JointAct: Constrained Joint Plans for LLM Agent Coordination

Lingfeng Zhou[1] Shuaixing Chen[3] Jin Gao[1] Dequan Wang[1,2*]
[1]Shanghai Jiao Tong University [2]Shanghai Innovation Institute [3]Carnegie Mellon University

Abstract

When agents share physical resources, local proposals need not form a compatible joint action. JointAct constructs complete joint plans under encoded resource and precedence constraints. An LLM selects a listed candidate identifier without rewriting its assignments; installation validation and lifetime checks govern controller use. We compare JointAct, independent proposal execution, and a seeded resource-priority coordinator in 540 runs: 30 seed-indexed repetitions per system on each of six traffic fixtures with fixed initial vehicle configurations. Success requires every initial vehicle to complete the respective endpoint under the implemented adverse-event exclusions. JointAct records 180/180 interaction successes and 123/180 destination successes, compared with 119/180 and 100/180 for Seeded Priority and 83/180 on both endpoints for independent execution. All 237 model selections preserve the candidate's assignment in the installed plan. A matched-proposal crossing case exposes different joint assignment rules. Analysis of all 30 Hybrid multi-resource runs connects initial assignments to subsequent coordination, shared recovery, and finite-horizon outcomes; a rolling case distinguishes interaction completion from full arrival. The comparisons assess complete systems whose construction and selection rules differ. Together, the records show how checked joint assignments become control inputs, while compatible passage can still leave unfinished routes at the configured horizon.

CCS Concepts

• **Computing methodologies** → **Multi-agent systems**; **Cooperation and coordination**.

Keywords

multi-agent coordination, joint action plans, LLM agents, constrained planning, traffic simulation

1 Introduction

At a shared crossing, two vehicles may each propose to advance, yet both cannot occupy the same region at once. A nearby group member may not claim that region but still needs an assignment. Once one vehicle proceeds, the remaining members may need a new arrangement before all vehicles can finish their routes.

Local proposals describe individual suggestions; controllers need a compatible assignment for all relevant members. We represent each alternative as a *joint plan*: one action assignment for every current group member. Membership determines coverage, while resource relationships determine admissible combinations. The choice is a mapping over the group, including non-conflicting members, rather than an independently selected action for each vehicle.

Joint plans are established subjects in multi-agent research [4, 10]. Robotic and driving systems already connect joint plans and negotiated intentions to external checks, candidate evaluation, and downstream planning [15, 23, 27]. We focus on the boundary between constructing an assignment, choosing it, and permitting its use. The selected object must preserve the checked actions; selection alone does not establish applicability after membership, window, or lifetime changes.

We present JointAct, a coordination service that makes a complete, constraint-checked joint plan the unit of selection. The service receives local proposals and structured conflict facts, then constructs a bounded list of assignments under its encoded resource rules. An LLM selects a listed candidate ID without rewriting its action mapping. Retrieving that ID preserves the entire assignment. Revalidation precedes installation, and separate controller-facing conditions determine whether the installed plan is current. Figure 1 illustrates this choice; Figure 2 connects construction, selection, and execution.

We compare JointAct (Hybrid) with independent proposal execution and a seeded resource-priority coordinator on six MetaDrive fixtures with fixed initial vehicle configurations, using 30 seed-indexed runs per system and fixture [21]. Task success requires completion by all initial vehicles without recorded adverse events. In crossing, Seeded Priority records fewer collisions but also fewer task successes than Independent; Hybrid attains both endpoints with zero observed collisions. Candidate construction, filtering, assignment, and selection differ between systems: this comparison assesses complete designs. Recorded crossing and multi-resource paths show how joint assignments determine controller inputs and accompany different subsequent coordination and recovery paths. A rolling endpoint provides a further finite-horizon contrast.

Our contributions are:

- **Constructing complete joint plans.** We specify how current membership, overlapping resources, precedence, and recorded resource holders constrain candidate assignments. Compatible proposals from non-claimants remain part of each plan, so checking covers the whole group rather than only the contending vehicles.
- **Connecting restricted choice to execution conditions.** The model returns a listed ID without rewriting assignments. Retrieval and installation checks preserve the selected mapping; group, window, and lifetime conditions delimit its use. Saved records verify mapping correspondence in all 237 model-selection windows.
- **Comparing systems and their outcome patterns.** The six-scenario, 540-run evaluation compares three complete systems. A matched-proposal crossing case exposes differences in assigned actions; all 30 Hybrid multi-resource records relate initial assignments to subsequent coordination, recovery, and finite-horizon outcomes.

*Corresponding author.

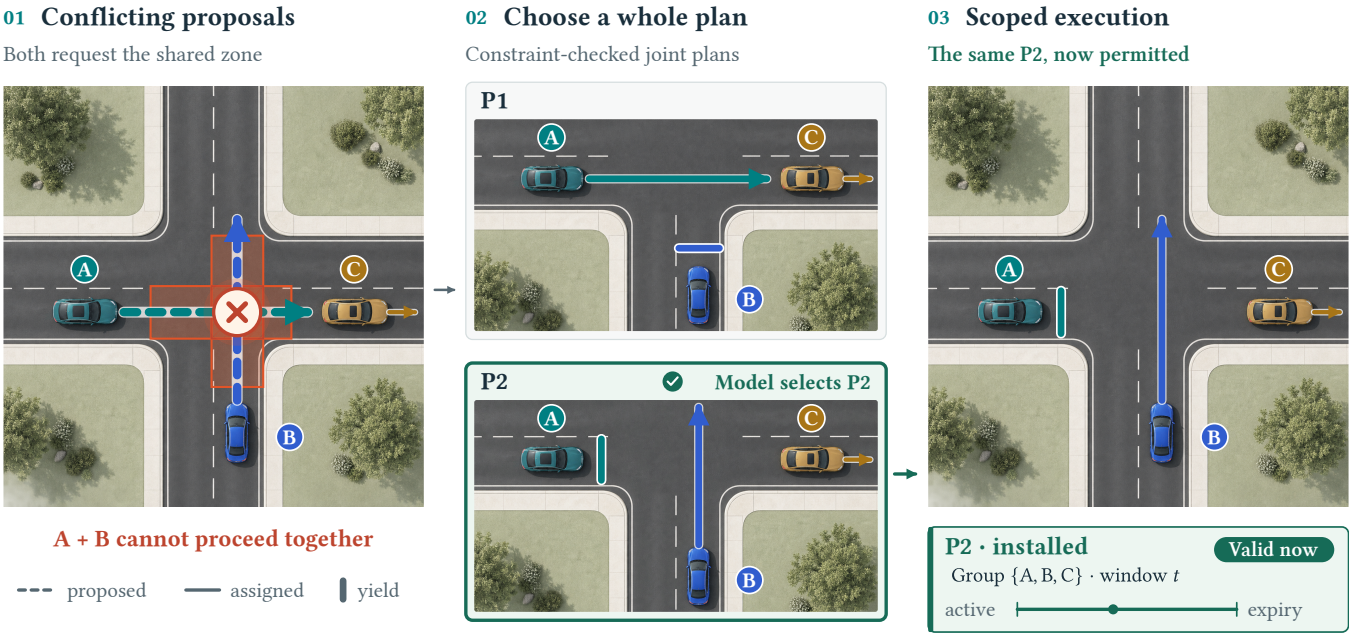


Figure 1: Local proposals conflict; the choice is a complete joint plan. In this schematic crossing, A and B propose incompatible progress; C belongs to the group without claiming the shared zone. JointAct constructs complete candidates checked against encoded joint constraints. In Hybrid, the model selects a listed ID, illustrated by P2, without rewriting the joint assignment. After revalidation and installation, a separate permit governs execution for the current group, window, and lifetime. Each candidate covers A, B, and C. Arrows and yield bars indicate intended roles at the same state, not measured trajectories.

2 Related Work

Driving policies and coupled decisions. Data-driven driving integrates perception, prediction, and planning through sensor fusion or structured representations [14, 19, 30]. Language-based systems connect driving to reasoning, instructions, reflection, and control prediction [32, 35, 40, 42]. These interfaces inform individual behavior; coordinated driving also models dependencies through communication and reasoning about other vehicles [13, 20]. Learning-based approaches use centralized critics or joint value factorization [25, 31], while LDPD and CoopReflect use checked teacher actions or multi-agent debriefing to improve policies [5, 24]. Our focus is the inference-time object selected from proposals by fixed components, rather than training a new driving policy.

Complete plans and model participation. RoCo generates per-robot sub-tasks, checks task, kinematic, and collision constraints, and invokes joint motion planning [27]. CoLMDriver connects dynamic and historical groups to intention negotiation, evaluation, and waypoint planning [23]; CoLMIN generates multiple complete multi-vehicle candidates and uses waypoint and consensus scores to guide selection and reflection [15]. Complete plans, multiple candidates, and external checks therefore already coexist. Feedback can revise a proposed group plan before downstream execution. The relevant distinction is where the model may change the object being checked, rather than whether plans are checked.

Constraint-directed repair and shared commitments provide further forms of cooperation [17, 43]. In JointAct, the service constructs and checks joint assignments before model selection. The model returns a listed ID without revising the assignment, and retrieval preserves that checked object. Table 5 in the appendix gives the extended comparison.

Checks and conditions of use. Teamwork theories make joint plans and commitments explicit [4, 10, 38]; coordination graphs and distributed constraint optimization represent coupled choices [11, 28]. Intersection reservations distinguish requests from granted access [8], and SayCan grounds skill selection in learned value estimates [16]. LLM-Modulo uses external critics to check generated plans and guide regeneration [12]. Code as Policies generates control programs, while ProgPrompt generates task programs with execution assertions [22, 36]. The symbolic planner PDDLStream combines planning with external sampling procedures [9]. These interfaces expose program construction or planning operations. Selecting a listed joint-plan ID instead fixes the assignment before model choice: the selector can choose a different checked combination, but cannot edit the actions inside it. Shielding, SOTER, and AgentSpec separately enforce action or runtime conditions [1, 7, 39]. JointAct combines selection from prechecked joint assignments with separate membership, window, and lifetime conditions for controller use. Its checks concern the encoded resource model; they do not establish physical feasibility or inherit guarantees from these other frameworks.

3 The Coordination Problem

Let H_t be the members of a coordination group in planning window t. Each member i supplies a non-binding local proposal q_i^t: a high-level maneuver, an acceleration interval, and local and route context. The complete, ordered collection forms the proposal bundle B_t. A conflict adapter supplies X_t, describing shared resources, their claimants and overlaps, fixture occupation metadata, previously recorded resource holders, and registered route precedence. Group membership specifies who must receive an assignment; resource dependencies specify which assignments may coexist.

A joint plan maps each current member to one maneuver and interval:

$$P : H_t \to \mathcal{U} \times \mathcal{B}, \qquad \mathcal{B} = \{[b^-, b^+] \mid b^- \le b^+\}. \tag{1}$$

The mapping is total and single-valued: every member receives exactly one assignment. Here $\mathcal{U} = \{\mathrm{AC}, \mathrm{DC}, \mathrm{KS}, \mathrm{LCL}, \mathrm{LCR}, \mathrm{SB}\}$ is the controller vocabulary, and the interval is measured in $\mathrm{m/s^2}$ (Appendix A). The labels *proceed*, *yield*, and *keep* denote illustrative roles represented through maneuvers and intervals.

Multi-agent path finding instead specifies a path for each agent, with joint validity defined by the chosen conflict model [37]. Cooperative space-time planning [34] and conflict-based search [33] operate on such path-planning objects. Here P assigns high-level controller actions for a coordination window; it is neither a joint trajectory nor a direct steering command. The resource checks therefore concern this action model, rather than the path constraints or optimality objectives of those planners.

Figure 1 illustrates why the group and resource scopes differ. A and B claim conflicting shared space, while C belongs to the group without claiming it. Alternatives may reverse the roles of A and B, but both must retain an assignment for C. Joint checking therefore concerns the complete mapping, including the recorded resource-holder constraints. The recorded holder constrains who may receive a proceeding role; it is not a continuous physical-occupancy measurement.

The coordination service takes H_t, B_t, and X_t and constructs and selects a joint plan, then separately decides whether to install it with a defined execution scope. Three questions organize the method: which complete combinations satisfy the encoded constraints, which retained candidate is selected, and whether that selected assignment currently has permission to govern controllers. We call a candidate *admissible* relative to those encoded constraints. Execution permission binds the installed plan to the relevant members, decision window, identity, and lifetime. Coordination and installation check these scope conditions; the control loop checks the validity interval. Neither substitutes for a physical-feasibility check. Section 4 develops construction, selection, and the transition from a selected object to controller use.

4 JointAct: Construct, Select, and Execute

Figure 2 follows JointAct's Hybrid path: construct whole-group alternatives, select one unchanged, and install it for a limited execution scope. Table 2 locates this path within the three evaluated systems.

4.1 Constructing the candidate plans

Members and resource dependencies. The simulator hosts the service; individual controllers execute assignments. Distance hysteresis connects ungrouped vehicles at 30 m, merges existing groups at 20 m, and retains within-group connections up to 40 m. Connected components with at least two vehicles define groups at each planning window. A group enters coordinated planning only when the adapter reports an active resource conflict.

The service collects one typed proposal per member into B_t. Group membership determines *coverage*; resource claims determine *dependencies*. A, B, and C in Figure 1 all need assignments even if only A and B claim shared space. The adapter represents overlap edges, claimants, prior holders, and route predecessors. These inputs constrain the combinations of member actions, whereas distance-based grouping determines which members must be included.

From resource components to complete assignments. The constructor connects active resources through overlap edges. For each component it lists, in agent-ID order, claimants compatible with prior holders on their claimed resources. Hybrid excludes a claimant with a registered predecessor still claiming an active resource. If this eligible list is nonempty, the component must select one claimant; otherwise its only choice is all-yield. This component-progress rule is a construction restriction, not a proof of continuing physical progress.

The constructor enumerates Cartesian products of these component choices. Each combination starts from every member's proposal. All active claimants receive $\mathrm{DC}[-1.2, -0.2]$; selected claimants receive $\mathrm{AC}[0.2, 1.0]$, in $\mathrm{m/s^2}$. Non-claimants retain their proposed maneuver *and* interval. A selected claimant must hold every active resource it claims; a combination that cannot satisfy this is rejected. Thus independently chosen component roles still have to form one consistent group mapping.

Whole-assignment validation checks coverage, at most one proceeding claimant per resource, holder continuity, conflicts between overlapping resources, and Hybrid's registered precedence. Here AC, LCL, and LCR count as proceeding maneuvers. Numeric intervals describe controller inputs; the validator does not perform a per-candidate physical rollout. In the A/B/C example, omitting C fails coverage, whereas assigning A and B to proceed together fails the resource check even if each maneuver has a valid type. C's illustrative "keep" role means preserving its proposal, not adding a controller symbol.

Budgets and ordering. With a retained-list cap $K = 32$, construction examines at most $\max(256, 16K) = 512$ component combinations. Valid mappings are deduplicated by candidate ID, a digest of assignments, holders, and proceeding members. Let E_t^{512} be these examined combinations and V their joint check. The retained list is

$$\begin{aligned} \widehat{C}_t &= \{P \in E_t^{512} : \mathsf{V}(P)\}, \\ C_t &= \mathrm{First}_{32}\Big(\mathrm{Sort}_{(-n_p, -n_q, \mathrm{ID})}\, \widehat{C}_t\Big). \end{aligned} \tag{2}$$

where n_p counts proceeding members and n_q counts maneuvers matching the original proposals. The preservation count does not require interval equality. Sorting therefore favors more proceeding members, then maneuver preservation, with candidate ID breaking

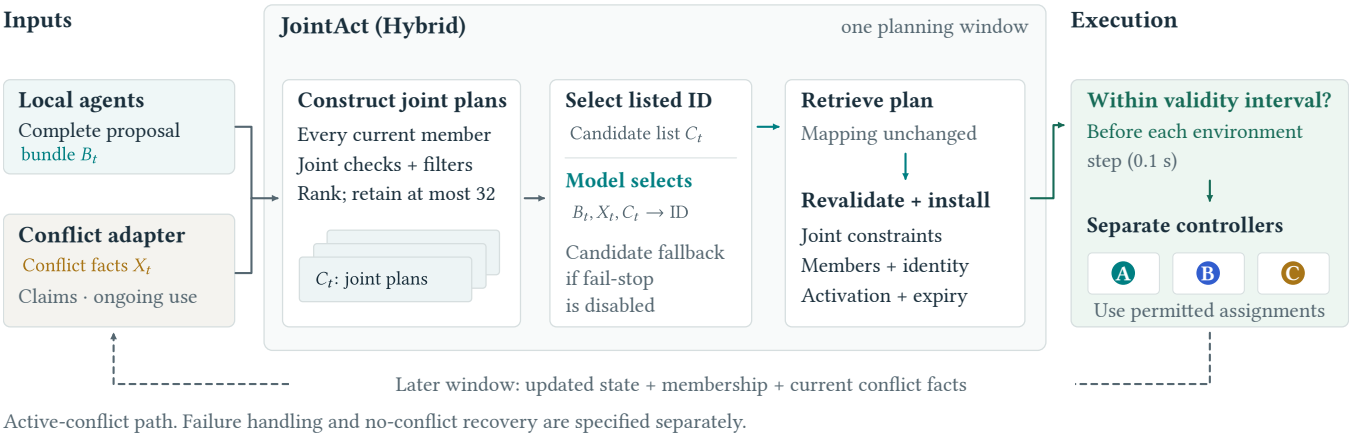


Figure 2: JointAct's Hybrid coordination path. Construction checks whole assignments before the model selects a listed ID. Retrieval preserves the mapping; revalidation precedes installation for the current members and window. The control loop checks activation and expiry each environment step. Candidate fallback is an internal alternative when failure-stop is disabled; the evaluated Seeded Priority baseline has a separate construction rule.

Table 1: JointAct's construction and execution checks. Resource checks concern the encoded action model; lifecycle and per-step checks have different scopes.

Question	System rule
Who is covered?	Exactly one typed assignment per current member, including non-claimants such as C.
Which moves coexist?	Check resource holders, exclusive proceeding roles, overlap conflicts, and Hybrid's route precedence.
What persists?	Retain recorded holders within one resource episode and membership key; changing that key starts a separate lookup.
Who proceeds?	In Hybrid construction, choose one holder/precedence-eligible claimant per component if possible; otherwise all yield.
When is use current?	Check identity, lineage, group/window, and lifetime at coordination/install; check validity interval each environment step.

ties. The first retained entry is marked as the internal *candidate fallback*. Enumeration and list caps are distinct: the model sees at most 32 of the checked alternatives within the 512-combination search, not every admissible assignment. If none survives, construction raises an error rather than relaxing a constraint. Table 1 separates construction from permission checks.

4.2 Selecting an intact assignment

In the schematic A/B/C state, P_1 = (proceed, yield, keep) and P_2 = (yield, proceed, keep) are two complete choices. Taking A's action from P_1 and B's from P_2 recreates the conflict; that mixture is not selectable.

Hybrid receives proposals, resource facts, and the full candidate mappings. Its prompt orders preferences as previous-holder continuity, proposal preservation, then useful progress with low yielding cost. Auxiliary `proposal_utility` weights are hand-written rules based on maneuver types and context/route keywords, not learned or measured traffic utilities. The fallback flag is omitted from the model input, but the deterministic ranking is retained, with the internal fallback listed first. Candidate-order effects are not isolated. Its only permitted response is a JSON object with `candidate_id`; the parser rejects extra fields and unlisted IDs. Retrieval preserves the selected assignment. No rationale, revised action, vote, or new candidate enters this interface.

Table 2 contrasts the resulting action paths. In particular, a Seeded Priority holder normally keeps its proposal: holding a resource need not produce an accelerating action. Hybrid instead assigns explicit AC/DC intervals to active claimants. Its candidate fallback is the first complete mapping in its own ranked list, not the evaluated Seeded Priority rule. The recorded crossing example in Section 5 shows how these assignment rules transform the same proposals into different control inputs.

4.3 Revalidation and execution permission

Selection and installation are separate. The runtime binds the selected mapping to a new plan ID, group, membership version, window, proposal lineage, and validity interval. It validates these fields and reapplies the resource validator before installation. The resource validator is shared with construction; this is another invocation, not an independently implemented verifier. Candidate ID identifies a listed choice, whereas runtime plan ID identifies its particular installation.

With failure-stop disabled, an invalid selector response can use candidate fallback. Final runs enable `fail_on_model_failure`: model failure or semantic rejection stops that run attempt. Generic construction or structural rejection can instead return proposals without a group plan. Thus no group permission does not imply fail-safe control; Appendix A distinguishes these branches.

Table 2: Components of the three evaluated systems. The rows distinguish active-conflict planning paths. All three use the same low-level controller and the same recovery rule outside active group conflict; their changed assignments can lead to different later states and planning windows. Intervals are in m/s^2.

Component	Independent	Seeded Priority	JointAct (Hybrid)
Local proposals	Same proposal model alias, prompt, and typed local-view interface during active-conflict planning.		
Group and resources	Same distance-based grouping and resource adapter; prior-holder state is supplied only to the two coordinated systems.		
Action assignment	Use each member's proposed maneuver and interval.	Normally preserve each resource holder's proposal and assign DC to other claimants; reconcile competing overlap holders with explicit AC/DC.	Enumerate complete mappings; selected claimants receive AC $[0.2, 1]$, others DC $[-1.2, -0.2]$. Non-claimants retain their proposals.
Joint conditions	No group-plan validation.	Coverage, holder continuity, exclusive proceeding roles, and overlap checks.	Same resource validator with precedence enabled; construction also requires an eligible proceeding claimant per overlap component when possible.
Choice rule	No candidate selection.	Retain an unreleased holder, else use a seeded draw keyed by resource, episode, and claimants. No candidate list.	Model returns one ID from a deterministically ranked list. The first candidate is the internal fallback; only its flag is omitted from model input.
Execution scope	Direct proposal assignments; no installed group plan.	Validate and install a complete assignment with group/window identity and lifetime. Check activation and expiry each environment step.	

Algorithm 1 JointAct Hybrid's successful active-conflict window. Final runs stop on model or semantic failure; generic failure branches are distinguished in Appendix A.

Require: current group H_t, resource facts X_t, cap $K = 32$
collect and validate one proposal per member in B_t
for all active overlap components in X_t **do**
  list holder-compatible claimants with no active predecessor
  choose one listed claimant; allow all-yield only if list empty
$\widehat{C}_t \leftarrow \varnothing$
for all first $\max(256, 16K)$ products of component choices **do**
  initialize P with all members' proposal assignments
  set active claimants to DC$[-1.2, -0.2]$
  set selected claimants to AC$[0.2, 1.0]$
  require selected claimants to hold all resources they claim
  check coverage, holder, overlap, and precedence constraints
  if valid: add P to $\widehat{C}_t$, deduplicated by candidate ID
sort by $(-n_p, -n_q, \text{ID})$; retain first K candidates
if no candidate remains: construction fails
mark first entry as internal candidate fallback
omit fallback flag from model input; preserve ranked order
request and parse one listed candidate ID from Hybrid
retrieve mapping unchanged; attach runtime plan identity
recheck structure, lineage, scope, lifetime, and resource rules
install validated assignments for all current members
check activation/expiry before each environment step

The execution clock and controller. Installation and assignment retrieval check current group/window and lifetime; group changes retire superseded permission at planning updates. The online loop then checks *valid-from/valid-until* before each environment step. One environment step is 0.1 s, containing five 0.02 s physics substeps. It does not repeat all member, resource, or dynamics checks at each substep. Planning waits synchronously for the model before simulation resumes, so this setup does not measure asynchronous latency tolerance.

The shared controller used with MetaDrive [21] grounds each interval $[b^-, b^+]$. For AC it clips $\max(0, b^+)$ to that interval; for DC it clips $\min(0, b^-)$. SB requests b^-. Other maneuvers clip a cruise-compensation value of 0.2 m/s^2. Speed overrides, acceleration limits, and a jerk bound then modify the request. LCL/LCR also bind a target-lane change to the current plan. These are controller operations, not further candidate-selection decisions.

When no active group conflict remains, the runner retires group permission and uses shared recovery. Without current group permission, active conflict, or a proximity block, a vehicle below 7.75 m/s receives AC toward the 8 m/s target; otherwise it receives KS. This recovery path is distinct from both a model-selected plan and candidate fallback.

Scope updates and holder continuity. A recorded holder can persist across windows with the same resource episode, group ID, membership version, and member set. When membership changes, lookup uses a new key: the old record may remain stored, but it is not automatically retrieved or migrated. Establishing the same holder under that new key is a new record, not continuity across keys. Release also follows inactive conflict or fewer than two claimants, so a release event does not establish measured physical clearance. Appendix A gives the key and release conditions. This scope limit matters when interpreting the later-window records in Section 5.

5 Evaluation and Analysis

5.1 Setup and comparison logic

The MetaDrive evaluation [21] comprises six predefined traffic fixtures (Table 3), with 30 seed-indexed runs per fixture and system: 180 episodes per system, 540 overall. Within each fixture, all 90 first-window state digests record identical initial positions, speeds, and lanes. Seeds therefore index repeated runs, not distinct traffic initializations. All 90 low-coupling runs bypass both model calls and active coordination, providing a no-conflict control check. Despite

Table 3: Fixtures, coordination coverage and planning intervals in simulated seconds. Window counts are configuration limits, not a difficulty ranking. Horizons and populations appear in Appendix Table 8.

Fixture	Coordination coverage	Interval (s)	Max. windows
Low coupling	No active conflict or model calls	25.0	2
Single-resource merge	Order two resource claimants	25.0	2
Crossing	Mutually exclusive passage	25.0	2
Rolling decisions	Successive plans; changing membership	22.5	4
Dynamic merge/split	Observed: two-member, version-1 plans	17.5	4
Overlapping multi-resource	Joint constraints on connected resources	16.7	3

its historical name, dynamic merge/split records only two-member, version-1 installations; rolling and overlapping resources contain the observed membership updates.

Table 2 details this complete-system comparison: construction, filtering, assignment and selection vary together. Common fixture–seed pairs organize comparisons, but subsequent proposal inputs follow each system's own rollout, as in closed-loop driving evaluations [6, 18]. Earlier development runs are excluded.

The recorded aliases are `deepseek-v4-flash` for proposals (1,024 output-token cap) and `kimi-k2.5` for selection (4,096 cap). Calls finish synchronously before simulation continues. Configured planning intervals appear in Table 3; installed-plan validity intervals are checked every 0.1 s environment step, which comprises five 0.02 s physics substeps. The 500/700/900-step horizons are 50/70/90 s. Appendix B gives the remaining settings and prompts.

Outcome definitions. Let I and D denote interaction and destination completion, and C, O, and K the recorded vehicle-collision, off-road, and deadlock flags. The historical success predicates are

$$S_I = I \wedge \neg C \wedge \neg O \wedge \neg K, \qquad S_D = D \wedge \neg C \wedge \neg O \wedge \neg K. \quad (3)$$

Each vehicle present at reset accumulates interaction completion on its first qualifying environment step: crossing requires entry within 6 m of the conflict point followed by exit beyond 10 m; other fixtures require x beyond the bottleneck reference by 10 m. Destination arrival also establishes interaction completion. I and D require every initial vehicle to complete the respective endpoint, with D using `arrive_dest`; their times are the latest members' first qualifying times. Coordination membership changes do not change this evaluation population.

Collision and off-road flags accumulate throughout the run, including after interaction completion. K requires I to be false and every vehicle's initial-to-final displacement to be below 1 m. Thus $K \Rightarrow \neg I$ and $D \Rightarrow I$: K is redundant in Eq. 3, not an independent general deadlock test. Success counts exclude collision/off-road runs, and destination success implies interaction success. Event counts count runs, not vehicles or impacts.

5.2 System-level outcomes

Table 4 reports all scenario counts and four aggregate outcomes. Relative to Seeded Priority, Hybrid records 61 additional interaction successes and 23 additional destination successes. Both coordinated systems have zero vehicle-collision runs, whereas Independent has 88. Their off-road totals also differ. Raw interaction completions are 92, 123, and 180 for Independent, Seeded Priority, and Hybrid; all 13 completion–success discrepancies contain off-road events. In dynamic merge/split, both coordinated systems complete the interaction in 30/30 runs: the four-success gap arises from Seeded Priority's four off-road runs, not additional clearances.

Crossing demonstrates why the outcome dimensions must be read together. Seeded Priority reduces collisions from Independent's seven to zero, yet lowers both success counts from 23 to 14. Hybrid reaches 30 on both endpoints with zero collisions. Coordination therefore does not uniformly improve every outcome; the assignments produced within its constraints matter. The matched-proposal record below shows the concrete assignment difference behind one such comparison.

The distribution of gains also depends on the endpoint. Rolling and overlapping resources account for 41 of the 61 additional interaction successes but only three of the 23 additional destination successes; crossing and dynamic account for the remaining 20 destination differences. These are decompositions of scenario counts, not flows of the same episodes between endpoints. Low coupling and single-resource merge show no success-count difference between the coordinated systems. Their ceiling totals establish no observed benefit in these fixtures, rather than general policy equivalence.

5.3 Recorded assignments, recovery, and endpoints

Crossing and rolling use the first archived seed, 20261003. The multi-resource analysis includes all 30 Hybrid runs, with the first archived run in each first-proceeding-member group as its representative. Companion evidence indexes the source cells, full identifiers and samples.

Crossing: matched proposals, different assignments. Only 2/30 paired first windows have identical complete proposal digests; 21/30 match when comparing maneuver and acceleration interval alone. The first archived seed belongs to both sets. Its proposals are agent0 `AC` $[0, 1]$ and agent1 `KS` $[-1, 0]$, in m/s^2. Seeded Priority retains holder agent1's `KS` proposal and assigns agent0 `DC` $[-1.2, -0.2]$, repeating this arrangement in its second window. Both terminal speeds approach zero, and neither endpoint completes by 50 s.

Hybrid constructs explicit proceed/yield candidates, selects agent0 `AC` $[0.2, 1]$ with agent1 `DC` $[-1.2, -0.2]$, and installs that mapping (Figure 3). At 25.0 s, before step 251, agent1 enters shared recovery and receives `AC` without an LLM call. Interaction completion follows at 32.6 s and all-member arrival at 40.4 s. The selected candidate is Hybrid's own fallback, also selected in 27/30 crossing runs. The comparison reveals the distinction between preserving a holder's local action and assigning explicit joint proceed/yield roles; it does not demonstrate a model outperforming the same list's first item. Independent's same-seed run has different proposals and collides at 5 s.

Table 4: Outcomes verified against saved episode records. Each scenario–system cell has 30 runs; aggregate rows have 180 per system. Both success metrics use Eq. 3, excluding collision, off-road, and deadlock. The two successes can coincide; collision and off-road can overlap. Off-road totals appear in the final row.

	Independent			Seeded Priority			Hybrid		
Scenario	Interaction success	Destination success	Collision episodes	Interaction success	Destination success	Collision episodes	Interaction success	Destination success	Collision episodes
Low coupling	30	30	0	30	30	0	30	30	0
Single-resource merge	29	29	1	30	30	0	30	30	0
Crossing	23	23	7	14	14	0	30	30	0
Rolling decisions	0	0	30	0	0	0	30	0	0
Dynamic merge/split	1	1	20	26	26	0	30	30	0
Overlapping multi-resource	0	0	30	19	0	0	30	3	0
All scenarios (180/system)	83	83	88	119	100	0	180	123	0
Off-road (180/system)		39			12			0	

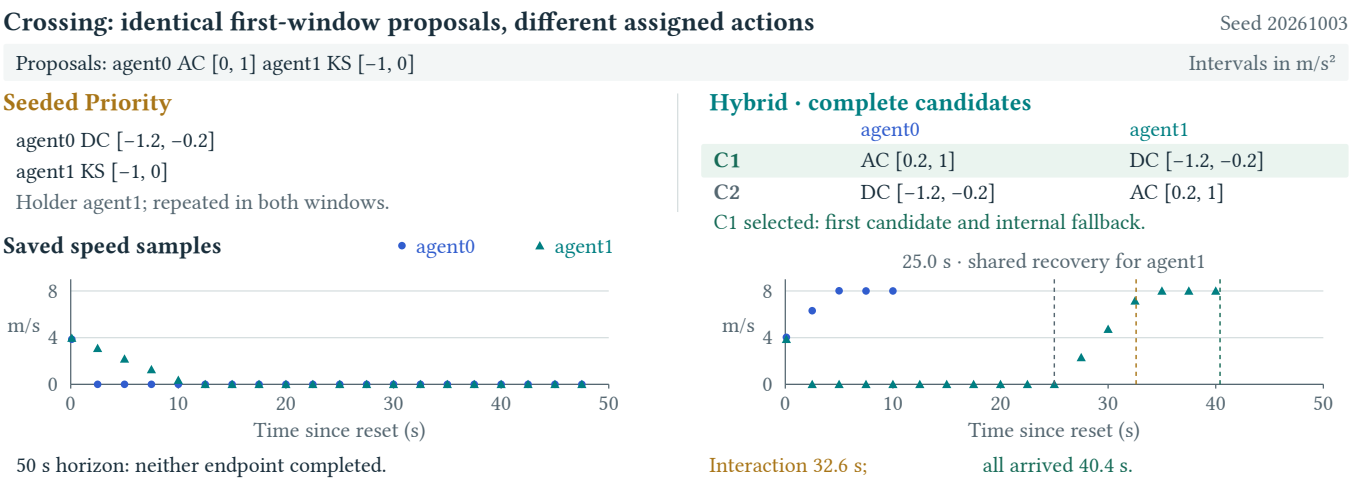


Figure 3: Crossing: the same proposals become different control assignments. Seed 20261003 is the first archived seed. Hybrid selects C1, its own candidate fallback; the two listed candidates cover both agents. Intervals are in m/s^2. Speed markers are saved samples; event lines use episode timestamps. Shared recovery for agent1 begins at 25.0 s without another model selection. Short candidate labels map to full identifiers in the companion evidence.

Overlapping resources: when coordination ends matters. Here agent0 claims zone-a, agent2 zone-b, and agent1 both resources. Agent1 is the common claimant in this dependency relation, not an inferred geometric middle vehicle. All 30 first windows share one state digest and the same candidate content set when list order is ignored. Their complete proposal digests have 30 distinct values and the candidates appear in two orders.

Figure 4 traces two observed continuations. In seed 20261003, the first plan accelerates agent0 and decelerates agent1/2. At 16.7 s, before step 168, the updated two-member, version-2 plan accelerates agent1 while agent2 continues yielding; agent0 has left the coordination group. At 33.4 s, before step 335, there is no group plan and shared recovery accelerates agent2. Interaction completes at 42.9 s, but agent2 still has 33.76 m remaining at the 50 s horizon. This is a sequence of newly installed assignments, not evidence of holder migration across membership keys.

In seed 884749598, the first plan instead accelerates the common claimant, agent1, and decelerates the other two. At 16.7 s no active group conflict remains; agent0 and agent2 enter shared recovery together. Interaction completes at 27.1 s and all vehicles arrive at 37.7 s. Thus the initial assignment is associated with whether the next planning boundary produces another coordinated group or releases both remaining claimants into recovery.

The complete 25/3/2 grouping and success counts appear in Figure 4. In the agent2-first group, agent0 proceeds at the next boundary and agent1 enters recovery at 33.4 s; interaction completes at 44.1 s, leaving agent1 43.10 m short at the horizon. All 27 non-arrivals end at the horizon with one outstanding vehicle whose last recorded speed is about 8 m/s. Checked assignments therefore

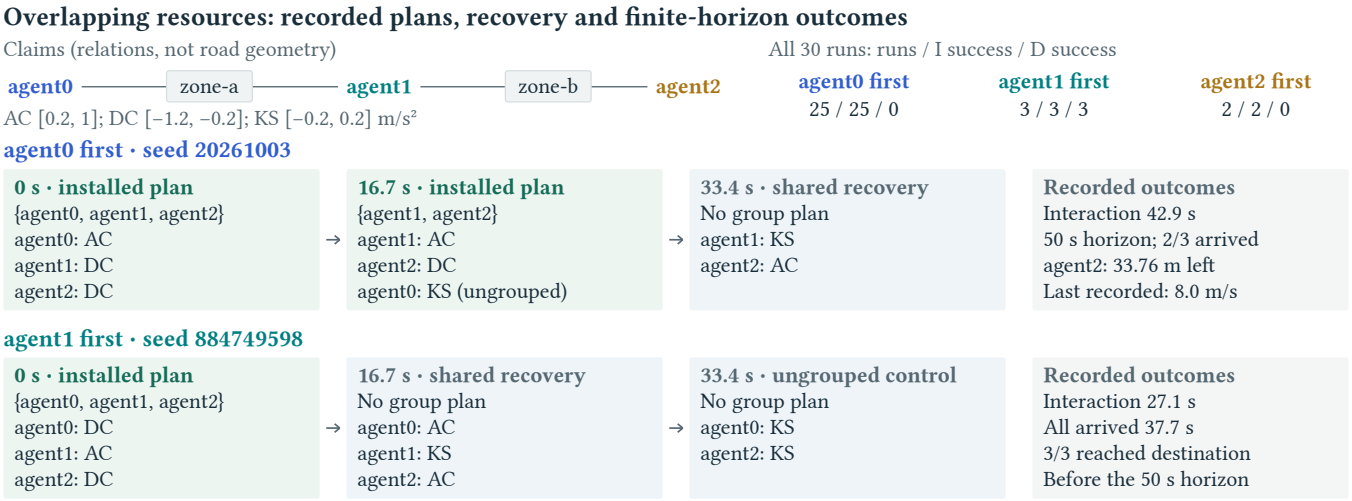


Figure 4: Overlapping resources: checked plans lead to different recorded continuations. The two paths are the first archived runs in the agent0-first and agent1-first groups. Cards show decision records, not trajectories; planning times precede the indexed environment step. I/D denote interaction/destination success. All 30 runs are grouped retrospectively by the first assignment. Candidate content is identical across first windows, while proposal digests and candidate order vary.

coexist with different continuation paths and finite-horizon outcomes. This retrospective association does not establish a counterfactual success rate for a common-claimant-first policy: proposals, list order and later states were not held fixed.

Rolling: a shorter remaining route, the same endpoint distinction. Hybrid rolling completes the interaction at 79.7 s. At the 90 s horizon, three of four initial vehicles have arrived; agent3 has 8.37 m remaining and a last recorded speed near 8 m/s. All 30 runs share this endpoint pattern. In both fixtures, unfinished vehicles remain moving at the horizon; their unexecuted continuation is unknown.

5.4 Selection interface and resource use

All 237 returned Hybrid IDs belong to the supplied candidate lists, and their complete assignments match the saved installed group plans. The lists contain one candidate once, two candidates 206 times and three candidates 30 times; none tests the cap of 32. The model selects the first-listed internal fallback 117 times. These checks establish assignment correspondence, while the retained ranking leaves candidate-order effects unresolved. The artifact's four execution-audit zeros are unmeasured placeholders, documented separately; they are not the source of this correspondence check.

The 1,238 windows contain 2,111 logical model calls and 2,116 transport attempts, including five proposal retries. Hybrid's selector uses 237 calls/attempts, 324,532 input tokens and 3,409 output tokens: 327,941 tokens directly attributable to that role. Its proposals use 594 calls and 595 attempts, compared with Seeded Priority's 800 and 803. Thus the systems' net difference of 29 attempts is not selector cost; their different trajectories also change proposal use. Appendix Table 9 reports all roles. Local cache hits are zero, but upstream cached prompt tokens are recorded. Blocking provider latency measures neither driving time nor real-time feasibility.

6 Limitations, Ethics, and AI Use

The six fixtures fix initial vehicle configurations and test one proposal/selector alias pair without immutable provider snapshots. Small candidate lists and retrospective path groupings leave larger-scale selection and generalization open. Stronger deterministic planners, including priority-order search and priority inheritance [26, 29], and a same-constructor selector comparison remain untested.

Checks depend on encoded resources and actions, and repeated validation can share omissions. Membership-scoped holders are not automatically transferred across changed groups; fixture occupation intervals are not directly validated. Neither eventual arrival nor full physical feasibility follows from admission. Dense traffic, communication faults, adversarial agents and asynchronous execution require further evaluation [3].

Experiments use synthetic states without human-subject data. Road deployment requires independent validation of perception, control and failure handling. Generative AI assisted language editing, organization, LaTeX, figure scripting and schematic artwork; case measurements come from saved execution records.

7 Conclusion

JointAct connects local proposals to complete, constraint-checked joint assignments, restricts model choice to listed IDs and manages permission to use the retrieved plan. Its evaluated Hybrid implementation improves both success totals over the tested Seeded Priority system. Recorded crossing and multi-resource paths explain what these complete-system differences involve: assigned maneuver intervals, subsequent coordination membership and entry into shared recovery. They also show the remaining gap between admitting a compatible assignment, completing the interaction and bringing every initial vehicle to its destination within the configured horizon.

References

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe Reinforcement Learning via Shielding. In *AAAI*.

[2] Benedikt Bollig, Matthias Függer, and Thomas Nowak. 2026. Provable Coordination for LLM Agents via Message Sequence Charts. In *ISoLA*.

[3] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A Zaharia, Joseph Gonzalez, and Ion Stoica. 2025. Why Do Multi-Agent LLM Systems Fail?. In *NeurIPS*.

[4] Philip R. Cohen and Hector J. Levesque. 1991. Teamwork. *Noûs* 25, 4 (1991).

[5] Jiaxun Cui, Chen Tang, Jarrett Holtz, Janice Nguyen, Alessandro G. Allievi, Hang Qiu, and Peter Stone. 2026. CoopReflect: Towards Natural Language Communication for Cooperative Autonomous Driving via Multi-Agent Learning. In *AAMAS*.

[6] Daniel Dauner, Marcel Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, Andreas Geiger, and Kashyap Chitta. 2024. NAVSIM: Data-Driven Non-Reactive Autonomous Vehicle Simulation and Benchmarking. In *NeurIPS*.

[7] Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. 2019. SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems. In *DSN*.

[8] Kurt Dresner and Peter Stone. 2008. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research* 31 (2008).

[9] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2020. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning. In *ICAPS*.

[10] Barbara J. Grosz and Sarit Kraus. 1996. Collaborative Plans for Complex Group Action. *Artificial Intelligence* 86, 2 (1996).

[11] Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. 2002. Coordinated Reinforcement Learning. In *ICML*.

[12] Atharva Gundawar, Karthik Valmeekam, Mudit Verma, and Subbarao Kambhampati. 2024. Robust Planning with Compound LLM Architectures: An LLM-Modulo Approach. arXiv:2411.14484v1

[13] Senkang Hu, Zhengru Fang, Zihan Fang, Yiqin Deng, Xianhao Chen, and Yuguang Fang. 2024. AgentsCoDriver: Large Language Model Empowered Collaborative Driving with Lifelong Learning. arXiv:2404.06345v2

[14] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. 2023. Planning-Oriented Autonomous Driving. In *CVPR*.

[15] Zhe Huang, Zhaoxin Fan, Shuo Wang, Wenjun Wu, Xuan Zhao, and Min Liu. 2026. CoLMIN: LLM-based Multi-Decision Path Negotiation for Cooperative Autonomous Driving. arXiv:2609.04807v1

[16] Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2023. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *CoRL*.

[17] Mert Inan, Malihe Alikhani, and Anthony Sicilia. 2026. Dialogue is the Plan: From Interface to Joint Action in Agentic AI. In *ACL*.

[18] Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. 2024. Bench2Drive: Towards Multi-Ability Benchmarking of Closed-Loop End-To-End Autonomous Driving. In *NeurIPS*.

[19] Bo Jiang, Shaoyu Chen, Qing Xu, Bencheng Liao, Jiajie Chen, Helong Zhou, Qian Zhang, Wenyu Liu, Chang Huang, and Xinggang Wang. 2023. VAD: Vectorized Scene Representation for Efficient Autonomous Driving. In *ICCV*.

[20] Kemou Jiang, Xuan Cai, Zhiyong Cui, Aoyong Li, Yilong Ren, Haiyang Yu, Hao Yang, Daocheng Fu, Licheng Wen, and Pinlong Cai. 2025. KoMA: Knowledge-Driven Multi-Agent Framework for Autonomous Driving with Large Language Models. *IEEE Transactions on Intelligent Vehicles* 10, 10 (2025).

[21] Quanyi Li, Zhenghao Peng, Lan Feng, Qihang Zhang, Zhenghai Xue, and Bolei Zhou. 2023. MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 3 (2023).

[22] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as Policies: Language Model Programs for Embodied Control. In *ICRA*.

[23] Changxing Liu, Genjia Liu, Zijun Wang, Jinchang Yang, and Siheng Chen. 2025. CoLMDriver: LLM-based Negotiation Benefits Cooperative Autonomous Driving. In *ICCV*.

[24] Jiaqi Liu, Chengkai Xu, Peng Hang, Jian Sun, Mingyu Ding, Wei Zhan, and Masayoshi Tomizuka. 2025. Language-Driven Policy Distillation for Cooperative Driving in Multi-Agent Reinforcement Learning. *IEEE Robotics and Automation Letters* 10, 5 (2025).

[25] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *NeurIPS*.

[26] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *AAAI*.

[27] Zhao Mandi, Shreeya Jain, and Shuran Song. 2024. RoCo: Dialectic Multi-Robot Collaboration with Large Language Models. In *ICRA*.

[28] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161, 1–2 (2005).

[29] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2022. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. *Artificial Intelligence* 310 (2022).

[30] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. 2021. Multi-Modal Fusion Transformer for End-to-End Autonomous Driving. In *CVPR*.

[31] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *ICML*.

[32] Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L. Waslander, Yu Liu, and Hongsheng Li. 2024. LMDrive: Closed-Loop End-to-End Driving with Large Language Models. In *CVPR*.

[33] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015).

[34] David Silver. 2005. Cooperative Pathfinding. In *AIIDE*.

[35] Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. 2024. DriveLM: Driving with Graph Visual Question Answering. In *ECCV*.

[36] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *ICRA*.

[37] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Barták. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*.

[38] Milind Tambe. 1997. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research* 7 (1997).

[39] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. 2026. AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. In *ICSE*.

[40] Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao Ma, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. 2024. DiLu: A Knowledge-Driven Approach to Autonomous Driving with Large Language Models. In *ICLR*.

[41] Shuren Xia, Qiwei Li, Taqiya Ehsan, and Jorge Ortiz. 2026. TraceFix: Repairing Agent Coordination Protocols with TLA+ Counterexamples. In *CAIS*.

[42] Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K. Wong, Zhenguo Li, and Hengshuang Zhao. 2024. DriveGPT4: Interpretable End-to-End Autonomous Driving via Large Language Model. *IEEE Robotics and Automation Letters* 9, 10 (2024).

[43] Hanqing Yang, Narjes Nourzad, Shiyu Chen, Marie Siew, Jingdi Chen, and Carlee Joe-Wong. 2026. COOP^2: Defining, Observing, and Repairing Cooperation in LLM Multi-Agent Systems. arXiv:2603.00349v2

Table 5: Decision objects and their connection to checks and execution. These descriptive comparisons concern system design, not cross-benchmark performance.

Work	Model's decision object	Checks, feedback, and execution connection
RoCo [27]	Sub-task for every robot; optional waypoints	Task, inverse-kinematics, and collision checks; joint motion planning.
CoLMDriver [23]	Negotiated driving intentions	Dynamic and historical groups; consensus and waypoint feedback; downstream planning.
CoLMIN [15]	Multiple complete negotiation candidates	Waypoint and consensus scores guide selection, reflection, and further negotiation.
LLM-Modulo [12]	Generated plan	External critics check the proposal and provide feedback for another generation.
JointAct	ID of a preconstructed joint assignment	Encoded joint checks precede selection; retrieval preserves the assignment; separate execution permission.

Table 6: Implemented validation and timing. Hybrid construction and revalidation share the same encoded resource validator; neither is full physical-feasibility verification.

Check	Condition and stage
Coverage and type	Exactly one typed assignment per current member, recognized maneuver and ordered bounds; construction and plan validation.
Identity and lineage	Group/version/window, expected plan ID, coordinator, member set, proposal IDs and validity interval; round validation.
Resource continuity	Holders must be claimants; unresolved prior holder cannot change within the same store key; semantic validation.
Resource exclusivity	No multiple proceeding claimants or non-holder proceeding; overlapping resources cannot have distinct proceeding holders; semantic validation.
Precedence	Hybrid blocks a proceeding successor with an active predecessor; construction and revalidation.
Component progress	Hybrid enumerates one eligible proceeding claimant per component, or none only if the eligibility set is empty; construction.
Installation/access	Installation checks member inclusion, active lifetime and version order; retrieval checks group/version/window and expiry.
Per-step boundary	For installed identities, activation and expiry only; every 0.1 s environment step.

Table 7: Failure paths in the implementation, not injected-fault results. The final configuration stops on model and semantic failures. Candidate fallback and Seeded Priority are different objects.

Condition	Implemented outcome
No active shared conflict	Retire prior group permission; use shared recovery without model selection.
Invalid proposal scope or missing member	Round-input validation raises an error before construction.
Selector unavailable, malformed, or unlisted ID	Final configuration stops the cell; if failure stopping is disabled, retrieve the designated candidate fallback.
Construction exception or plan structural rejection	Runtime returns original proposal assignments, no installed group plan. Model errors staged by the runner still stop the final cell.
Selected plan fails semantic validation	Runtime returns proposals, not another candidate; final runner escalates the semantic failure.
Expired or premature installed identity	Environment-step boundary raises an error. It does not recompute all resource checks.
Repeated same plan ID	Installation returns the existing record; a different older/conflicting plan cannot overwrite it.

A Extended Comparisons and Technical Checks

Table 5 compares decision objects and checking arrangements. Tables 6 and 7 distinguish the implemented checks and failure paths. Message-sequence-chart coordination connects global protocols to local programs [2]; TraceFix repairs coordination protocols using verification feedback [41]. Their protocol-level objects differ from the joint assignments checked here; neither is an evaluated baseline.

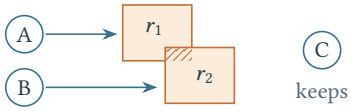
A.1 Objects, validation, and failure handling

Proposals contain agent/action/interval data, local and route context, identity and validity fields, and model provenance. Bundles add member coverage and a digest. Plans bind assignments, resources and holder metadata to group/version/window identity, coordinator, proposal lineage, and lifetime. Candidate IDs identify assignments; runtime plan IDs identify installations.

Table 6 specifies resource exclusivity, precedence and coverage. Component-progress eligibility applies during Hybrid construction. Revalidation repeats the resource validator; occupation intervals remain context metadata, not a numerical swept-volume check. The holder store uses the key $(r, e, (g, m, \mathrm{sort}(G)))$: resource r, conflict episode e, group g, membership version m, and members G. Planning windows do not appear in this key, so a holder can persist across windows within the same membership scope. A changed membership queries a new key; the old record remains stored but is not automatically migrated. Establishing the same agent as holder under the new key is a new record, not evidence of transfer. Release follows the adapter's resource-cleared flag, inactive conflict, fewer than two claimants, or claimants outside the group. These are

Constructed context: $H = \{A, B, C\}$

A and B claim overlapping resources; C does not.



Plan	A	B	C
P_1	proceed	yield	keep
P_2	yield	proceed	keep

Object or condition	Responsible check	Implemented handling
A and B both proceed; or {A:yield, B:proceed} omits C	Joint constraints; member coverage	Exclude during construction; no selector rewrite repairs the object.
Listed ID P_2	Lookup; revalidation; execution permission	Retrieve the entire P_2 unchanged; use only after checks and installation.
ID $Q \notin \{P_1, P_2\}$; or unavailable / unusable response	Selector interface; final-run policy	Stop the cell in the final setup; optional candidate fallback otherwise.
Listed ID P_2, but wrong scope or expired installed lifetime	Round validation; environment-step bound	No stale group-plan use; no second candidate retry after plan rejection.
P_1 and P_2 both pass the encoded constraints	Candidate admissibility; selection preference	Choose among the listed plans; candidate fallback is not Seeded Priority.

Proceed / yield / keep are illustrative roles, not additional controller action codes.

Figure 5: Model choice and implemented checks. Constructed inputs separate candidate validity, ID selection, and installation/use. Final runs stop on model failure; candidate fallback is an optional disabled-error-stop path, distinct from Seeded Priority. Plan rejection does not retry another candidate. The diagram illustrates code branches, not observed model errors or a physical safety guarantee.

recorded priority-state transitions, not continuous measurements of physical zone occupancy.

The final runner stops on model or semantic failure. Disabling `fail_on_model_failure` permits the candidate fallback for unusable selector responses. General construction or structural-validation failure instead returns proposal assignments without a group plan (Table 7); semantic rejection does not select another candidate. Figure 5 illustrates these branches with constructed inputs.

A.2 Action grounding and clock

Let $[l, u]$ be the requested interval and $\mathrm{clip}_{[l,u]}$ interval clipping. The longitudinal controller first computes

$$a^{\mathrm{req}} = \begin{cases} \mathrm{clip}_{[l,u]}(\max(0, u)), & \text{AC}, \\ \mathrm{clip}_{[l,u]}(\min(0, l)), & \text{DC}, \\ l, & \text{SB}, \\ \mathrm{clip}_{[l,u]}(0.2), & \text{KS, LCL, LCR}. \end{cases} \tag{4}$$

For speed at least 8 m/s, a positive request is replaced by $\min(0, 8 - v)$; below or at 0.2 m/s, a negative request becomes zero. The result is clipped to $[-6, 3]$ m/s^2 and then to within 0.3 m/s^2 of the previous acceleration, corresponding to the 3 m/s^3 jerk bound and 0.1 s environment step. Thus controller overrides can move the output outside the proposed interval. Positive acceleration is divided by 3 and negative acceleration by 6 for MetaDrive's normalized throttle/brake input. These are commanded controls, not measured vehicle accelerations.

LCL/LCR request the adjacent route-lane index $-1/+1$, clipped to the available lane range. The lateral adapter retains that target for the same plan ID instead of requesting another lane change at every step. If a target cannot be formed, the runner uses its route-following steering path.

After an agent has no active group permission or resource conflict, the shared recovery rule requests AC$[0.2, 1]$ if it is below 7.75 m/s and not blocked by the proximity rule; otherwise it requests KS$[-0.2, 0.2]$. This ordinary control path is shared by the systems and is not an additional model selection or a certified safety controller.

B Experimental Configuration and Selector Input

Table 8 gives horizons, populations, aliases, and clocks; Table 3 gives planning intervals and window limits.

The selector request supplies proposals, conflicts, candidates and the preference order in Section 4. Its system prompt reads:

Table 8: Driving final configuration. Horizons count 0.1 s environment steps; output-token caps are limits, not measured use. Planning intervals and window limits appear in Table 3.

Scenario	Vehicles	Steps	Seconds
Low coupling	2	500	50
Single-resource merge	2	500	50
Crossing	2	500	50
Rolling decisions	4	900	90
Dynamic merge/split	2	700	70
Overlapping multi-resource	3	500	50

Component	Configuration
Proposal model	`deepseek-v4-flash`; 1,024 output-token cap
Hybrid selector	`kimi-k2.5`; 4,096 output-token cap; listed-ID response
Candidate limits	At most 512 combinations examined; 32 candidates retained
Local cache	Disabled; upstream provider caching is separate
Environment control	0.1 s per step; activation/expiry checks at this rate
Physics integration	Five 0.02 s substeps per environment step
Model waiting	Synchronous; simulation does not advance during calls

Table 9: Recorded provider use by system and role over 180 runs per system. Logical calls count model requests; attempts include transport retries. The rows sum to the original system totals and follow each system's own trajectory.

System/role	Logical calls	Attempts	Input tokens	Output tokens
Independent proposals	480	481	187,830	38,881
Seeded Priority proposals	800	803	314,160	64,389
Hybrid proposals	594	595	226,942	47,503
Hybrid selector	237	237	324,532	3,409

> Select exactly one trusted, prevalidated short-horizon group-plan candidate. Legality, declared route-topology precedence, and feasible component progress are already enforced by trusted code; compare only the listed legal_candidates using the local proposals, proposal_utility, and selection_objective. The provider-visible candidate list intentionally does not reveal which candidate is the trusted fallback. Return one JSON object with exactly one field, candidate_id. Its value must exactly match a listed candidate_id. Do not emit assignments, holders, rationale, repair, votes, ACKs, or any new candidate.

The historical "does not reveal" wording refers to omission of the `deterministic_fallback` flag. Candidate order is retained: the internal fallback is the first item in the deterministic ranking.

Table 10: Completion and success over 180 runs per system. Success additionally excludes collision, off-road and the implemented deadlock flag. Destination completion and success coincide in these records, but their definitions differ.

	Interaction		Destination	
System	Complete	Success	Complete	Success
Independent	92	83	83	83
Seeded Priority	123	119	100	100
Hybrid	180	180	123	123

All 237 saved requests omit the flag; 117 selections choose that first item. Candidate-order effects are not isolated.

The auxiliary utility fields contain hand-written weights. AC, LCL, and LCR start with yielding cost 0.8, other actions with 0.35; the local-context keywords `high`, `urgent`, or `active` add 0.1, while DC/SB override the cost to 0.2. Route urgency starts at 0.75 for AC/LCL/LCR and 0.4 otherwise; route-text keywords `merge`, `cross`, or `bottleneck` add 0.15. Values are capped at 1. Proposal-preservation weight equals yielding cost. These are prompt features, not learned or measured traffic utility. Candidate sorting instead uses proceeding count, maneuver-symbol preservation count, and candidate ID.

C Supplemental Outcomes and Record Scope

Table 10 contrasts completion with success (Eq. 3). Since $K \Rightarrow \neg I$ and $D \Rightarrow I$, the historical deadlock flag is redundant in both predicates, not a general deadlock monitor. All 13 interaction-completion/success disagreements involve off-road events, including the four Seeded Priority dynamic runs discussed in Section 5.

Table 9 separates local-proposal and selector use along each system's trajectory. Across the experiment, 2,111 logical calls generate 2,116 transport attempts; five proposal calls are retried. Local response-cache hits are zero, but 2,103 saved provider responses report nonzero upstream cached prompt tokens. The model aliases do not identify immutable provider snapshots. Summed latency is neither simulator elapsed time nor real-time deadline evidence.

Crossing and rolling use the first seed in the archived plan, 20261003. The multi-resource paths use the first archived run in each first-proceeding-member group, with all 30 runs included in the grouped outcomes. Companion evidence provides episode and window indices, mappings, samples, and both proposal-matching definitions. Samples are normally recorded every 25 environment steps (2.5 s); a terminal sample may contain no active agents. Missing agents are not zero-speed observations. Decision times precede their indexed environment step, while samples retain their recorded post-step times; last-recorded speed need not be the speed at the exact termination instant.

Four `authority_execution_audit` fields in the archived summaries are constant-zero placeholders, not measured violation counts; companion `artifact-field-notes.json` identifies the fields and source assignments. The 237 comparisons of selected IDs, candidates and installed mappings instead use saved requests and plans, independently of these placeholders.