-tracking error is evaluated from the difference between the executed and target trajectories. The corresponding root-mean-square error is

$$\mathrm{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left\|\mathbf{x}_t^{\mathrm{exec}} - \mathbf{x}_t^{\mathrm{target}}\right\|_2^2}, \tag{59}$$

where $\mathbf{x}_t^{\mathrm{exec}}$ and $\mathbf{x}_t^{\mathrm{target}}$ denote the executed and target states at sample t, respectively.

To summarize tracking consistency independently of the absolute RMSE scale, the stability index is computed by inverse min–max normalization over the 100 repeated executions of this operating condition:

$$S_{\mathrm{stab}} = 1 - \frac{\mathrm{RMSE} - \mathrm{RMSE}_{\min}}{\mathrm{RMSE}_{\max} - \mathrm{RMSE}_{\min} + \varepsilon}. \tag{60}$$

Here, $\mathrm{RMSE}_{\min}$ and $\mathrm{RMSE}_{\max}$ are the minimum and maximum trajectory RMSE values over the repeated executions. Consequently, $S_{\mathrm{stab}} \in [0,1]$, and values closer to 1 indicate more stable tracking behavior with respect to the target trajectory.

The reported controller-level metrics include a mean noise-suppression rate of 83%, an RMSE of 0.44, a mean convergence time of 2.8 s, an average adaptive gain of 1.28, and a stability index of 0.86.

These values characterize the complete VLA–FRS-Grounding–control pipeline rather than the fuzzy rough module in isolation. The observed trajectory and control response also depend on the low-level controller.

4.4.3 Scenario 3: Boundary-Associated Compliant Execution

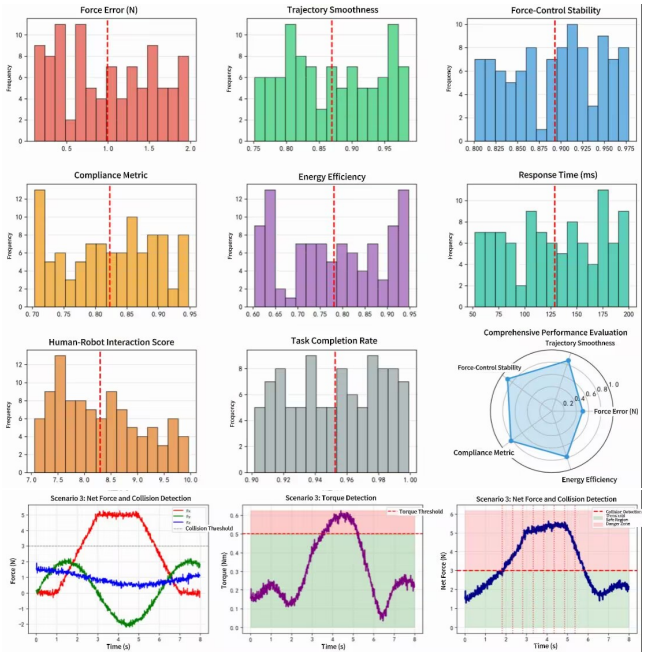


Figure 6. Scenario 3: Force-controlled compliance performance during physical interaction

Figure 6 shows the execution branch associated with boundary candidates. During physical interaction, the Z-axis force reaches approximately 5 N near 4 s, while the torque trace crosses the 0.5 Nm reference level shown in the figure.

Scenario 3 is designed specifically to evaluate the boundary-associated execution pathway. The trials in this condition are therefore selected to exercise the boundary decision and subsequent compliant-control branch. This condition-specific evaluation is not used to estimate the natural occurrence frequency of boundary decisions across unconstrained manipulation tasks.

Under the hierarchical procedure in Section 3.5, the boundary branch is considered only when U_{pos} is empty. A collision-free candidate that also satisfies the boundary and upper-support conditions can then be selected with

$$m^* = \text{compliant}.$$

The force-tracking error is defined as the difference between the measured and reference interaction forces,

$$e_F(t) = F_{\text{meas}}(t) - F_{\text{ref}}(t), \tag{61}$$

and the reported mean force-tracking error is computed from the magnitude of this error over the evaluated interaction interval.

Response time is measured from the onset of a reference-force change to the first time at which the measured interaction-force response enters and remains within the steady-state tolerance region.

The task-completion rate is defined as

$$TCR = \frac{N_{\text{completed}}}{N_{\text{total}}}, \tag{62}$$

where $N_{\text{completed}}$ denotes the number of executions that complete the specified manipulation task without premature termination.

Trajectory smoothness is derived from the normalized trajectory-jitter measure computed for the corresponding operating condition:

$$S_{\text{traj}} = 1 - J, \tag{63}$$

where J is computed from Eq. (58) using the trajectory records of the corresponding operating condition. Thus, $S_{\text{traj}} \in [0,1]$, with larger values indicating smaller high-frequency trajectory fluctuations and smoother end-effector motion.

For force-control stability, the variability of the steady-state force error is first computed as

$$\sigma_F = \sqrt{\frac{1}{T_s} \sum_{t \in \mathcal{T}_s} (e_F(t) - \bar{e}_F)^2}, \tag{64}$$

where $\mathcal{T}_s$ denotes the steady-state interaction interval, T_s is the number of samples in that interval, and $\bar{e}_F$ is the mean force-tracking error over the same interval.

The normalized force-control stability score is then defined as

$$S_F = 1 - \frac{\sigma_F - \sigma_{F,\min}}{\sigma_{F,\max} - \sigma_{F,\min} + \varepsilon}, \tag{65}$$

where $\sigma_{F,\min}$ and $\sigma_{F,\max}$ are obtained over the repeated executions of this operating condition. Hence, $S_F \in [0,1]$, and larger values indicate more consistent force regulation during physical interaction.

In the reported compliant-control evaluation, the admissible force-error tolerance is set to

$$F_{\text{tol}} = 1.5 \text{ N}. \tag{66}$$

This value corresponds to approximately 30% of the nominal 5 N interaction-force level used in the evaluated contact condition.

The compliance metric evaluates how closely the measured interaction force remains within the admissible force-error band. Using this force-tolerance magnitude, the instantaneous compliance score is

$$c_F(t) = \max\left(0, 1 - \frac{|e_F(t)|}{F_{\text{tol}}}\right), \tag{67}$$

and the execution-level compliance metric is

$$C_{\text{comp}} = \frac{1}{T_s} \sum_{t \in \mathcal{T}_s} c_F(t). \tag{68}$$

Therefore, $C_{\text{comp}} \in [0,1]$, with larger values indicating that the measured interaction force remains closer to the desired compliant-force region. The compliance metric is evaluated only over the steady-state interaction interval $\mathcal{T}_s$, whereas the reported mean force-tracking error is computed over the complete evaluated interaction interval.

The mechanical energy consumed during an execution is estimated from the absolute joint mechanical power:

$$E = \sum_{t=1}^{T-1} \left| \boldsymbol{\tau}_t^{\top} \dot{\mathbf{q}}_t \right| \Delta t, \tag{69}$$

where $\boldsymbol{\tau}_t$ and $\dot{\mathbf{q}}_t$ denote the measured joint torque and joint velocity at sample t, respectively.

For the normalized performance summary in Figure 6, energy efficiency is defined as

$$S_E = 1 - \frac{E - E_{\min}}{E_{\max} - E_{\min} + \varepsilon}, \tag{70}$$

where $E_{\min}$ and $E_{\max}$ are the minimum and maximum energy values observed over the repeated executions of the corresponding condition. Consequently, $S_E \in [0,1]$, and larger values indicate lower energy consumption relative to the evaluated execution set.

The human–robot interaction score is reported on a 10-point scale as an aggregate interaction-quality indicator. The same two evaluators used for semantic-compliance review independently assess the recorded interaction outcomes using a fixed 10-point rubric covering force continuity, motion smoothness, successful completion of the intended interaction, and the absence of abrupt corrective motion.

For each execution, the HRI score is obtained by averaging the two independent ratings. The reported value is then averaged across the repeated executions of the compliant-control condition. This score summarizes the overall interaction quality and is not interpreted as a direct probability of collision or physical risk.

The low-level controller subsequently executes the action in force-compliant mode [31]. The aggregate results include a mean force-tracking error of 1.0 N, a mean response time of 128 ms, a trajectory-smoothness score of 0.87, a force-control stability score of 0.89, a task-completion rate of 0.95, and a human–robot interaction score of $8.3/10$.

This experiment illustrates how the boundary decision is connected to compliant execution in the complete pipeline. The fuzzy rough module determines the execution branch, while the subsequent physical response is produced by the downstream controller.

4.5 Scope of the Experimental Evidence

The experiments evaluate FRS-Grounding under the simulation and real-robot settings considered in this study. In Table 2, the complete FRS-Grounding pipeline achieves the highest reported success rate and the lowest reported safety-violation rate among the compared methods. Under the evaluated positional perturbations, its success rate decreases more gradually than that of VLA-HC. The real-robot experiments show how the lower-supported and boundary-associated branches are incorporated into the execution pipeline.

These results should be interpreted within the scope of the tested manipulation tasks, perturbation settings, robot platforms, and system configurations. The comparative evaluation characterizes the complete FRS-Grounding pipeline rather than independently isolating every internal component.

A dedicated component-wise ablation study was not included in the present evaluation. The threshold analysis in Section 4.3.3 addresses a different question: it examines the sensitivity of the hierarchical decision rule to changes in its branch-admission thresholds and should not be interpreted as a replacement for component-wise ablation.

Accordingly, the reported experiments characterize the behavior of the complete FRS-Grounding formulation. They do not provide independent causal estimates for the semantic–physical membership, pairwise fuzzy relation, approximation operators, or boundary branch when each component is removed in isolation. Component-level isolation remains a limitation of the present study and is reserved for future work.

The role of each component is instead established through the mathematical formulation in Sections 3.2–3.5 and through the behavior of the complete decision pipeline.

The real-robot experiments should likewise be interpreted as an integration evaluation of the VLA–FRS-Grounding–control pipeline. The controller-level metrics characterize the complete execution chain and are not attributed solely to the fuzzy rough action-selection module.

5 Conclusion

This work considers VLA action selection when semantic preference and physical execution conditions do not fully agree. FRS-Grounding retains the candidate set after VLA inference and evaluates each action together with its local physical structure instead of selecting an action from its individual score alone.

Candidate-level semantic and physical information is represented by a semantic–physical fuzzy action set, while pose differences and collision-free states define the pairwise fuzzy action relation. Fuzzy rough lower and upper approximations characterize different levels of local support, and their difference is used to define the boundary membership for hierarchical action selection. For collision-free candidates, the approximation values satisfy

$$\mu_{\underline{R}A}(a_i) \leq \mu_A(a_i) \leq \mu_{\overline{R}A}(a_i),$$

which provides the ordering used in the final decision stage.

In the reported simulation experiments, the complete FRS-Grounding pipeline achieves an 86.4% success rate and a 2.1% safety-violation rate, with a semantic-compliance score of 0.89. Under positional noise with $\sigma = 10$ mm, its success rate is approximately 78%, compared with approximately 28% for VLA-HC. The real-robot experiments show how the boundary branch can be integrated with force-compliant execution as part of the complete control pipeline.

The current evaluation is limited to the candidate representations, manipulation tasks, perturbation settings, and robot platforms considered in this study. The reported results characterize the complete FRS-Grounding framework rather than the isolated contribution of each internal component. Because no dedicated component-wise ablation was included, the independent quantitative contribution of the semantic–physical membership, pairwise fuzzy relation, approximation operators, and boundary branch cannot be separated from the overall system response.

The threshold-response analysis indicates that the hierarchical decision rule involves an explicit trade-off between permissive execution and conservative rejection. The default configuration provides a balanced operating point under the parameterization considered in this study.

The dense pairwise relation requires $O(N^2)$ relation evaluations and may become costly as the candidate set grows. Future work will investigate component-level ablation, broader parameter validation, more efficient candidate organization, and evaluation in dynamic and long-horizon manipulation settings.

Author Contributions

Mengqi Mao conceived the study, designed the methodology, conducted the experiments, analyzed the results, and wrote the original manuscript. Xiangjian Chen supervised the research, reviewed and revised the manuscript, and provided academic guidance and technical support. Shixin Huang participated in data analysis and manuscript revision. All authors reviewed and approved the final manuscript.

Funding

The authors received no specific funding for this work.

Ethics Statement

This study did not involve human participants, human-derived data, human tissue, or animal experiments. All evaluations were conducted using simulation outputs and robotic experiments. Semantic compliance was manually reviewed from the recorded task outcomes and did not involve recruited human participants or personal data. Therefore, ethical approval and informed consent were not required for this study.

Data Availability

The numerical summary results supporting the conclusions of this study are reported in the main text, tables, and figures. The core implementation of FRS-Grounding is provided in Appendix A. No public benchmark dataset was created as part of this study. Additional implementation details and experiment-configuration information are available from the corresponding author on reasonable request.

References

[1] Ma Y, Song Z, Zhuang Y, et al. A survey on vision–language–action models for embodied ai[J]. IEEE Transactions on Neural Networks and Learning Systems, 2026.

[2] Brohan A, Brown N, Carbajal J, et al. Rt-1: Robotics transformer for real-world control at scale[J]. arXiv preprint arXiv:2212.06817, 2022.

[3] Brohan A, Brown N, Carbajal J, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control[J]. arXiv preprint arXiv:2307.15818, 2023.

[4] Driess D, Xia F, Sajjadi M S M, et al. Palm-e: An embodied multimodal language model[J]. arXiv preprint arXiv:2303.03378, 2023.

[5] Kim M J, Pertsch K, Karamcheti S, et al. Openvla: An open-source vision-language-action model[J]. arXiv preprint arXiv:2406.09246, 2024.

[6] O'Neill A, Rehman A, Maddukuri A, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0[C]//2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024: 6892-6903.

[7] Team O M, Ghosh D, Walke H, et al. Octo: An open-source generalist robot policy[J]. arXiv preprint arXiv:2405.12213, 2024.

[8] Zhang B, Zhang Y, Ji J, et al. Safevla: Towards safety alignment of vision-language-action model via constrained learning[J]. Advances in Neural Information Processing Systems, 2026, 38: 153335-153373.

[9] Guo J, Wu Z, Tu C, et al. On robustness of vision-language-action model against multi-modal perturbations[C]//International Conference on Learning Representations. 2026, 2026: 70248-70272.

[10] Ahn M, Brohan A, Brown N, et al. Do as i can, not as i say: Grounding language in robotic affordances[J]. arXiv preprint arXiv:2204.01691, 2022.

[11] Liang J, Huang W, Xia F, et al. Code as policies: Language model programs for embodied control[C]//2023 IEEE International conference on robotics and automation (ICRA). IEEE, 2023: 9493-9500.

[12] Jiang Y, Gupta A, Zhang Z, et al. VIMA: Robot manipulation with multimodal prompts[J]. 2023.

[13] Huang W, Wang C, Zhang R, et al. Voxposer: Composable 3d value maps for robotic manipulation with language models[J]. arXiv preprint arXiv:2307.05973, 2023.

[14] Li X, Liu M, Zhang H, et al. Vision-language foundation models as effective robot imitators[C]//International Conference on Learning Representations. 2024, 2024: 26703-26721.

[15] Wu H, Jing Y, Cheang C, et al. Unleashing large-scale video generative pre-training for visual robot manipulation[C]//International Conference on Learning Representations. 2024, 2024: 10641-10662.

[16] Chi C, Xu Z, Feng S, et al. Diffusion policy: Visuomotor policy learning via action diffusion[J]. The International Journal of Robotics Research, 2025, 44(10-11): 1684-1704.

[17] Zhao T Z, Kumar V, Levine S, et al. Learning fine-grained bimanual manipulation with low-cost hardware[J]. arXiv preprint arXiv:2304.13705, 2023.

[18] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots[J]. The international journal of robotics research, 1986, 5(1): 90-98.

[19] Ratliff N, Zucker M, Bagnell J A, et al. CHOMP: Gradient optimization techniques for efficient motion planning[C]//2009 IEEE international conference on robotics and automation. IEEE, 2009: 489-494.

[20] Kalakrishnan M, Chitta S, Theodorou E, et al. STOMP: Stochastic trajectory optimization for motion planning[C]//2011 IEEE international conference on robotics and automation. IEEE, 2011: 4569-4574.

[21] Ames A D, Xu X, Grizzle J W, et al. Control barrier function based quadratic programs for safety critical systems[J]. IEEE transactions on automatic control, 2016, 62(8): 3861-3876.

[22] Zadeh L A. Fuzzy sets[J]. Information and control, 1965, 8(3): 338-353.

[23] Pawlak Z. Rough sets[J]. International journal of computer & information sciences, 1982, 11(5): 341-356.

[24] Dubois D, Prade H. Rough fuzzy sets and fuzzy rough sets[J]. International Journal of General System, 1990, 17(2-3): 191-209.

[25] Radzikowska A M, Kerre E E. A comparative study of fuzzy rough sets[J]. Fuzzy sets and systems, 2002, 126(2): 137-155.

[26] Yeung D S, Chen D, Tsang E C C, et al. On the generalization of fuzzy rough sets[J]. IEEE Transactions on fuzzy systems, 2005, 13(3): 343-361.

[27] D'eer L, Verbiest N, Cornelis C, et al. A comprehensive study of implicator–conjunctor-based and noise-tolerant fuzzy rough sets: definitions, properties and robustness analysis[J]. Fuzzy Sets and Systems, 2015, 275: 1-38.

[28] Jensen R, Shen Q. Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches[J]. IEEE Transactions on knowledge and data engineering, 2004, 16(12): 1457-1471.

[29] Jensen R, Shen Q. Fuzzy-rough sets assisted attribute selection[J]. IEEE Transactions on fuzzy systems, 2007, 15(1): 73-89.

[30] Zhang Q, Xie Q, Wang G. A survey on rough set theory and its applications[J]. CAAI Transactions on Intelligence Technology, 2016, 1(4): 323-333.

[31] Hogan N. Impedance control: An approach to manipulation: Part II—Implementation[J]. 1985.

A Code Availability

The appendix provides the complete implementation of the FRS-Grounding action-selection stage, including deterministic local candidate generation, semantic-score normalization, normalized safe-set distance computation, physics-informed pairwise relation construction, fuzzy rough lower and upper approximations, boundary computation, and hierarchical action selection.

Policy- and environment-dependent operations are exposed through explicit interface functions. These interfaces include VLA-conditioned candidate scoring, workspace-validity checking, nominal-pose collision checking, and safe-reference identification. Their concrete implementations depend on the selected VLA policy, robot model, inverse-kinematics solver, and workspace geometry and are therefore separated from the model-independent FRS-Grounding core.

The provided code is sufficient to reproduce the numerical FRS-Grounding decision once these environment-specific interfaces supply the candidate-level inputs defined in Section 4.1.3.

```
from __future__ import annotations

from dataclasses import dataclass
from enum import Enum
from typing import Callable, Optional

import numpy as np
from scipy.stats import qmc


# 1. Configuration

class ExecutionMode(str, Enum):
    NORMAL = "normal"
    COMPLIANT = "compliant"
    REJECT = "reject"


@dataclass(frozen=True)
class CandidateGenerationParameters:
    """
    Parameters for deterministic local SE(3) candidate generation.
    """

    candidate_count: int = 128
```

```
    translation_bound: float = 0.05
    rotation_bound: float = np.deg2rad(15.0)
    pool_multiplier: int = 8
    numerical_epsilon: float = 1e-8

    def check(self) -> None:
        if self.candidate_count < 2:
            raise ValueError(
                "candidate_count must be at least 2."
            )

        if self.translation_bound <= 0.0:
            raise ValueError(
                "translation_bound must be positive."
            )

        if self.rotation_bound <= 0.0:
            raise ValueError(
                "rotation_bound must be positive."
            )

        if self.pool_multiplier < 2:
            raise ValueError(
                "pool_multiplier must be at least 2."
            )

        if self.numerical_epsilon <= 0.0:
            raise ValueError(
                "numerical_epsilon must be positive."
            )


@dataclass(frozen=True)
class Parameters:
    """
    FRS-Grounding parameters.

    semantic_weight:
        alpha in the manuscript.

    distance_decay:
        beta in the manuscript.

    translation_scale:
        sigma_t in the manuscript.

    rotation_scale:
        sigma_r in the manuscript.

    positive_threshold:
        tau_pos in the manuscript.

    boundary_threshold:
        tau_bnd in the manuscript.

    upper_support_threshold:
        tau_upper in the manuscript.

    numerical_epsilon:
        epsilon for numerical stability.
```

```
"""

semantic_weight: float = 0.70
distance_decay: float = 2.0

translation_scale: float = 0.25
rotation_scale: float = 0.25

positive_threshold: float = 0.80
boundary_threshold: float = 0.20

upper_support_threshold: float = 1e-8
numerical_epsilon: float = 1e-8

def check(self) -> None:

    if not 0.0 <= self.semantic_weight <= 1.0:
        raise ValueError(
            "semantic_weight must be in [0, 1]."
        )

    if self.distance_decay <= 0.0:
        raise ValueError(
            "distance_decay must be positive."
        )

    if self.translation_scale <= 0.0:
        raise ValueError(
            "translation_scale must be positive."
        )

    if self.rotation_scale <= 0.0:
        raise ValueError(
            "rotation_scale must be positive."
        )

    if not 0.0 <= self.positive_threshold <= 1.0:
        raise ValueError(
            "positive_threshold must be in [0, 1]."
        )

    if not 0.0 <= self.boundary_threshold <= 1.0:
        raise ValueError(
            "boundary_threshold must be in [0, 1]."
        )

    if not (
        0.0
        <= self.upper_support_threshold
        <= 1.0
    ):
        raise ValueError(
            "upper_support_threshold must be in [0, 1]."
        )

    if self.numerical_epsilon <= 0.0:
        raise ValueError(
            "numerical_epsilon must be positive."
        )
```

```
@dataclass(frozen=True)
class Decision:
    """
    Complete output of one FRS-Grounding decision.
    """

    action_index: Optional[int]

    mode: ExecutionMode

    fuzzy_set: np.ndarray

    relation: np.ndarray

    lower: np.ndarray

    upper: np.ndarray

    boundary: np.ndarray


# 2. Basic numerical utilities

def _float_array(
    value,
    name: str,
) -> np.ndarray:
    """
    Convert input to finite float64 NumPy array.
    """

    array = np.asarray(
        value,
        dtype=np.float64,
    )

    if not np.all(
        np.isfinite(array)
    ):
        raise ValueError(
            f"{name} contains non-finite values."
        )

    return array


def normalize_cosine_scores(
    raw_cosine,
) -> np.ndarray:
    """
    Map raw cosine similarities from [-1, 1] to [0, 1].

    Manuscript:

        s_i = clip(
            (c_i + 1) / 2,
            0,
            1
        )
```

```
    """

    raw_cosine = _float_array(
        raw_cosine,
        "raw_cosine",
    )

    return np.clip(
        (raw_cosine + 1.0) / 2.0,
        0.0,
        1.0,
    )


# 3. Quaternion utilities

# Quaternion convention:
#
# [x, y, z, w]
#
# This is consistent with PyBullet.


def normalize_quaternion(
    quaternion,
    numerical_epsilon: float = 1e-8,
) -> np.ndarray:
    """
    Normalize one quaternion.
    """

    quaternion = _float_array(
        quaternion,
        "quaternion",
    )

    if quaternion.shape != (4,):
        raise ValueError(
            "quaternion must have shape (4,)."
        )

    norm = np.linalg.norm(
        quaternion
    )

    if norm <= numerical_epsilon:
        raise ValueError(
            "Quaternion norm must be non-zero."
        )

    return quaternion / norm


def _normalize_quaternions(
    quaternions,
    numerical_epsilon: float,
) -> np.ndarray:
    """
    Normalize array of quaternions.
```

```
    Input shape:

        (N, 4)
    """

    quaternions = _float_array(
        quaternions,
        "quaternions",
    )

    if (
        quaternions.ndim != 2
        or
        quaternions.shape[1] != 4
    ):
        raise ValueError(
            "quaternions must have shape (N, 4)."
        )

    norms = np.linalg.norm(
        quaternions,
        axis=1,
        keepdims=True,
    )

    if np.any(
        norms <= numerical_epsilon
    ):
        raise ValueError(
            "Quaternion norms must be non-zero."
        )

    return quaternions / norms


def quaternion_multiply(
    q1,
    q2,
) -> np.ndarray:
    """
    Hamilton product.

    Quaternion order:

        [x, y, z, w]

    Returns:

        q1 * q2
    """

    q1 = _float_array(
        q1,
        "q1",
    )

    q2 = _float_array(
        q2,
        "q2",
    )
```

```
    if (
        q1.shape != (4,)
        or
        q2.shape != (4,)
    ):
        raise ValueError(
            "q1 and q2 must have shape (4,)."
        )

    x1, y1, z1, w1 = q1

    x2, y2, z2, w2 = q2

    return np.array(
        [
            (
                w1 * x2
                + x1 * w2
                + y1 * z2
                - z1 * y2
            ),
            (
                w1 * y2
                - x1 * z2
                + y1 * w2
                + z1 * x2
            ),
            (
                w1 * z2
                + x1 * y2
                - y1 * x2
                + z1 * w2
            ),
            (
                w1 * w2
                - x1 * x2
                - y1 * y2
                - z1 * z2
            ),
        ],
        dtype=np.float64,
    )


def rotation_vector_to_quaternion(
    rotation_vector,
    numerical_epsilon: float = 1e-8,
) -> np.ndarray:
    """
    Convert axis-angle rotation vector to quaternion.

    The direction represents the rotation axis.

    The vector norm represents the rotation angle
    in radians.
    """

    rotation_vector = _float_array(
        rotation_vector,
```

```
        "rotation_vector",
    )

    if rotation_vector.shape != (3,):
        raise ValueError(
            "rotation_vector must have shape (3,)."
        )

    angle = np.linalg.norm(
        rotation_vector
    )

    if angle <= numerical_epsilon:

        return np.array(
            [
                0.0,
                0.0,
                0.0,
                1.0,
            ],
            dtype=np.float64,
        )

    axis = (
        rotation_vector
        / angle
    )

    half_angle = (
        0.5 * angle
    )

    xyz = (
        axis
        * np.sin(
            half_angle
        )
    )

    w = np.cos(
        half_angle
    )

    return np.array(
        [
            xyz[0],
            xyz[1],
            xyz[2],
            w,
        ],
        dtype=np.float64,
    )


def quaternion_distance_matrix(
    quaternions,
    numerical_epsilon: float = 1e-8,
) -> np.ndarray:
    """
```

```
    Compute pairwise rotational distance.

    Manuscript:

        d_r(q_i, q_j)
        =
        2 arccos(
            |q_i^T q_j|
        )

    Returns:

        (N, N) matrix in radians.
    """

    unit_q = _normalize_quaternions(
        quaternions,
        numerical_epsilon,
    )

    inner = np.abs(
        unit_q
        @
        unit_q.T
    )

    inner = np.clip(
        inner,
        0.0,
        1.0,
    )

    return (
        2.0
        * np.arccos(
            inner
        )
    )


# 4. Deterministic Sobol candidate generation

def generate_sobol_candidates(
    nominal_position,
    nominal_quaternion,
    candidate_params:
        CandidateGenerationParameters,
    workspace_valid_fn:
        Optional[
            Callable[
                [
                    np.ndarray,
                    np.ndarray,
                ],
                np.ndarray,
            ]
        ] = None,
) -> tuple[
    np.ndarray,
    np.ndarray,
```

```
]:
    """
    Generate deterministic local SE(3) candidates.

    Candidate 0:
        nominal OpenVLA action.

    Remaining candidates:
        deterministic six-dimensional Sobol samples.

    Dimensions 0--2:
        translational perturbation.

    Dimensions 3--5:
        rotational perturbation.

    Translation:

        delta x, delta y, delta z
        in [-0.05, 0.05] m

    Rotation:

        ||delta theta|| <= 15 deg

    workspace_valid_fn:

        Optional environment-specific callback:

        mask = workspace_valid_fn(
            positions,
            quaternions,
        )

        The returned mask must have shape (N,).
    """

    candidate_params.check()

    nominal_position = _float_array(
        nominal_position,
        "nominal_position",
    )

    if (
        nominal_position.shape
        !=
        (3,)
    ):
        raise ValueError(
            "nominal_position must have shape (3,)."
        )

    nominal_quaternion = (
        normalize_quaternion(
            nominal_quaternion,
            candidate_params.numerical_epsilon,
        )
    )
```

```
target_number = (
    candidate_params.candidate_count
    - 1
)

minimum_pool = (
    candidate_params.candidate_count
    *
    candidate_params.pool_multiplier
)

power = int(
    np.ceil(
        np.log2(
            max(
                minimum_pool,
                256,
            )
        )
    )
)

sampler = qmc.Sobol(
    d=6,
    scramble=False,
)

sobol_samples = (
    sampler.random_base2(
        m=power
    )
)

centered = (
    2.0
    * sobol_samples
    - 1.0
)

# Remove the sample closest to zero because
# the nominal action is explicitly candidate 0.

total_norm = np.linalg.norm(
    centered,
    axis=1,
)

centered = centered[
    total_norm
    >
    candidate_params.numerical_epsilon
]

# ----------------------------------------
# Translation
# ----------------------------------------

translation_samples = (
    centered[:, :3]
    *
```

```
    candidate_params.translation_bound
)

# ----------------------------------------
# Rotation
# ----------------------------------------

raw_rotation = (
    centered[:, 3:]
)

raw_norm = np.linalg.norm(
    raw_rotation,
    axis=1,
    keepdims=True,
)

safe_norm = np.maximum(
    raw_norm,
    candidate_params.numerical_epsilon,
)

rotation_axis = (
    raw_rotation
    /
    safe_norm
)

normalized_radius = np.clip(
    raw_norm
    /
    np.sqrt(3.0),
    0.0,
    1.0,
)

rotation_vectors = (
    rotation_axis
    *
    normalized_radius
    *
    candidate_params.rotation_bound
)

pool_size = (
    centered.shape[0]
)

candidate_positions = (
    np.empty(
        (
            pool_size,
            3,
        ),
        dtype=np.float64,
    )
)

candidate_quaternions = (
    np.empty(
```

```
        (
            pool_size,
            4,
        ),
        dtype=np.float64,
    )
)

for index in range(
    pool_size
):

    candidate_positions[index] = (
        nominal_position
        +
        translation_samples[index]
    )

    delta_q = (
        rotation_vector_to_quaternion(
            rotation_vectors[index],
            candidate_params.numerical_epsilon,
        )
    )

    # Local rotational perturbation:
    #
    # q_candidate
    # =
    # q_nominal * q_delta

    candidate_q = (
        quaternion_multiply(
            nominal_quaternion,
            delta_q,
        )
    )

    candidate_quaternions[index] = (
        normalize_quaternion(
            candidate_q,
            candidate_params.numerical_epsilon,
        )
    )

# ----------------------------------------
# Workspace filtering
# ----------------------------------------

if workspace_valid_fn is None:

    valid_mask = np.ones(
        pool_size,
        dtype=bool,
    )

else:

    valid_mask = np.asarray(
        workspace_valid_fn(
```

```
            candidate_positions,
            candidate_quaternions,
        ),
        dtype=bool,
    )

    if (
        valid_mask.shape
        !=
        (pool_size,)
    ):
        raise ValueError(
            "workspace_valid_fn must return "
            "shape (N,)."
        )

valid_positions = (
    candidate_positions[
        valid_mask
    ]
)

valid_quaternions = (
    candidate_quaternions[
        valid_mask
    ]
)

if (
    valid_positions.shape[0]
    <
    target_number
):
    raise RuntimeError(
        "The Sobol pool does not contain enough "
        "workspace-valid candidates. Increase "
        "pool_multiplier or relax workspace bounds."
    )

# ----------------------------------------
# Final candidate set
# ----------------------------------------

positions = np.empty(
    (
        candidate_params.candidate_count,
        3,
    ),
    dtype=np.float64,
)

quaternions = np.empty(
    (
        candidate_params.candidate_count,
        4,
    ),
    dtype=np.float64,
)

# Candidate 0 = nominal action
```

```
    positions[0] = (
        nominal_position
    )

    quaternions[0] = (
        nominal_quaternion
    )

    positions[1:] = (
        valid_positions[
            :target_number
        ]
    )

    quaternions[1:] = (
        valid_quaternions[
            :target_number
        ]
    )

    return (
        positions,
        quaternions,
    )


# 5. Distance to reference safe-action set

def distance_to_safe_set(
    positions,
    quaternions,
    safe_reference_mask,
    params: Parameters,
) -> np.ndarray:
    """
    Compute distance from every candidate to the
    nearest reference-safe candidate.

    Manuscript:

        d_i =
        min_j sqrt(
            ||p_i-p_j||^2 / sigma_t^2
            +
            d_r(q_i,q_j)^2 / sigma_r^2
        )

    Only candidates satisfying safe_reference_mask
    participate in the minimum.
    """

    params.check()

    positions = _float_array(
        positions,
        "positions",
    )

    quaternions = _float_array(
```

```
    quaternions,
    "quaternions",
)

safe_reference_mask = np.asarray(
    safe_reference_mask,
    dtype=bool,
)

if (
    positions.ndim != 2
    or
    positions.shape[1] != 3
):
    raise ValueError(
        "positions must have shape (N, 3)."
    )

n = (
    positions.shape[0]
)

if (
    quaternions.shape
    !=
    (n, 4)
):
    raise ValueError(
        "quaternions must have shape (N, 4)."
    )

if (
    safe_reference_mask.shape
    !=
    (n,)
):
    raise ValueError(
        "safe_reference_mask must have shape (N,)."
    )

safe_indices = np.flatnonzero(
    safe_reference_mask
)

# This function itself does not decide the
# execution mode.
#
# run_frs_pipeline() intercepts this condition
# and returns REJECT before this function is called.

if (
    safe_indices.size
    ==
    0
):
    raise ValueError(
        "The reference safe-action set is empty."
    )

# Pairwise translation differences
```

```
    delta = (
        positions[:, None, :]
        -
        positions[None, :, :]
    )

    translation_sq = np.sum(
        delta * delta,
        axis=2,
    )

    # Pairwise rotational distance

    rotation_distance = (
        quaternion_distance_matrix(
            quaternions,
            params.numerical_epsilon,
        )
    )

    # Dimensionless normalized pose distance

    normalized_distance = np.sqrt(
        translation_sq
        /
        (
            params.translation_scale
            ** 2
        )
        +
        rotation_distance ** 2
        /
        (
            params.rotation_scale
            ** 2
        )
    )

    distance = np.min(
        normalized_distance[
            :,
            safe_indices,
        ],
        axis=1,
    )

    return distance


# 6. Semantic--physical fuzzy action set

def construct_fuzzy_set(
    semantic_score,
    physical_distance,
    params: Parameters,
) -> np.ndarray:
    """
    Construct semantic--physical membership.
```

```
Manuscript:

    rho_i = exp(-beta d_i)

    mu_A(a_i)
    =
    alpha s_i
    +
    (1-alpha) rho_i
"""

params.check()

semantic = _float_array(
    semantic_score,
    "semantic_score",
)

distance = _float_array(
    physical_distance,
    "physical_distance",
)

if semantic.ndim != 1:
    raise ValueError(
        "semantic_score must have shape (N,)."
    )

if distance.ndim != 1:
    raise ValueError(
        "physical_distance must have shape (N,)."
    )

if (
    semantic.shape
    !=
    distance.shape
):
    raise ValueError(
        "semantic_score and physical_distance "
        "must have equal length."
    )

if np.any(
    (semantic < 0.0)
    |
    (semantic > 1.0)
):
    raise ValueError(
        "semantic_score must be normalized "
        "to [0, 1]."
    )

if np.any(
    distance < 0.0
):
    raise ValueError(
        "physical_distance must be non-negative."
    )
```

```
    # rho_i = exp(-beta d_i)

    physical_term = np.exp(
        -params.distance_decay
        *
        distance
    )

    # mu_A = alpha*s + (1-alpha)*rho

    fuzzy_set = (
        params.semantic_weight
        *
        semantic
        +
        (
            1.0
            -
            params.semantic_weight
        )
        *
        physical_term
    )

    return np.clip(
        fuzzy_set,
        0.0,
        1.0,
    )


# 7. Physics-informed fuzzy action relation

def construct_action_relation(
    positions,
    quaternions,
    collision_free,
    params: Parameters,
) -> np.ndarray:
    """
    Construct pairwise fuzzy relation.

    Manuscript:

        R_ij
        =
        K_pose(i,j)
        K_col(i,j)

    where

        K_col(i,j)
        =
        C(a_i) C(a_j)
    """

    params.check()

    positions = _float_array(
        positions,
```

```
    "positions",
)

quaternions = _float_array(
    quaternions,
    "quaternions",
)

collision_free = np.asarray(
    collision_free,
    dtype=bool,
)

if (
    positions.ndim != 2
    or
    positions.shape[1] != 3
):
    raise ValueError(
        "positions must have shape (N, 3)."
    )

n = (
    positions.shape[0]
)

if (
    quaternions.shape
    !=
    (n, 4)
):
    raise ValueError(
        "quaternions must have shape (N, 4)."
    )

if (
    collision_free.shape
    !=
    (n,)
):
    raise ValueError(
        "collision_free must have shape (N,)."
    )

unit_q = (
    _normalize_quaternions(
        quaternions,
        params.numerical_epsilon,
    )
)

# ----------------------------------------
# Translation relation
# ----------------------------------------

delta = (
    positions[:, None, :]
    -
    positions[None, :, :]
)
```

```
translation_sq = np.sum(
    delta * delta,
    axis=2,
)

# ----------------------------------------
# Rotation relation
# ----------------------------------------

rotation_distance = (
    quaternion_distance_matrix(
        unit_q,
        params.numerical_epsilon,
    )
)

# ----------------------------------------
# K_pose
# ----------------------------------------

pose_relation = np.exp(
    -translation_sq
    /
    (
        2.0
        *
        params.translation_scale ** 2
    )
    -
    rotation_distance ** 2
    /
    (
        2.0
        *
        params.rotation_scale ** 2
    )
)

# ----------------------------------------
# K_col = C_i C_j
# ----------------------------------------

collision = (
    collision_free.astype(
        np.float64
    )
)

collision_relation = (
    collision[:, None]
    *
    collision[None, :]
)

# ----------------------------------------
# R = K_pose K_col
# ----------------------------------------

relation = (
```

```
        pose_relation
        *
        collision_relation
    )

    return np.clip(
        relation,
        0.0,
        1.0,
    )


# 8. Fuzzy rough approximations

def fuzzy_rough_regions(
    fuzzy_set,
    relation,
) -> tuple[
    np.ndarray,
    np.ndarray,
    np.ndarray,
]:
    """
    Lower approximation:

        lower_i
        =
        min_j max(
            1-R_ij,
            mu_A(a_j)
        )

    Upper approximation:

        upper_i
        =
        max_j min(
            R_ij,
            mu_A(a_j)
        )

    Boundary:

        boundary_i
        =
        max(
            0,
            upper_i-lower_i
        )
    """

    fuzzy_set = _float_array(
        fuzzy_set,
        "fuzzy_set",
    )

    relation = _float_array(
        relation,
        "relation",
    )
```

```
if fuzzy_set.ndim != 1:
    raise ValueError(
        "fuzzy_set must have shape (N,)."
    )

n = (
    fuzzy_set.shape[0]
)

if (
    relation.shape
    !=
    (n, n)
):
    raise ValueError(
        "relation must have shape (N, N)."
    )

memberships = (
    fuzzy_set[
        None,
        :
    ]
)

# ----------------------------------------
# Lower approximation
# ----------------------------------------

lower = np.min(
    np.maximum(
        1.0 - relation,
        memberships,
    ),
    axis=1,
)

# ----------------------------------------
# Upper approximation
# ----------------------------------------

upper = np.max(
    np.minimum(
        relation,
        memberships,
    ),
    axis=1,
)

lower = np.clip(
    lower,
    0.0,
    1.0,
)

upper = np.clip(
    upper,
    0.0,
    1.0,
```

```
    )

    # ----------------------------------------
    # Boundary
    # ----------------------------------------

    boundary = np.maximum(
        0.0,
        upper - lower,
    )

    return (
        lower,
        upper,
        boundary,
    )


# 9. Hierarchical action selection

def select_action(
    fuzzy_set,
    lower,
    upper,
    boundary,
    collision_free,
    params: Parameters,
) -> tuple[
    Optional[int],
    ExecutionMode,
]:
    """
    Hierarchical decision rule.

    Stage 1:
        U_pos

        collision-free
        AND
        lower > tau_pos

    Stage 2:
        U_bnd

        collision-free
        AND
        boundary > tau_bnd
        AND
        upper > tau_upper

    Stage 3:
        reject
    """

    params.check()

    fuzzy_set = _float_array(
        fuzzy_set,
        "fuzzy_set",
    )
```

```
lower = _float_array(
    lower,
    "lower",
)

upper = _float_array(
    upper,
    "upper",
)

boundary = _float_array(
    boundary,
    "boundary",
)

collision_free = np.asarray(
    collision_free,
    dtype=bool,
)

n = (
    fuzzy_set.shape[0]
)

if (
    lower.shape != (n,)
    or
    upper.shape != (n,)
    or
    boundary.shape != (n,)
    or
    collision_free.shape != (n,)
):
    raise ValueError(
        "All candidate-level arrays "
        "must have shape (N,)."
    )

# ----------------------------------------
# Positive / normal branch
# ----------------------------------------

positive_mask = (
    collision_free
    &
    (
        lower
        >
        params.positive_threshold
    )
)

positive_candidates = np.flatnonzero(
    positive_mask
)

if (
    positive_candidates.size
    >
```

```
    0
):
    local_index = np.argmax(
        fuzzy_set[
            positive_candidates
        ]
    )

    action_index = int(
        positive_candidates[
            local_index
        ]
    )

    return (
        action_index,
        ExecutionMode.NORMAL,
    )

# ----------------------------------------
# Boundary / compliant branch
# ----------------------------------------

boundary_mask = (
    collision_free
    &
    (
        boundary
        >
        params.boundary_threshold
    )
    &
    (
        upper
        >
        params.upper_support_threshold
    )
)

boundary_candidates = np.flatnonzero(
    boundary_mask
)

if (
    boundary_candidates.size
    >
    0
):
    local_index = np.argmax(
        fuzzy_set[
            boundary_candidates
        ]
    )

    action_index = int(
        boundary_candidates[
            local_index
        ]
    )
```

```
        return (
            action_index,
            ExecutionMode.COMPLIANT,
        )

    # ----------------------------------------
    # Rejection branch
    # ----------------------------------------

    return (
        None,
        ExecutionMode.REJECT,
    )


# 10. Core FRS-Grounding function

def frs_grounding(
    positions,
    quaternions,
    semantic_score,
    physical_distance,
    collision_free,
    params: Parameters,
) -> Decision:
    """
    Execute the FRS-Grounding candidate-selection stage.

    Inputs:

        positions:
            (N, 3)

        quaternions:
            (N, 4)

        semantic_score:
            VLA-conditioned candidate scores
            in [0, 1].

        physical_distance:
            normalized safe-set distances.

        collision_free:
            Boolean C(a_i).

    Returns:

        Decision
    """

    params.check()

    positions = _float_array(
        positions,
        "positions",
    )

    quaternions = _float_array(
        quaternions,
```

```
    "quaternions",
)

semantic_score = _float_array(
    semantic_score,
    "semantic_score",
)

physical_distance = _float_array(
    physical_distance,
    "physical_distance",
)

collision_free = np.asarray(
    collision_free,
    dtype=bool,
)

if (
    positions.ndim != 2
    or
    positions.shape[1] != 3
):
    raise ValueError(
        "positions must have shape (N, 3)."
    )

n = (
    positions.shape[0]
)

if (
    quaternions.shape
    !=
    (n, 4)
):
    raise ValueError(
        "quaternions must have shape (N, 4)."
    )

if (
    semantic_score.shape
    !=
    (n,)
):
    raise ValueError(
        "semantic_score must have shape (N,)."
    )

if (
    physical_distance.shape
    !=
    (n,)
):
    raise ValueError(
        "physical_distance must have shape (N,)."
    )

if (
    collision_free.shape
```

```
    !=
    (n,)
):
    raise ValueError(
        "collision_free must have shape (N,)."
    )

# ----------------------------------------
# Fuzzy action set
# ----------------------------------------

fuzzy_set = construct_fuzzy_set(
    semantic_score=semantic_score,
    physical_distance=physical_distance,
    params=params,
)

# ----------------------------------------
# Pairwise relation
# ----------------------------------------

relation = construct_action_relation(
    positions=positions,
    quaternions=quaternions,
    collision_free=collision_free,
    params=params,
)

# ----------------------------------------
# Fuzzy rough regions
# ----------------------------------------

lower, upper, boundary = (
    fuzzy_rough_regions(
        fuzzy_set=fuzzy_set,
        relation=relation,
    )
)

# ----------------------------------------
# Hierarchical decision
# ----------------------------------------

action_index, mode = (
    select_action(
        fuzzy_set=fuzzy_set,
        lower=lower,
        upper=upper,
        boundary=boundary,
        collision_free=collision_free,
        params=params,
    )
)

return Decision(
    action_index=action_index,
    mode=mode,
    fuzzy_set=fuzzy_set,
    relation=relation,
    lower=lower,
```

```
        upper=upper,
        boundary=boundary,
    )


# 11. Empty-safe-set rejection

def _reject_decision(
    candidate_count: int,
) -> Decision:
    """
    Construct a reject result when A_safe is empty.

    No safe-set distance is evaluated because

        A_safe = empty

    makes the minimum-distance expression undefined.
    """

    return Decision(
        action_index=None,

        mode=ExecutionMode.REJECT,

        fuzzy_set=np.zeros(
            candidate_count,
            dtype=np.float64,
        ),

        relation=np.zeros(
            (
                candidate_count,
                candidate_count,
            ),
            dtype=np.float64,
        ),

        lower=np.zeros(
            candidate_count,
            dtype=np.float64,
        ),

        upper=np.zeros(
            candidate_count,
            dtype=np.float64,
        ),

        boundary=np.zeros(
            candidate_count,
            dtype=np.float64,
        ),
    )


# 12. Complete FRS pipeline

def run_frs_pipeline(
    nominal_position,
    nominal_quaternion,
```

```
    semantic_cosine_fn:
        Callable[
            [
                np.ndarray,
                np.ndarray,
            ],
            np.ndarray,
        ],

    safe_reference_fn:
        Callable[
            [
                np.ndarray,
                np.ndarray,
            ],
            np.ndarray,
        ],

    collision_check_fn:
        Callable[
            [
                np.ndarray,
                np.ndarray,
            ],
            np.ndarray,
        ],

    workspace_valid_fn:
        Optional[
            Callable[
                [
                    np.ndarray,
                    np.ndarray,
                ],
                np.ndarray,
            ]
        ] = None,

    params:
        Parameters
        =
        Parameters(),

    candidate_params:
        CandidateGenerationParameters
        =
        CandidateGenerationParameters(),

) -> Decision:
    """
    Complete interface between environment-dependent modules
    and the model-independent FRS-Grounding stage.

    External callbacks
    ------------------

    semantic_cosine_fn:
        Implements VLA-conditioned candidate scoring.
```

```
    Input:
        positions:    (N, 3)
        quaternions:  (N, 4)

    Output:
        raw candidate cosine similarities
        with shape (N,) and values in [-1, 1].

    These values are normalized internally by

        s_i = clip(
            (c_i + 1) / 2,
            0,
            1
        ).

safe_reference_fn:
    Environment-specific safe-reference evaluation.

    A candidate should be marked True only when

        C(a_i) = 1

    and

        clearance_i >= 0.02 m.

    Output shape:
        (N,)

collision_check_fn:
    Nominal-pose collision checking.

    Returns:

        C(a_i)

    as a Boolean array of shape (N,).

workspace_valid_fn:
    Optional workspace-validity check used during
    candidate generation.

Once these candidate-level quantities are supplied,
all fuzzy fusion, relation construction, fuzzy rough
approximation, boundary computation, and hierarchical
action selection are performed here.
"""

params.check()

candidate_params.check()

# ----------------------------------------
# Candidate generation
# ----------------------------------------

positions, quaternions = (
    generate_sobol_candidates(
        nominal_position=nominal_position,
```

```
        nominal_quaternion=nominal_quaternion,
        candidate_params=candidate_params,
        workspace_valid_fn=workspace_valid_fn,
    )
)

n = (
    positions.shape[0]
)

# ----------------------------------------
# VLA-conditioned candidate scoring
# ----------------------------------------

raw_cosine = _float_array(
    semantic_cosine_fn(
        positions,
        quaternions,
    ),
    "raw_cosine",
)

if (
    raw_cosine.shape
    !=
    (n,)
):
    raise ValueError(
        "semantic_cosine_fn must return "
        "shape (N,)."
    )

semantic_score = (
    normalize_cosine_scores(
        raw_cosine
    )
)

# ----------------------------------------
# Nominal-pose collision checking
# ----------------------------------------

collision_free = np.asarray(
    collision_check_fn(
        positions,
        quaternions,
    ),
    dtype=bool,
)

if (
    collision_free.shape
    !=
    (n,)
):
    raise ValueError(
        "collision_check_fn must return "
        "shape (N,)."
    )
```

```
# ----------------------------------------
# Reference safe-action set
# ----------------------------------------

safe_reference_mask = np.asarray(
    safe_reference_fn(
        positions,
        quaternions,
    ),
    dtype=bool,
)

if (
    safe_reference_mask.shape
    !=
    (n,)
):
    raise ValueError(
        "safe_reference_fn must return "
        "shape (N,)."
    )

# ----------------------------------------
# Empty safe-action set
#
# A_safe = empty
#
# -> reject
# ----------------------------------------

if not np.any(
    safe_reference_mask
):
    return _reject_decision(
        n
    )

# Every reference-safe candidate must
# necessarily be collision-free.

if np.any(
    safe_reference_mask
    &
    (~collision_free)
):
    raise ValueError(
        "Reference-safe actions must be "
        "collision-free."
    )

# ----------------------------------------
# Safe-set distance
# ----------------------------------------

physical_distance = (
    distance_to_safe_set(
        positions=positions,
        quaternions=quaternions,
        safe_reference_mask=safe_reference_mask,
        params=params,
```

```
        )
    )

    # ----------------------------------------
    # FRS decision
    # ----------------------------------------

    return frs_grounding(
        positions=positions,
        quaternions=quaternions,
        semantic_score=semantic_score,
        physical_distance=physical_distance,
        collision_free=collision_free,
        params=params,
    )


# 13. Threshold re-evaluation

def reselect_with_thresholds(
    decision: Decision,
    collision_free,

    positive_threshold: float,

    boundary_threshold: float,

    upper_support_threshold:
        float
        =
        1e-8,

) -> tuple[
    Optional[int],
    ExecutionMode,
]:
    """
    Re-evaluate only the hierarchical branch selection.

    Candidate generation, fuzzy-set construction,
    pairwise relations, and fuzzy rough approximations
    are not recomputed.

    This utility corresponds to the threshold-response
    analysis in the manuscript.
    """

    params = Parameters(
        semantic_weight=0.70,

        distance_decay=2.0,

        translation_scale=0.25,

        rotation_scale=0.25,

        positive_threshold=(
            positive_threshold
        ),
```

```
    boundary_threshold=(
        boundary_threshold
    ),

    upper_support_threshold=(
        upper_support_threshold
    ),

    numerical_epsilon=1e-8,
)

return select_action(
    fuzzy_set=decision.fuzzy_set,
    lower=decision.lower,
    upper=decision.upper,
    boundary=decision.boundary,
    collision_free=collision_free,
    params=params,
)
```