

AssemblyGrid v1: A Benchmark for Multi-Robot Production with Temporary Coalitions, Local Information, and Geometric Constraints

Fouad Bahrpeyma David Heik Dirk Reichelt

Faculty of Computer Science/Mathematics
Hochschule für Technik und Wirtschaft Dresden – University of Applied Sciences
Dresden, Germany

Bahrpeyma@ieee.org david.heik@htw-dresden.de dirk.reichelt@htw-dresden.de

Abstract

Flexible robotic production requires decisions about process progression, material routing, resource assignment, temporary cooperation, and simultaneous execution to be made jointly, because each decision can affect the feasibility of the others. This becomes especially challenging under decentralized control, where each robot acts from bounded local information while overall progress depends on collective decisions, shared resources, material state, and workspace compatibility. Such settings are relevant not only to manufacturing control, but also to cooperative multi-agent decision making under partial observability, local coordination, temporary cooperation, and resource contention. These properties closely match the class of problems addressed by MARL, where decentralized agents must learn from interaction how locally informed decisions contribute to long-term production progress under interdependent actions and an evolving system state. This paper introduces ASSEMBLYGRID v1, a reproducible benchmark for repeated multi-robot production that combines explicit process progression, decentralized observations, material transfer, temporary multi-robot coalitions, productive concurrency, and geometry-dependent feasibility within one task-level formulation. The benchmark suite comprises FLOW, COALITION, and CONCURRENCY workload families, each with three scenario levels. Task success and evaluation measures are defined independently of learning reward and solution method, allowing MARL and other learning-based and non-learning methods to address the same production problem.

ASSEMBLYGRID v1 is evaluated through executable conformance checks, dedicated mechanism studies, and algorithmic evaluations. The algorithmic evaluation includes a privileged centralized reference, structured decentralized controllers, and MARL methods including IPPO, MAPPO, and QMIX. Experimental results confirm the existence of feasible centralized and decentralized control solutions, while the MARL results show that decentralized policies can learn effective production behavior from local observations and actions.

1 Introduction

Modern production systems are moving away from fixed production structures toward configurations in which capacity, routing, and resource assignments can be adapted to changing production requirements. Reconfigurable and flexible manufacturing systems enable such adaptation through modular, scalable, and convertible system designs [1, 2]. Matrix-structured manufacturing extends this flexibility by introducing redundant workstations and alternative routing possibilities [3, 4], while line-less assembly and swarm-production concepts further relax fixed material paths and resource assignments [5–8]. As these alternatives increase, the central challenge shifts from providing flexibility to controlling it: production decisions become increasingly state dependent and interdependent, since the choice of what to execute, where

material should move, which resources should participate, and what can proceed concurrently must be resolved together. In a flexible robotic workcell, this control problem involves four closely related decisions: which product operation to execute, where its material should be routed, which resources should perform the operation, and whether the resulting assignment is geometrically feasible. These decisions are inherently coupled. The selected operation determines the required material and process state, resource assignment affects which activities can proceed concurrently, and workspace geometry may render an otherwise valid process and resource assignment infeasible. Similar dependencies among routing, resource allocation, look-ahead, layout, and material supply have been identified in matrix-production control and integrated production-system planning [9–11].

These dependencies make task-level controller design a central challenge, since task progress, resource use, concurrency, and geometric feasibility must be considered jointly. In this paper, a *controller* refers to any method that selects task-level actions for the robots, including learned policies, planners, schedulers, optimization methods, and heuristics. Benchmarks can support the development and evaluation of such controllers by defining the underlying decision problem reproducibly without prescribing how it should be solved. Multi-agent reinforcement learning (MARL) is particularly relevant in this setting because multiple interacting agents must learn from experience how locally informed decisions contribute to the common production objective. In the cooperative setting considered here, all agents contribute to this shared production task. This task-level cooperation is distinct from a *coalition*, which here denotes a temporary subset of robots that jointly executes a particular operation and dissolves when that operation ends.

The Multi-Agent Particle Environment (MPE) [12] provides compact continuous multi-agent coordination tasks and became a widely used testbed for decentralized policies trained with centralized information. The StarCraft Multi-Agent Challenge (SMAC) [13] became a standard cooperative benchmark for centralized training with decentralized execution, and SMACv2 later extended this benchmark with procedural generation and stronger partial observability to reduce dependence on fixed scenario regularities [14]. Hanabi [15] represents cooperative inference under imperfect information and conventions; Google Research Football [16] provides a physical team-game environment; and MAgent [17] addresses large-population multiagent interaction. Closer to logistics and structured task execution, Level-Based Foraging (LBF) [18] includes tasks requiring simultaneous participation by several agents, whereas Multi-Robot Warehouse (RWARE) [19] combines local observations, delivery objectives and congestion. Flatland [20] represents railway routing under operational conflicts; Overcooked-AI [21], together with later variants by Ruhdorfer et al. [22] and Gessler et al. [23], combines sequential subtasks, blocking and division of labour; and Melting Pot [24] evaluates behavior across changing social configurations.

A complementary body of infrastructure research addresses reproducibility and implementation. PettingZoo standardizes multiagent environment interfaces [25]; the Vectorized Multi-Agent Simulator (VMAS) [26] and Gigastep [27] provide vectorized physical and high-throughput environments; and JaxMARL [28], MARLlib [29], and BenchMARL [30] provide reusable evaluation infrastructure across algorithms and environments. Together, these benchmarks and frameworks cover partial observability, coordination, routing, congestion, conventions, division of labour and large-scale interaction. Repeated typed material transformation, temporary operation-specific robot coalitions, bounded decentralized execution and geometry-dependent admissibility are typically distributed across separate benchmark traditions instead of being combined within one production problem definition.

The remaining mechanisms are developed more directly in neighboring research areas. Multi-robot task allocation was formalized early in [31] and later synthesized in [32], while market-based and other decentralized mechanisms provide established solution families [33–35]. Coalition-formation methods, such as those studied in [36] and, more recently in [37] and [38], extend this setting to tasks for which the participating robot set is itself part of the decision problem. Manufacturing MARL represents production processes through scheduling,

transport and resource-allocation decisions, as surveyed by Bahrpeyma et al. [39] and Xu et al. [40], including distributed factory scheduling [41], joint layout and scheduling [42], dynamic flexible job-shop scheduling [43–45], factory-scale control [46], size generalization [47], breakdown response [48], and knowledge-guided control [49].

Collaborative assembly and task-and-motion planning address another part of the problem by representing workspace interference, synchronization, shared manipulation and collision-aware motion. Representative studies include collaborative assembly and synchronization work by Boschetti et al. [50], human-robot collaboration analysis by Keshvarparast et al. [51], collaborative manipulation by Feng et al. [52], assembly-oriented coordination by Marvel et al. [53], assembly-task reviews by Jiang et al. [54], and task-and-motion planning formulations by Chen et al. [55]. Recent embodied systems such as RAMP [56], RoboFactory [57], the large-scale collaborative setting of Brown et al. [58], APEX-MR (multi-robot asynchronous planning and execution) [59], and Fabrica [60] further incorporate compositional assembly constraints, collaborative transport, and asynchronous or contact-rich execution. These studies address physical and cooperative execution at a substantially higher level of physical detail, although evaluation generally centers on individual assembly tasks or manipulation episodes instead of continuing production with repeated products. Recent reviews of MARL for industrial cooperative manipulation, such as Huertos et al. [61], also identify standardized industrial benchmarks and reproducibility as continuing research needs.

The gap addressed here is the integration of mechanisms that are typically represented in separate problem formulations or at different abstraction levels. Scheduling models repeated process progression and production objectives; task-allocation research represents assignment and coalition formation; collaborative assembly represents geometric and physical feasibility; and cooperative MARL provides established formulations for decentralized decision making under partial observability. What remains less commonly represented is their combination within a single benchmark definition in which typed material progression, temporary local coalitions, productive concurrency and geometric admissibility are simultaneously part of the decision problem, with task outcomes defined independently of the controller and its learning reward, as summarized in Table 1.

Table 1 summarizes how the relevant mechanisms are distributed across neighboring benchmark and problem families, and positions AssemblyGrid v1 relative to them. Hence, AssemblyGrid v1 is organized around six design principles: (1) some operations involve cooperation among multiple robots; (2) products progress through explicit recipe-defined production operations, thereby keeping material and process state explicit; (3) the robot group required for one operation should be temporary and operation-specific instead of fixed across the production process; (4) information locality and manipulation locality should be represented separately; (5) geometric admissibility should constrain otherwise valid task and resource choices; and (6) benchmark outcomes should be defined independently of controller-specific objectives, thereby allowing learning, planning, scheduling, optimization and heuristic methods to be studied on the same production problem. AssemblyGrid v1 combines these elements in a controller-independent benchmark for continuing multi-robot production under decentralized local information, with explicit recipe and material progression, temporary multi-robot cooperation, and task-level geometric constraints. Its official suite organizes workload variation into FLOW, COALITION, and CONCURRENCY, each evaluated under a common benchmark definition. The paper contributes this production task and scenario suite together with a reproducible evaluation framework that uses conformance checks, mechanism-level validation, and reference controllers to demonstrate benchmark executability. The remainder of the paper follows the benchmark from definition to validation and scope. Section 2 defines the environment and decentralized decision problem, Section 3 specifies benchmark instances, workload families, evaluation measures and benchmark requirements, and Section 4 presents the experimental validation and analysis. Section 5 then discusses scope, limitations and extensions, and Appendix B provides the complete formal

Table 1: Cross-literature mechanism matrix.

Family	Repeated process / material state	Temporary multi-robot coalition	Bounded local decentralized execution	Task and motion / geometry coupling	Production-rate outcome	Reusable benchmark
Generic cooperative MARL (MPE, SMAC/SMACv2, Hanabi) [12–15]	–	○	✓	○	–	✓
Logistics and routing MARL (LBF, RWARE, Flatland) [18–20]	○	○	✓	○	○	✓
Overcooked family [21–23]	○	–	✓	○	○	✓
Multi-robot task allocation / coalition formation [31, 32, 36]	○	✓	○	○	○	○
Reconfigurable / matrix / line-less / swarm production [1, 3, 5, 7]	✓	○	○	○	✓	○
Manufacturing MARL / scheduling [39, 41, 42]	✓	○	✓	○	✓	○
Task and motion planning / collaborative assembly [50, 54, 55]	○	✓	○	✓	○	○
RAMP / RoboFactory / Fabrica [56, 57, 60]	○	○	○	✓	–	✓
ASSEMBLYGRID v1	✓	✓	✓	○ (Assembly-Grid v1 abstract proxy)	✓	✓

✓ primary mechanism
 ○ present in some members or at a coarser abstraction
 – generally outside the modeled object

benchmark definition.

2 The AssemblyGrid v1 environment

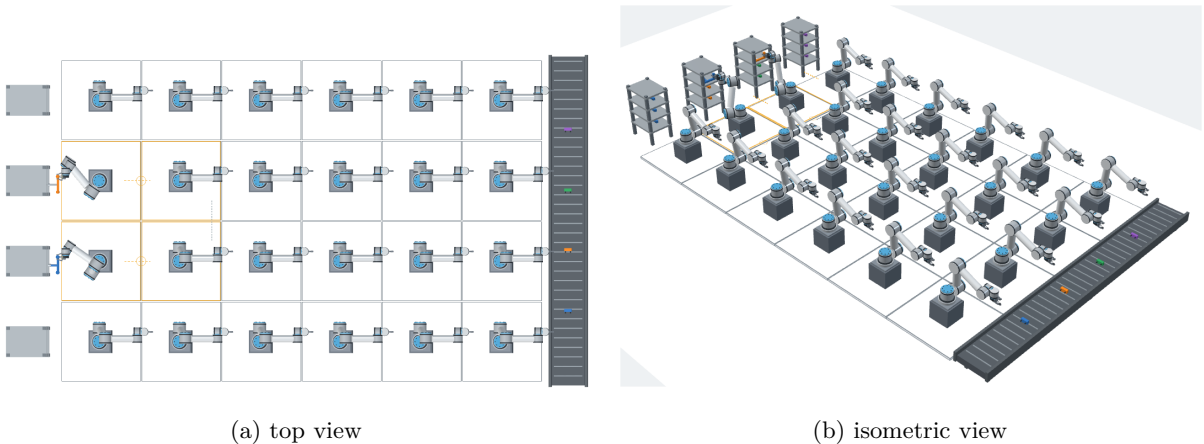


Figure 1: Architectural illustration of the AssemblyGrid v1 task-level production cell (illustrative, while geometry remains abstract).

Figure 1 illustrates the AssemblyGrid v1 production cell, where fixed-base manipulators occupy a regular lattice, raw materials enter from the left, products progress through recipe-defined operations, and completed products leave on the right. Overlapping workspaces determine where robots can interact, form temporary coalitions, and interfere with concurrent activities. Each

robot is paired one-to-one with a task-level agent, and thus the benchmark connects physical interaction and material flow directly to recipe requirements, local actions, coalition formation, and decentralized decision making.

2.1 Physical organization and local interaction

The logical representation of AssemblyGrid v1 is shown in Figure 2.

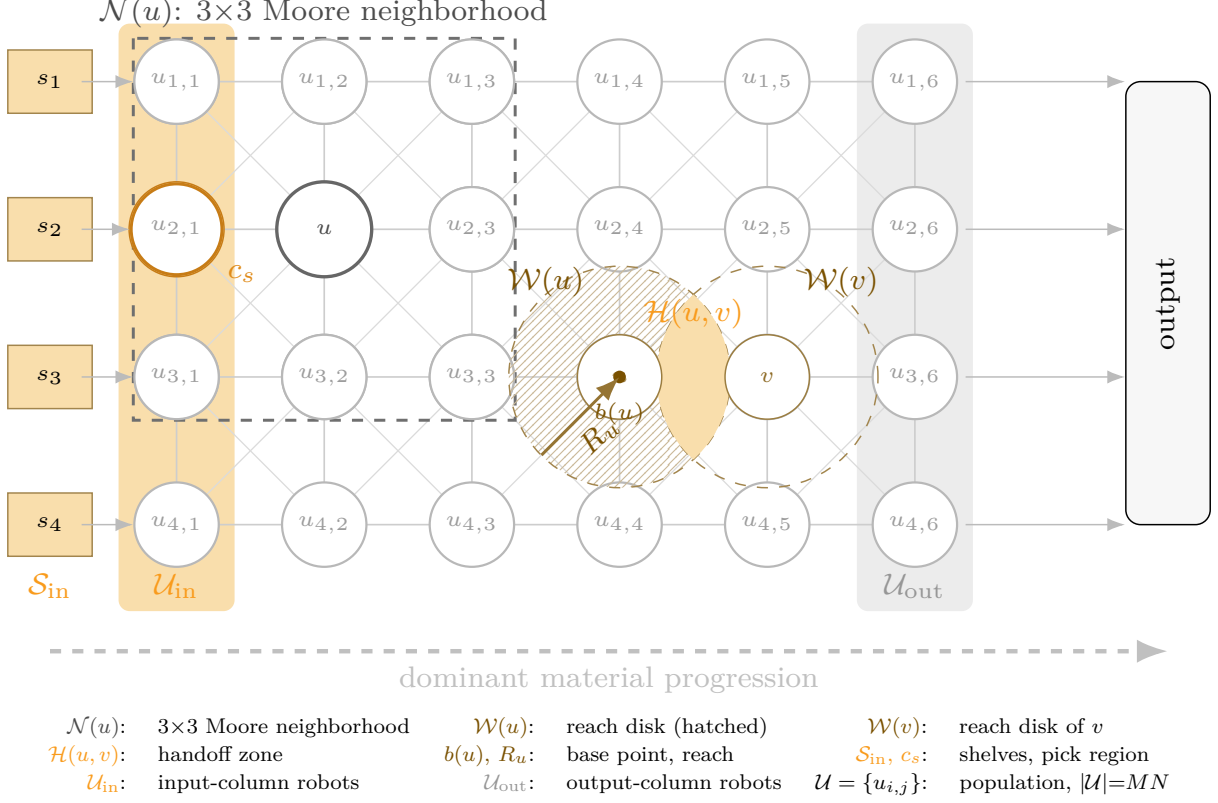


Figure 2: Physical organization and local interaction in AssemblyGrid v1.

Robot grid and local topology. For an $M \times N$ grid, the robot population is

$$\mathcal{U} = \{u_{i,j} : 1 \leq i \leq M, 1 \leq j \leq N\}, \quad |\mathcal{U}| = MN, \quad (1)$$

where M and N are the numbers of grid rows and columns, $u_{i,j}$ is the fixed robot at grid position (i, j) , and $\mathcal{U}$ is the complete robot population. AssemblyGrid v1 distinguishes three forms of locality: observation locality $\mathcal{G}_{obs}$, bilateral-handoff locality $\mathcal{G}_{handoff}$, and cooperative-manipulation locality $\mathcal{G}_{manip}$. These relations share the same Moore-radius-one topology in the default AssemblyGrid v1 configuration while remaining semantically distinct, since information access, material transfer and joint manipulation are different benchmark relations. Each robot's local interaction set is its closed Moore neighborhood $\mathcal{N}(u_{i,j})$ (Appendix B.7). As illustrated in Figure 2, a robot away from the grid boundary has eight direct neighbors. Under Moore-radius-one manipulation locality, every mutually adjacent coalition is contained within at most a (2×2) local block. Thus $k_{topo}^{max} = \omega(\mathcal{G}_{manip})$, which equals four for the official AssemblyGrid v1 grids; smaller coalitions may occupy any admissible subset of such a block.

Abstract geometric feasibility. AssemblyGrid v1 uses a lightweight task-space geometry model to decide whether local robot interactions are spatially admissible. Figure 2 illustrates the key quantities: robot u has an abstract reachable workspace $\mathcal{W}(u)$ of radius R_u about

its base $b(u)$, and neighboring robots can hand off only inside the overlap of their reachable workspaces. Cooperative operations assign role-specific targets around a common task location; concurrent activities must additionally satisfy reach, clearance, and workspace-conflict constraints. Approach and retreat distances contribute to the abstract motion delay. Geometric distances are expressed in normalized task-space units; the formal workspace relations and geometry-profile ownership are specified in Appendix B.3, while the active profile or named variant is part of the benchmark-instance definition.

Scope of the geometry model. The geometry model represents spatial admissibility at the task level. Reach, workspace overlap, clearance, and interference determine whether handoffs, cooperative operations, and concurrent activities are admissible and contribute to their abstract execution time. Robot-specific kinematics, trajectory planning, grasping, and dynamics lie outside this abstraction and may be incorporated through higher-detail extensions.

Input, internal flow, and output. As shown in Figure 2, Raw material enters along the left boundary through $\mathcal{S}_{\text{in}}$ and is picked by the left-column robots $\mathcal{U}_{\text{in}}$; finished products leave through the right-column robots $\mathcal{U}_{\text{out}}$ (Appendix B.7). Material transfer between robots occurs through local bilateral handoff.

2.2 Product recipes and operation requirements

A product progresses according to its recipe without following a predefined robot route. The recipe specifies the production operations that must be completed, the order in which they may occur, the material each operation consumes or produces, and any coalition or manipulation roles required for execution. The recipe separates *what must be produced* from *which robots perform each operation and where*.

Material and product state. For every material item q , representing a part or subassembly, the environment records its location, holder when applicable, abstract task pose, process status, and associated product. In the default setting a robot holds at most one item, $\sum_q \mathbf{1}\{\text{holder}_t(q) = u\} \leq 1$, where $\text{holder}_t(q)$ denotes the robot holding item q at decision epoch t , when one exists.

Recipe graph. Each product type $r \in \mathcal{R}$ is defined by a recipe structure $\Gamma_r = (\mathcal{T}_r, \mathcal{E}_r, \mathcal{X}_r)$, whose precedence component $(\mathcal{T}_r, \mathcal{E}_r)$ is a directed acyclic graph (DAG). Here $\mathcal{T}_r$ contains the recipe operations, $\mathcal{E}_r$ contains mandatory predecessor relations, and $\mathcal{X}_r$ records mutually exclusive operation alternatives. A *sequence* enforces a declared order, an *AND* relation requires all predecessor branches, and an *exclusive OR (XOR)* relation selects one alternative and closes its mutually exclusive siblings for that product unit. Thus sequence and AND are encoded by the mandatory predecessor structure, while XOR is represented explicitly by $\mathcal{X}_r$. For product unit z and recipe operation τ , the pair (z, τ) denotes that particular operation for that product and is referred to as an *operation occurrence*; its readiness indicator $\text{ready}_t(z, \tau) \in \{0, 1\}$ (Appendix B.4) states whether the recipe precedence permits it to proceed. Readiness alone does not guarantee execution: the required material, a suitable robot group, roles, site and auxiliary resources, reach, clearance, and compatibility with simultaneous work must also be satisfied. Successful completion applies the recipe-defined material transition Δ_τ : required inputs are consumed or incorporated and the resulting part or subassembly state is created or updated atomically. Applying Δ_τ atomically makes material conservation, subassembly provenance, joins, and terminal product formation explicit without a second progress mechanism separate from the recipe. To make these requirements explicit, every recipe operation records what it consumes, how successful completion transforms the material state, how many robots it needs, the roles required, which exclusive resources it uses, and how long productive processing takes; its geometric requirements

and work location follow from the coalition and the active geometry profile. We write this compactly as

$$\tau = (\text{type}_\tau, \text{inputs}_\tau, \Delta_\tau, k_\tau^{\min}, k_\tau^{\max}, \text{roles}_\tau, \text{res}_\tau, d_\tau), \quad (2)$$

where Δ_τ is the recipe-defined material-state transition and the remaining fields are summarized in Table 2.

Table 2: Recipe-operation fields.

Symbol	Meaning
type_τ	operation type
inputs_τ	required input material or subassemblies
Δ_τ	material-state transition applied atomically on successful completion
$[k_\tau^{\min}, k_\tau^{\max}]$	admissible coalition-size interval
roles_τ	ordered manipulation roles and any material-role bindings used by the canonical assignment; AssemblyGrid v1 instantiates exactly one role per participating robot
geom_τ	derived, not declared by the recipe: geometric requirements fixed by the active geometry profile
sites_τ	derived, not declared by the recipe: the common target point of the input holders
res_τ	additional exclusive resources
d_τ	nominal productive processing duration

Recipe-defined cooperation. Each operation declares the coalition sizes that are semantically valid:

$$k_\tau^{\min} \leq |C| \leq \min(k_\tau^{\max}, k_{\text{topo}}^{\max}), \quad (3)$$

where C is a concrete coalition and $k_{\text{topo}}^{\max} = \omega(\mathcal{G}_{\text{manip}})$ is the clique-number bound imposed by the active manipulation topology. For the official AssemblyGrid v1 Moore-radius-one grids, this value is four. By default $k_\tau^{\min} = k_\tau^{\max}$, and the recipe therefore requests an exact size; a wider declared interval remains capped by the active topology. When $k_\tau^{\min} < k_\tau^{\max}$ and explicit roles are used, the AssemblyGrid v1 role declaration covers the maximum admissible team. Material-bound roles are assigned first to the robots holding their associated inputs, and the remaining participating robots, ordered row-major, receive the remaining declared roles in recipe order; a realized size- k candidate therefore instantiates exactly k roles. The productive duration d_τ is fixed with respect to coalition size; cooperation therefore represents operation requirements such as holding, stabilizing, aligning, inserting, or fastening. Geometry may nevertheless add approach and retreat time. Before evaluation, recipes are validated for schema correctness, precedence consistency, material semantics, coalition-size compatibility, and realizability under the declared benchmark constraints, with the complete validation, topology, transition, and reporting definitions given in Appendix B.

2.3 Local decisions and temporary coalitions

Time is represented discretely, with decision epoch t denoting the beginning of simulation tick t and each tick serving as the benchmark unit of simulated duration. Available robots select task actions at these decision epochs, while committed activities may remain active across multiple ticks. At each decision epoch, an available robot selects among feasible local actions. A ready operation occurrence (z, τ) may be executable by more than one admissible robot set. Each feasible combination of the operation and a participating robot set is represented as an *execution candidate* (hereafter, candidate refers to an execution candidate). The identity of an execution

candidate is given by the canonical key $\kappa(c) = (z, \tau, \text{sort}_{\text{rm}}(U_c))$, where U_c is the proposed robot set and sort_{rm} denotes row-major sorting. For a fixed benchmark instance and decision state, AssemblyGrid v1 uses a fixed rule to decide how it is carried out in the environment: Material-bound roles are assigned to the robots holding the required inputs, and the remaining roles are assigned to the other robots in the proposed set in row-major order. The operation site is computed from the relevant robot base positions, and the active geometry profile determines the geometric assignment. Thus, each candidate key corresponds to a single concrete realization. These attributes therefore specify one concrete realization without creating a second execution candidate identity for the same key. The task-level action types are

$$\mathcal{A}^u = \{\text{idle}, \text{pick}(q), \text{handoff}(v, q), \text{receive}(v, q), \text{support}(c), \text{deliver}(z)\}, \quad (4)$$

where $\mathcal{A}^u$ is the task-level action set of robot u , q is a locally accessible material item, v is a direct handoff neighbor, c is an execution candidate, and z is a product unit ready for delivery. By selecting $\text{support}(c)$, robot u chooses to participate in the particular operation execution represented by candidate c . Recipe-specific operations such as inspection or assembly are represented through candidate selection, without separate action types. Each agent thus selects a task-level activity, while the environment evaluates the geometry and timing associated with the selected option under the active benchmark profile.

Bilateral handoff. A handoff occurs only when two neighboring robots make matching choices in the same decision epoch: one robot chooses $\text{handoff}(v, q)$ and the other chooses $\text{receive}(u, q)$. The material can move only through the shared reachable region $\mathcal{H}(u, v)$ illustrated in Figure 2 (Appendix B.4). Requiring both robots to choose the transfer preserves handoff as a coordinated decision and prevents the environment from executing the transfer automatically.

Dynamic coalition formation. Each eligible agent may select $\text{support}(c)$ for one candidate, while admission occurs only when every required robot supports the same option and the environment confirms that the combined execution is feasible. The resulting admitted activity $e = e(c)$ is executed by the temporary coalition $C(e)$, whose participating robots remain committed to their assigned roles until the activity completes, after which they return to the available resource pool. Support is a pre-admission choice, whereas commitment begins only after admission.

Execution-candidate participation lifecycle. By selecting $\text{support}(c)$, robot u records its intention to participate in execution candidate c ; the record may persist across decision epochs, and the required robots need not select the candidate simultaneously. A robot can support at most one candidate at a time. If support is created at decision epoch t_0 , it is eligible for admission at epochs $t_0, \dots, t_0 + 3$: admission is checked before the clock advances, and expiration is applied afterward; consequently, the first expiry occurs at $t_0 + 4$. Re-selecting the same candidate refreshes neither the timeout nor the one-epoch switching lock; switching to a different candidate is permitted from $t_0 + 1$. A fully supported candidate that is still blocked at $t_0 + 3$ remains unadmitted and expires at $t_0 + 4$ unless it becomes invalid earlier. Accordingly, pending-support state retains the candidate identity and the timing information needed to determine its remaining lifetime and switching eligibility.

Coordination cue. Coordination in AssemblyGrid v1 is informed by an environment-mediated support-fraction cue without introducing an explicit messaging channel. For each execution candidate available to an agent, the observation includes a *support fraction*: the number of robots required by the candidate whose support is already recorded, divided by the number of robots the candidate requires. Observations at decision epoch t are formed from the state before

the actions selected at t are applied. The cue indicates how much pending participation has accumulated for that specific execution candidate without revealing the complete support state.

Task selection and geometric execution. Each eligible robot selects its own task-level action. When the required members support the same execution candidate, the environment evaluates its joint compatibility before admission. If candidate c is admitted, the resulting activity $e = e(c)$ has realized duration

$$D(e) = D_{\text{approach}}(e) + D_{\text{process}}(e) + D_{\text{retreat}}(e), \quad (5)$$

where $D_{\text{approach}}(e)$ and $D_{\text{retreat}}(e)$ are the abstract approach and retreat delays, respectively, and $D_{\text{process}}(e) = d_\tau$ is the productive processing duration for an activity executing operation τ in AssemblyGrid v1. Decomposing execution into approach, process, and retreat keeps the action space at task level while allowing reach, clearance, and concurrent workspace conflicts to affect feasibility and resource occupancy. Formally, a robot with base point b_u and reach radius r_u can serve a target point x only if $\|x - b_u\|_2 \leq r_u$. Each activity e is represented by the planar approach and retreat segments $\mathcal{S}(e)$ of its members and its target point $x(e)$. With $\text{dist}(s, s')$ the minimum Euclidean distance between two segments, zero when they intersect, two activities e and e' conflict geometrically if $\min_{s \in \mathcal{S}(e), s' \in \mathcal{S}(e')} \text{dist}(s, s') < c(e, e')$ or $\|x(e) - x(e')\|_2 < c(e, e')$, where $c(e, e') = \rho_{\text{conf}} + \max_{u \in C(e) \cup C(e')} c_u$ combines the workspace-conflict radius with the largest robot clearance c_u among the members of both activities.

A fully supported candidate is admitted only if its robots, required material, role requirements, operation site, exclusive auxiliary resources, and abstract geometric predicates are jointly compatible. Overlap between possible local coalition footprints is not itself a conflict. Two admitted or executing activities conflict if they share an exclusive robot or auxiliary resource, or if their simultaneous abstract motion segments or target workspaces violate the active geometry setting's clearance rule.

Action masking. Per-agent masks remove only hard impossibilities that can be determined from the agent's permitted local information, such as an unavailable material pick, an unreachable local target under the active geometry setting, or an action outside the robot's static capability. Let m_t^u denote the local action mask supplied to agent u at decision epoch t . The mask is limited to locally detectable impossibilities; matching choices by other agents and hidden remote conflicts remain unresolved until the joint transition. Coalition agreement, matching handoff choices, remote contention, and conflicts between otherwise legal local actions are resolved by the joint transition model.

Candidate exposure boundary. Candidates may be generated from the global state, but robot u sees only those it can locally take part in. Remote conflicts remain hidden and are handled only during joint admission.

2.4 Decentralized information model

Each agent selects its action from local execution information, while the next production state depends on the combined actions of all agents; the formal model therefore separates a global state and transition process from the action and observation spaces available to each agent. Denoting the AssemblyGrid v1 environment by $\mathcal{E}_{\text{AG}}$, we represent it as the decentralized partially observable controlled process

$$\mathcal{E}_{\text{AG}} = \langle \mathcal{U}, \mathcal{S}, \{\mathcal{A}^u\}, P, \{\mathcal{O}^u\}, O \rangle, \quad (6)$$

where $\mathcal{U}$ is the robot population, $\mathcal{S}$ is the global state space, $\{\mathcal{A}^u\}$ are the per-agent action spaces, P is the joint transition model, $\{\mathcal{O}^u\}$ are the local observation spaces, and O is the

observation function (Appendix B.1). The recipe family, feasibility rules, timing model, success condition, and termination rule complete the environment definition.

At decision epoch t , agent u receives execution information

$$I_t^u = (o_t^u, m_t^u), \quad (7)$$

where $o_t^u \in \mathcal{O}^u$ is its local observation and m_t^u is the local action mask defined above. A decentralized controller can be represented by a collection of agent policies $\Pi = \{\pi^u\}_{u \in \mathcal{U}}$. For a history-dependent controller,

$$a_t^u \sim \pi^u(\cdot \mid h_t^u), \quad h_t^u = (I_0^u, a_0^u, \dots, a_{t-1}^u, I_t^u), \quad (8)$$

while a memoryless controller uses $a_t^u \sim \pi^u(\cdot \mid I_t^u)$. The independently selected agent actions form the joint action

$$\mathbf{a}_t = (a_t^u)_{u \in \mathcal{U}} \in \prod_{u \in \mathcal{U}} \mathcal{A}^u, \quad s_{t+1} \sim P(\cdot \mid s_t, \mathbf{a}_t). \quad (9)$$

Thus multiple decision makers select their own action components, while their consequences are coupled through one environment transition. In this paper, a *policy* is an agent-level action-selection mapping used by a controller.

Reward independence. $\mathcal{E}_{\text{AG}}$ is specified independently of reward and discounting. Reward functions and discount factors are introduced only by learning methods that require them. Reinforcement-learning algorithms may add a learning reward R and discount factor γ to instantiate a cooperative decentralized partially observable Markov decision process (Dec-POMDP) [62] on top of $\mathcal{E}_{\text{AG}}$ when the reward is common to all agents, or a partially observable stochastic game when it contains agent-specific terms, while planners, schedulers, optimization methods, and heuristic controllers use $\mathcal{E}_{\text{AG}}$ directly. Benchmark outcomes are defined and scored through common production and task-success measures independently of accumulated learning reward. Learning return may additionally be reported as a method-specific optimization diagnostic, without becoming part of the benchmark task definition. Methods with different internal objectives therefore remain comparable under the same benchmark instances and metrics. The corresponding global state s_t carries material and product state, recipe progress, robot holdings, pending candidate-support records and their lifecycle timing, coalition commitments, resources, abstract geometric state, and benchmark time, as detailed in Appendix B.2.

Local observation principle. The decentralized observation o_t^u contains the task information locally available to agent u , including the agent’s own state, nearby robot and material state, and the execution candidates in which the agent can participate. The complete execution information I_t^u additionally contains the local action mask m_t^u . Table 3 summarizes the boundary directly: an agent sees local task state, its available execution options, its role and local geometric information, and the aggregate support for those options, while it does not see the global production state, remote contention, detailed intentions of other agents, or explicit messages.

Table 3: Decentralized execution information in AssemblyGrid v1.

Available to a policy for agent u	Not exposed to agent u
• Own task-level state and current commitment • Task-level state of local neighbors • Locally visible material and transfer opportunities • Eligible local execution candidates and their defining attributes • Own candidate role, expected duration, and locally computable reach margin • Local action mask for hard impossibilities detectable from permitted local information • Aggregate support fraction for each candidate • Recipe information needed to interpret local choices	• Global benchmark state • State of remote robots • Remote material and remote resource contention • Global joint action or global feasibility oracle • Raw robot configurations and low-level trajectories • Masking based on hidden nonlocal conflicts • Explicit supporter list or other agents' pending intentions • Free-form or addressable messages

The canonical operation occurrence and proposed robot set identify the same execution candidate for every eligible agent. The associated operation site and geometric assignment are deterministic candidate attributes, while role information is presented from each agent's own perspective. Each eligible agent must nevertheless be able to distinguish the candidate options available to it. The local-information restriction governs action selection during evaluation, while training may use additional global information provided that the resulting execution policy still selects actions from the permitted local information I_t^u . Section 4 uses this distinction to separate centralized training from controllers that use privileged information during execution. Memory over an agent's own execution-information and action history preserves the same information boundary, since that history contains only information already available to the agent. Recurrent or history-dependent policies are therefore permitted within the decentralized information structure.

3 Benchmark design, scenario suite, and measurement

This section defines the benchmark instances, scenario suite, task-success conditions, and evaluation measures used for reproducible comparison. The official suite contains the FLOW, COALITION, and CONCURRENCY workload families, while geometric admissibility constrains all three and is varied separately in matched controls.

3.1 Benchmark instances and reproducibility

Benchmark, runtime, method, and evaluation parameters. A benchmark instance is one concrete production problem. Settings that can change which actions are possible, how the production state evolves, what information agents receive, or what counts as task success are benchmark-defining. These include the grid and interaction topology, recipe, robot capabilities, geometry settings, material and resource structure, demand process, work-in-progress (WIP) bound, and initial condition. AssemblyGrid v1 also fixes runtime semantics shared by the official scenarios, including the tick and transition order and the coalition-support switching and timeout rules. These runtime semantics are benchmark controlled and cannot be altered by a controller during evaluation; changing them requires an explicitly named benchmark or runtime variant. Method-level quantities such as network architecture, optimizer, learning reward, and learning rate may differ between controllers without changing the benchmark task. Evaluation-level quantities such as the declared horizon, dataset split, logging, and plotting procedure belong to

the evaluation protocol and must be held fixed within a reported comparison. Some quantities are derived not chosen. In particular, the topological coalition limit is

$$k_{\text{topo}}^{\max} = \omega(\mathcal{G}_{\text{manip}}) \quad (10)$$

where $\mathcal{G}_{\text{manip}}$ is the direct-manipulation graph and $\omega(\mathcal{G}_{\text{manip}})$ is its clique number. The topological coalition limit is therefore determined by the manipulation graph and is never a free parameter. Since geometry changes which commitments are feasible, changing the geometry setting also changes the benchmark instance.

Geometry settings. A *geometry profile* is a named set of spatial and timing assumptions that AssemblyGrid v1 applies together whenever an activity is checked for geometric admissibility. The profile determines how robot reach relates to grid spacing, the clearance required between simultaneous workspaces, the approach and retreat timing, and the threshold used to flag workspace conflicts. Keeping these settings together under one named profile makes the geometric assumptions visible and consistent across evaluations. A *geometry-control variant* is an explicitly declared change to one or more of these settings for a matched study; it does not silently redefine the workload family.

Scenarios, instances, and random seeds. The evaluation hierarchy is *workload family* $\rightarrow$ *difficulty level* $\rightarrow$ *official scenario* $\rightarrow$ *benchmark instance* $\rightarrow$ *evaluation execution*. An official scenario gives the complete benchmark configuration for one family and difficulty level. A *benchmark instance* is one concrete production problem generated from that scenario. Its generation random seed determines any scenario-level randomized choices. If the environment contains stochastic evolution, an execution random seed determines that realization during evaluation. The algorithm or training random seed controls controller construction, such as initialization or exploration, and is not part of the benchmark instance. Exact reproducibility metadata and instance fingerprints are specified in Appendix A.

Feasibility. A benchmark instance is **feasible** when at least one successful execution exists independently of the evaluated policy; the material-reachability, handoff-chain, coalition-size, operation-site, and resource conditions this requires are stated in Appendix B.5. An instance that violates them is invalid. Task success is defined in Section 3.4.

3.2 The AssemblyGrid v1 scenario suite

The official AssemblyGrid v1 suite contains three workload families with different purposes (Table 4). FLOW emphasizes recipe depth and material progression, COALITION emphasizes temporary multi-robot operations, and CONCURRENCY emphasizes simultaneous productive work under increasing production pressure, with each family providing easy, medium, and hard scenarios. These labels encode predefined family-specific workload configurations and are compared within a workload family; they do not imply a monotonic ordering of every controller outcome. Scaling, topology transfer, failures, communication interventions, and robot-specific physics remain outside the AssemblyGrid v1 suite evaluated in this study. All nine official scenario definitions are released as runnable benchmark configurations from which conformance-checked benchmark instances are generated using declared instance seeds, and the MARL-controller characterization in this paper covers all three workload families at every difficulty level. The non-learning executability validation is presented in detail for the COALITION family, which most directly exercises temporary multi-robot cooperation together with local information, material progression, and geometric feasibility.

Table 4: Official AssemblyGrid v1 workload families.

Family	Primary workload characteristic	Benchmark question
FLOW	Sequential production depth under increasing system load	Can controllers route material and respect increasingly deep recipe precedence without a coalition requirement?
COALITION	Temporary role-constrained coalition formation and increasing coalition structure	Can local robots form, maintain, and dissolve temporary coalitions as the required cooperative structure becomes more demanding?
CONCURRENCY	Productive overlap under increasing production pressure	Can controllers exploit independent ready work while avoiding congestion and unnecessary blocking?

All nine official scenarios use the same default geometry setting, with reach, clearance, motion delay, and workspace-conflict checks remaining active. The matched geometry study changes only the workspace-conflict radius and keeps the production scenario, underlying generated production realizations, generation and execution seeds, controller, and all non-geometric settings fixed; the resulting geometry settings are matched instance variants. The matched design isolates geometric interference from workload and controller changes, with the corresponding results reported in Section 4.2. Benchmark instances are generated deterministically from their declared random seeds. Conditional on the environment version, scenario configuration, instance-generation seed, execution seed, and joint action sequence, AssemblyGrid v1 transitions are deterministic. The determinism claim concerns environment transitions; exact bitwise reproduction of neural-network training can additionally depend on hardware, accelerator kernels, framework versions, and deterministic-computation settings. The controller experiments use separate fixed random-seed sets for training, validation, and final testing, as specified in Section 4.

3.3 Difficulty levels: easy, medium, and hard

Within each family, easy, medium, and hard define difficulty levels. A level may change several parameters when those parameters jointly create the intended workload structure; easy-to-hard differences therefore describe complete scenarios. Figure 3 explains the main change within each family, including the WIP pressure used in the CONCURRENCY family, while Table 5 gives the complete scenario settings.

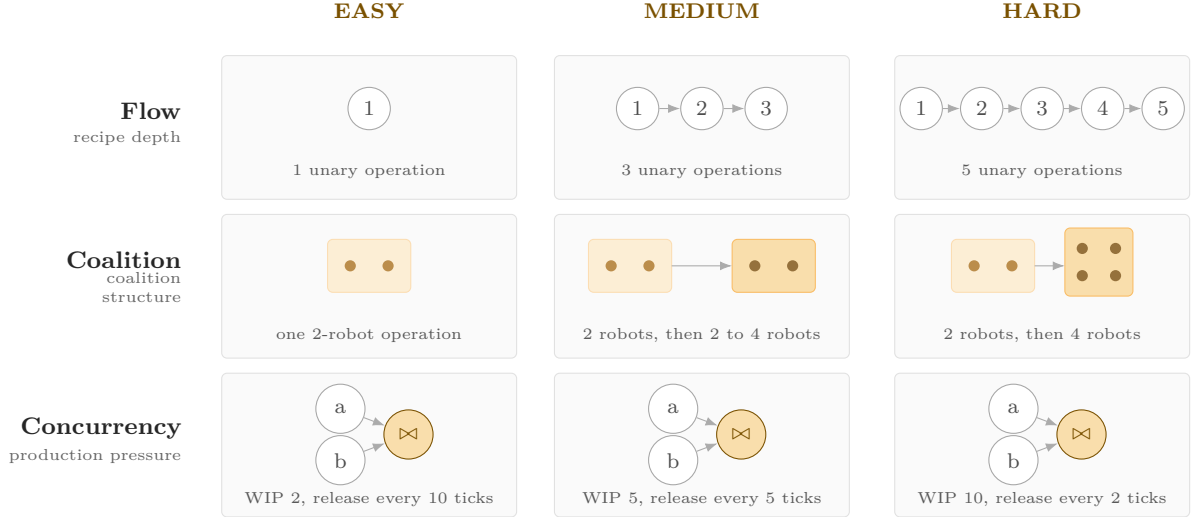


Figure 3: Difficulty levels across the three workload families. The figure highlights each family’s primary progression; complete scenario levels may change several parameters together, as listed in Table 5.

Across the official suite, all nine scenarios use the default normalized workspace-conflict radius $\rho_{\text{conf}} = 0.25$; geometry-control variants are declared separately, while Table 5 reports the remaining workload parameters that define each scenario. Keeping this geometry setting fixed across the workload suite allows differences among FLOW, COALITION, and CONCURRENCY to be interpreted without simultaneously changing the common spatial-interference layer.

Table 5: Official AssemblyGrid v1 scenarios.

Family	Difficulty	Grid	WIP	Release	Recipe	$ \mathcal{T}_r $	$\max_{\tau} k_{\tau}^{\min}$
FLOW	easy	4×5	3	8	single unary operation	1	1
	medium	5×6	5	5	3-stage serial	3	1
	hard	6×8	10	2	5-stage serial	5	1
COALITION	easy	5×6	4	7	single 2-robot operation	1	2
	medium	5×6	4	5	fixed 2-robot operation, then variable 2-to-4-robot operation	2	2
	hard	6×8	1	8	2-robot then 4-robot operations	2	4
CONCURRENCY	easy	5×6	2	10	two unary branches with 2-robot join	3	2
	medium	5×6	5	5	two unary branches with 2-robot join	3	2
	hard	6×8	10	2	two unary branches with 2-robot join	3	2

Flow. Sequential production is the primary challenge. The recipe grows from one unary operation to three and then five unary operations, the grid and WIP limit increase, and products are released more frequently. The family therefore tests routing and precedence handling under progressively larger serial-flow scenarios.

Coalition. Temporary multi-robot cooperation is the primary challenge. The easy scenario requires one fixed two-robot operation. The medium scenario uses a fixed two-robot predecessor followed by a variable-arity operation that admits coalitions of two to four robots. Because productive duration does not depend on coalition size and skill semantics are inactive, a third or fourth robot does not bring any benefit, and thus this operation also tests whether controllers avoid committing more robots than it requires. Roles within a coalition are assigned by the environment’s canonical rule so that controllers can select the coalition composition without separately deciding the robot-to-role assignment. The hard scenario requires a fixed four-robot operation after a two-robot predecessor. The hard scenario uses a WIP cap of one, thereby

making the stricter minimum coalition requirement the primary workload change; the complete scenario also changes the grid size as reported in Table 5.

Concurrency. The same branching recipe is used in all three scenarios: two independent unary branches followed by a pairwise join. Increasing the WIP limit and product release frequency exposes more simultaneous ready work, and the hard scenario also enlarges the grid. The family therefore provides progressively higher-load scenarios in which productive overlap and contention become more prominent. Because WIP, release interval, and, for the hard level, grid size change together, the family is a workload progression.

3.4 Task success and evaluation modes

A reproducible evaluation needs three distinct definitions: when an individual product counts as successful, how the evaluation is bounded, and how performance is measured. Product success is defined as a state condition for one product, while the evaluation mode determines whether the system is observed over a continuing time horizon or until a fixed batch is finished; the performance measures used to compare controllers are defined separately in Section 3.5.

Task success. A product unit z is successful once its terminal recipe output has been produced and delivered through an admissible output interface. The terminal output can be created only after the required recipe path has satisfied its precedence, material-consumption, and exclusive-choice rules; delivery of that output therefore suffices to certify product completion.

Evaluation modes. AssemblyGrid v1 supports two evaluation modes:

- **Continuing production:** products may continue to arrive during a fixed evaluation horizon T . Production rate, offered-demand service, and the fraction of admitted products completed by T are therefore natural outcomes; a single makespan for the continuing stream is not defined. The horizon is an evaluation-protocol parameter and is not part of the environment dynamics. Since all nine official configurations use the same value, it is listed among the shared constants in Table 21.
- **Finite batch:** a fixed product set $\mathcal{Z}$ is released according to the benchmark instance and the evaluation follows that batch until all products are delivered or a declared finite-batch timeout T_B is reached. Product completion times C_z and the batch makespan $C_{\max}$ are then appropriate outcome measures for completed batches; an incomplete batch is reported as timed out together with its delivered and unfinished counts, and $C_{\max}$ is left undefined for that execution.

The per-product success condition is the same in both modes; only the evaluation boundary changes.

Common evaluation framework. AssemblyGrid v1 defines a common set of production outcomes and mechanism diagnostics for all controllers evaluated on the benchmark. Productive concurrency can improve these outcomes by increasing useful parallel activity, while the outcomes themselves remain the primary performance objectives.

3.5 Benchmark measures

Benchmark evaluation should address two questions: how much production was achieved, and which mechanisms enabled or limited that production; AssemblyGrid v1 therefore separates *outcome measures* from *mechanism diagnostics*. Delivered products and throughput are the primary continuing-production outcomes. Demand-facing companions report how many scheduled release opportunities are admitted or lost and what fraction of offered demand is ultimately delivered.

Admitted-product completion, mean WIP, and restricted mean flow time are secondary outcome measures; candidate admission, coalition formation, blocking, concurrency, utilization, occupancy, and geometry quantities are mechanism diagnostics. Unless stated otherwise, T denotes the evaluation horizon in ticks, with the complete metric definitions given in Appendix B.11.

Outcome measures. Let $\mathcal{O}_T$ be the set of scheduled release opportunities at or before horizon T , let $\mathcal{Z}_T = \{z : a_z \leq T\}$ be the product units actually admitted at those opportunities, and let $\mathcal{D}_T = \{z \in \mathcal{Z}_T : C_z \leq T\}$ be those delivered at or before the horizon. In continuing production,

$$\text{Throughput}(T) = \frac{|\mathcal{D}_T|}{T}, \quad \text{Completion}_{\text{adm}}(T) = \frac{|\mathcal{D}_T|}{|\mathcal{Z}_T|} \quad \text{for } |\mathcal{Z}_T| > 0. \quad (11)$$

To make admission selectivity explicit, define

$$\text{ReleaseAcceptance}(T) = \frac{|\mathcal{Z}_T|}{|\mathcal{O}_T|}, \quad \text{DemandFulfillment}(T) = \frac{|\mathcal{D}_T|}{|\mathcal{O}_T|} \quad \text{for } |\mathcal{O}_T| > 0, \quad (12)$$

where $N_{\text{lost}}(T) = |\mathcal{O}_T| - |\mathcal{Z}_T|$ is the number of scheduled releases not admitted. Throughput measures delivered products per tick and is the principal continuing-production measure. $\text{Completion}_{\text{adm}}(T)$ is explicitly the admitted-product completion fraction, not a demand-fulfillment rate. A scheduled release is admitted only when the WIP cap and raw-material availability permit it, otherwise that release opportunity is lost. A controller that clears work faster can therefore admit more products, which changes $|\mathcal{Z}_T|$. Because the release schedule is fixed, these quantities follow from the delivered count, the admitted-product completion fraction, and $|\mathcal{O}_T|$. They prevent a high admitted-product completion fraction from being interpreted as high service of offered demand. Within one official scenario, $|\mathcal{O}_T|$ is fixed by the common release schedule and evaluation horizon, hence demand fulfillment is a constant rescaling of delivered count and need not duplicate every controller table. Cross-scenario interpretation must additionally account for differences in release process, WIP limit, recipe structure, and workload. The same conditioning on admitted products applies to mean WIP and restricted mean flow time below. Let

$$\text{WIP}(t) = |\mathcal{Z}_t \setminus \mathcal{D}_t|, \quad \overline{\text{WIP}}(T) = \frac{1}{T} \sum_{t=1}^T \text{WIP}(t), \quad (13)$$

where $\mathcal{Z}_t$ is the set of products admitted by decision epoch t and $\mathcal{D}_t$ is the subset already delivered. Mean WIP is reported for continuing-production evaluations, and the time-resolved WIP trajectory may additionally be reported when transient congestion or release dynamics are under study.

To account for products that remain unfinished at the horizon, the restricted mean flow time is

$$\text{RMFT}(T) = \frac{1}{|\mathcal{Z}_T|} \sum_{z \in \mathcal{Z}_T} (\hat{C}_z(T) - a_z), \quad (14)$$

where a_z is the release epoch of product z , C_z is its delivery epoch when $C_z \leq T$, and $\hat{C}_z(T) = \min(C_z, T)$ for a delivered product and $\hat{C}_z(T) = T$ otherwise. A product released exactly at T belongs to $\mathcal{Z}_T$ and contributes zero censored age if unfinished, while a product delivered exactly at T belongs to $\mathcal{D}_T$. The difference $C_z - a_z$ counts elapsed environment transitions. In finite-batch mode, a completed batch has makespan $C_{\text{max}} = \max_{z \in \mathcal{Z}} C_z$; timed-out incomplete batches have no reported makespan (Appendix B.11).

Mechanism diagnostics. Let $N_{\text{adm}}(T)$ be the number of execution candidates that acquire all required support and are admitted by horizon T , and let $N_{\text{fail}}(T)$ be the number of nonempty candidate-support attempts that expire or become invalid before admission. Candidates of every arity contribute, including singleton candidates. If all recorded support for an attempt

is withdrawn before admission, the attempt is treated as abandoned and is excluded from both counts; an attempt that retains nonempty support and subsequently expires or becomes invalid contributes to $N_{\text{fail}}(T)$. A nonempty support attempt that is still active when the evaluation horizon is reached is right-censored: it is counted in neither $N_{\text{adm}}(T)$ nor $N_{\text{fail}}(T)$ and is therefore excluded from the admission-success denominator. Because abandoned and right-censored attempts are excluded, this is a resolved admission success rate and is read together with candidate switching and admission latency:

$$\text{AdmSuccess}(T) = \frac{N_{\text{adm}}(T)}{N_{\text{adm}}(T) + N_{\text{fail}}(T)} \quad (15)$$

where the rate is defined only when $N_{\text{adm}}(T) + N_{\text{fail}}(T) > 0$. Admission failures per evaluation horizon are reported as $N_{\text{fail}}(T)$, and admission latency is the mean number of ticks from the first recorded support for an execution candidate to its successful admission, averaged over admitted activities. The coalition success rate $\text{CoalSuccess}(T)$, coalition-formation failures, and coalition-formation latency are the same quantities restricted to candidates with $|U_c| \geq 2$. Every operation in the official COALITION recipes requires at least two robots, thus in that family the admission and coalition quantities coincide and are reported under the coalition names; CONCURRENCY results report the admission quantities, and neither is reported for FLOW, whose official recipes are entirely unary. Geometry diagnostics include geometry-blocked activities and task-level approach/retreat delay. For an admitted operation e , this delay is $D_{\text{approach}}(e) + D_{\text{retreat}}(e)$ ticks and may be accumulated or averaged as a geometry diagnostic. A blocked-activity event is counted when an individually admissible activity with a valid abstract-motion assignment fails to start at a decision epoch because an already committed activity holds a robot it needs, has exhausted a finite-capacity resource it needs, or occupies an overlapping workspace. Picks, bilateral handoffs, deliveries and operation admissions are all counted. Admissions within one epoch are arbitrated in a fixed candidate order, and the counting rule follows that arbitration: for an operation admission, a shared robot, shared exclusive resource or workspace overlap against an operation admitted earlier in the same epoch is treated as simultaneous arbitration and is not counted, whereas exhaustion of a finite-capacity resource is counted even when the exhausting activity was admitted in the same epoch. Logistics activities are counted against every committed activity, including reservations created earlier in the same epoch. Geometry-blocked activities are the subset in which abstract workspace overlap is the only cause, and are reported separately from the robot-blocked and resource-blocked subsets. The count is per blocked activity and start attempt, not per robot pair or conflicting reservation; therefore the same activity blocked over several epochs can contribute more than one event. Additional diagnostics are listed in Appendix B.11.

The remaining diagnostics used explicitly in Section 4 are defined as follows. Let B_t be the number of robots busy in persistent physical or logistics activity at sampled decision epoch t , and let $\mathcal{T}_{\text{samp}}$ be the set of sampled ticks. For the canonical evaluations reported here, $\mathcal{T}_{\text{samp}} = \{0, \dots, T - 1\}$, with these state diagnostics are sampled once per evaluation tick; any alternative diagnostic subsampling must be declared explicitly. Robot occupancy is

$$\text{Occ} = \frac{1}{|\mathcal{T}_{\text{samp}}|} \sum_{t \in \mathcal{T}_{\text{samp}}} \frac{B_t}{MN}, \quad (16)$$

where MN is the total number of robots; accordingly, $\text{Occ} \in [0, 1]$ is the time-average fraction of robots that are busy. For blocked-ready work, let R_t be the set of logically ready operation samples at tick t , and let $F_t \subseteq R_t$ be those with at least one currently feasible coalition assignment. The ready-operation feasible-coalition fraction and blocked-ready-operation rate are

$$\text{RF} = \frac{\sum_{t \in \mathcal{T}_{\text{samp}}} |F_t|}{\sum_{t \in \mathcal{T}_{\text{samp}}} |R_t|}, \quad \text{BR} = 1 - \text{RF}, \quad (17)$$

where the common denominator must be nonzero. Thus, RF is the fraction of ready-operation samples for which at least one feasible coalition is available, whereas BR is the fraction for which none is available. These quantities describe access to a feasible coalition when work is ready; they are not measures of reward, controller failure, or production success. A controller can therefore sustain productive output even when BR is high, as temporary infeasibility may affect some ready-operation occurrences while a feasible subset continues to progress.

Finally, define a no-progress indicator for tick t to be one when WIP is nonzero, no progress event occurs during that tick, and no recipe-operation activity, whether singleton or cooperative, remains active. A progress event is a successful pick, handoff, candidate admission, recipe-operation completion, or delivery. The reported *operational no-progress-tick* diagnostic is the sum of this indicator over the evaluation horizon. The metric is therefore interpreted as an inactivity diagnostic: intentional waiting, temporary idleness, or controller choices may satisfy the condition even when a feasible continuation exists. Due-date tardiness, value-weighted throughput, backlog age, workload balance, and compute cost remain optional extensions or instrumentation and never replace the benchmark outcome measures.

3.6 Productive concurrency

Since concurrency is useful only when it advances production, AssemblyGrid v1 distinguishes robot occupancy from productive operation concurrency: a system may exhibit high occupancy while making little or no product progress. Table 6 summarizes the distinct interpretations.

Table 6: Occupancy, visited-state productive-concurrency capacity, and realized productive concurrency.

Quantity	Question answered	Interpretation
Robot occupancy	How many robots are busy?	Resource-use intensity
Visited-state productive-concurrency capacity	How many mutually compatible recipe-operation activities could productively occupy the next interval from the current state?	Simultaneous productive opportunity in the state reached by the controller
Realized productive concurrency	How many concurrent activities actually advance product progress?	Simultaneous useful progress

State-dependent concurrency capacity. Productive concurrency capacity is state-dependent as the set of operations that can proceed simultaneously changes as products progress, robots become occupied or available, materials move, and resource or geometric conflicts appear or disappear. The coalition-size bound applies only to a single operation and therefore does not determine this system-wide capacity. At any decision epoch, the achievable concurrency is determined by the ready operations and the combinations of them that can execute together without conflict. Because the state s_t is itself reached under a controller’s preceding decisions, averages of this capacity along an execution are *visited-state* quantities.

Concurrency capacity. At decision epoch t , the canonical productive-concurrency capacity is the largest set of mutually compatible recipe-operation activities that could occupy the next execution interval while preserving work already in progress. Let $\mathcal{E}_t^{\text{run}}$ be the committed recipe-operation activities already executing and let $\mathcal{C}_t^{\text{new}}$ be the individually admissible execution candidates in the current state. Nonproductive logistics activities may limit which productive activities can run, but they do not count toward the concurrency capacity. Each candidate $c \in \mathcal{C}_t^{\text{new}}$ corresponds to the prospective activity $e(c)$ that would be created if the candidate were admitted. The activities considered in the capacity calculation are therefore

$$\mathcal{E}_t = \mathcal{E}_t^{\text{run}} \cup \{e(c) : c \in \mathcal{C}_t^{\text{new}}\}. \quad (18)$$

Every member of $\mathcal{E}_t$ is a recipe-operation activity whose successful completion advances product progress. Let $\mathcal{J}_t$ be the conflict-free subsets of $\mathcal{E}_t$ that contain every activity in $\mathcal{E}_t^{\text{run}}$. The canonical count capacity and its realized utilization are therefore

$$K_{\text{prod}}^*(s_t) = \max_{I \in \mathcal{J}_t} |I|, \quad U_{\text{prod}}^{(0)}(t) = \frac{K_{\text{prod},t}^\pi}{K_{\text{prod}}^*(s_t)} \text{ when } K_{\text{prod}}^*(s_t) > 0, \quad (19)$$

where $K_{\text{prod},t}^\pi$ is the number of recipe-operation activities actually executing in that interval under policy π . The statewise quantity $K_{\text{prod}}^*(s_t)$ is controller-independent once s_t is fixed, but its execution-average is generally controller-dependent because different controllers visit different states. Accordingly, cross-controller differences in mean capacity are interpreted descriptively. Two capacity activities are incompatible when they share an exclusive robot or resource, are alternative realizations of the same operation occurrence, or violate the active geometric constraints. Compatibility is therefore not a purely pairwise relation: a resource with finite capacity above one imposes a cardinality constraint on the whole selected set, which the reference implementation enforces against the accumulated selection. Implementation details for the capacity calculation, including exact-search limits and approximation handling, are given in Appendix B.8. Mutually exclusive alternative routes of the same product are also incompatible, ensuring that the capacity does not count routes that cannot be admitted simultaneously.

Productive concurrency versus parallelism. A recipe-operation activity is **productive** when its completion advances the recipe state of at least one product. Once admitted, such an operation remains in this productive class throughout its committed approach, process, and retreat interval; these phases form one indivisible task-level operation in AssemblyGrid v1. Picking, carrying, positioning, and handoff enable production but do not themselves count as productive activities (Appendix B.8). The utilization ratio is evaluated only when $K_{\text{prod}}^*(s_t) > 0$; states with no productive opportunity are excluded from the average and are not assigned a perfect score.

3.7 Conformance checks

Conservative algorithm-independent lower bounds (robot-time, critical-path, and resource) and the eleven benchmark invariants that an implementation consistent with the benchmark must preserve in every reachable state are given in Appendix B (Appendix B.12 for the invariants). Section 4 reports the executable checks that exercise these requirements.

4 Benchmark validation

This section tests whether the reference implementation realizes the mechanisms of Sections 2 and 3 and whether the official scenarios support productive control under the benchmark rules and information structure.

4.1 Evaluation protocol

The evidence has three levels: conformance checks test the declared rules, controlled mechanism studies test the intended effects, and reference controllers characterize productive execution under privileged centralized, structured decentralized, and learned decentralized control. Together, these experiments establish benchmark conformance, mechanism behavior, and executability across distinct controller classes. The experiments address four validation questions.

- **Recipe and material semantics.** Does the reference implementation correctly realize material conservation, sequence, AND, XOR, joins, bilateral handoffs, and terminal delivery under the published recipe definition?

- **Temporary coalition coordination.** Under the declared decentralized observation interface, including the support-fraction cue, do multi-robot operation requirements produce observable coalition-formation behavior and failed formation attempts?
- **Productive concurrency.** Does the benchmark distinguish concurrent *productive* work from robot busyness in a matched deterministic control?
- **Geometric feasibility.** Does the abstract geometry layer classify reach, clearance, and simultaneous-segment conflicts as specified in deterministic controlled cases?

Conformance and mechanism checks use deterministic (fixed and reproducible) test cases together with the conformance checks included in the release. The four non-learning reference controllers are evaluated on the same 50 held-out final-test instances used for learned-policy evaluation; these are unseen realizations of the same scenario generator, not transfers to new layouts, recipes or scales,. The reported MARL campaign comprises exactly $3 \times 3 \times 3 \times 10 = 270$ training runs: independent proximal policy optimization (IPPO), multi-agent proximal policy optimization (MAPPO), and QMIX, a monotonic value-function factorisation method, on the three workload families at three difficulty levels with ten independent algorithm seeds per method and scenario. All MARL methods start from random initialization without demonstrations, behavior cloning, or pretrained weights and train for 1,536,000 environment steps. Periodic training checkpoints were not retained; accordingly, each policy is evaluated at its final training checkpoint on the 50 held-out final-test instances; no post-hoc checkpoint selection is performed, and final-test measurements are not used during training. The exact instance indices and optimization records are provided in Appendix A. Table 7 summarizes the criteria used to assess the campaign.

Table 7: Benchmark-validation criteria.

Validation criterion	Requirement
Benchmark conformance	Formal invariants, recipe semantics, interfaces, and metric boundaries pass executable checks
Privileged productive-execution evidence	The centralized method achieves positive mean deliveries in all nine official scenarios and at least one delivery on every final-test instance of the three COALITION scenarios using global information and centralized action selection
Structured decentralized control	Greedy and Parallel-aware achieve productive execution through decentralized benchmark interfaces on the held-out final-test set
Learned decentralized control	Representative MARL controllers are trained through the common decentralized execution interface with ten independent seeds per scenario and held-out final-test evaluation
Mechanism evidence	Controlled evidence exercises recipe/material, coalition, productive-concurrency, and geometry semantics; workload-level coalition results use the declared support-fraction cue
Reproducibility record	Benchmark version, benchmark-instance definition, random seeds, information access, budget, reward definition, and abstraction level are recorded

The non-learning controllers serve different purposes. Random masked provides a simple feasible-action baseline, Greedy aims to assess if basic decentralized execution is possible, Parallel-aware favors actions that support concurrent progress, and the Centralized controller uses global information to show what can be achieved with centralized decisions. These controllers draw on common ideas from multi-robot task allocation, decentralized allocation, and dispatching [31–33]. Table 8 summarizes their roles, available information, and selection rules.

Table 8: Non-learning validation references.

Reference	Information	Task and operation selection	Routing and tie handling
Random masked	local	uniform sample from the currently legal per-agent masked actions	seeded random choice
Greedy [31, 32]	local	continue the pending candidate unless another enabled candidate has strictly higher visible support; otherwise select the highest-support enabled candidate	move material only along routes that reduce distance to the deterministic staging or output target
Parallel-aware [32, 33]	local	same operation logic as Greedy	among equally progressing routes, prefer the destination with lower locally observed congestion
Centralized	privileged	select a compatible set from the globally admissible execution candidates using weighted independent-set selection	robots outside selected coalitions use the local parallel-aware logistics rule

The three MARL methods share the same decentralized execution information I_t^u and action masks. IPPO uses a local actor and a local observation-based critic [63, 64]. MAPPO uses the same local actor interface and a privileged agent-conditioned global-state critic during training [65]. QMIX uses decentralized local action values combined through a monotonic global-state-conditioned mixer during training [66]. MAPPO and QMIX therefore use centralized information only during training, whereas the Centralized method uses privileged information for action selection during evaluation. All three MARL methods use the same training budget and the same team-level reward terms, with method-specific handling of the support credit described below; detailed network, optimizer, replay, and update settings are reported in Appendix A, Table 17.

Training rewards belong to the algorithm layer and do not enter the benchmark measures. The three MARL methods use the same team-level reward terms, with method-specific handling of the support credit. The team reward is

$$r_t^{\text{team}} = 0.10 \Delta p_t + 2.0 \Delta \text{coal}_t + 3.0 \Delta \text{op}_t + 11.0 \Delta \text{deliv}_t - 0.25 \Delta \text{fail}_t - 0.05 \Delta \text{churn}_t - 0.001, \quad (20)$$

where Δp_t counts progress events (successful picks, handoffs, candidate admissions, operation completions, and deliveries), Δcoal_t counts candidate admissions, which are coalition formations when $|U_c| \geq 2$, Δop_t counts completed operations, Δdeliv_t counts deliveries, Δfail_t counts failed admissions, Δchurn_t counts candidate switching, and every agent receives the same r_t^{team} . Let $c_{t,u}^{\text{sup}} \in \{0, 1\}$ equal one when robot u held no pending support before the step and holds pending support after it, so that a switch between candidates and a support admitted within the same tick both give zero. The training rewards are then

$$r_{t,u}^{\text{IPPO/MAPPO}} = r_t^{\text{team}} + 0.05 c_{t,u}^{\text{sup}}, \quad r_t^{\text{QMIX}} = r_t^{\text{team}} + 0.05 \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} c_{t,u}^{\text{sup}}, \quad (21)$$

where the support credit is agent-specific for IPPO and MAPPO, and a shared team quantity for QMIX. For IPPO and MAPPO it is therefore agent-specific reward shaping; it is small relative to the delivery coefficient and does not change the benchmark objective. The delivery coefficient is the sum of two implementation terms: the algorithm-layer environment reward,

which contributes 1.0 per delivered product since every official recipe declares unit product value and that profile assigns zero weight to its tardiness, motion-energy, safety, and WIP penalty terms, plus a 10.0 milestone term added by the training runner. Delivery has the largest weight; admission and support terms are smaller and cease to accrue once an activity is committed. The reward is used for policy optimization, while evaluation follows the benchmark measures defined in Section 3.5. The MARL results are therefore conditional on this reward specification.

Each independently trained policy is treated as one MARL-reference replicate. Each policy contributes one mean over its 50 final-test instances, and MARL-reference summaries and their uncertainty are computed across the ten independent training replicates for each method and scenario ($n = 10$). By contrast, uncertainty for the deterministic non-learning methods is computed across the 50 benchmark instances, not across training seeds; these two uncertainty sources are therefore reported and interpreted separately, following the empirical-practice recommendations of Henderson et al. [67], Colas et al. [68], and Agarwal et al. [69]. All valid MARL runs, including zero-delivery outcomes, are retained in the aggregate summaries. Runtime or infrastructure failures are reported separately from valid algorithmic outcomes.

4.2 Validation of benchmark mechanisms

The mechanism-level experiments connect the formal benchmark definition to observable benchmark behavior before the workload results are interpreted. They proceed from deterministic recipe and material checks to temporary coalition coordination, then to a matched productive-concurrency control and deterministic geometry cases.

Recipe and material semantics. The conformance evaluation examines whether the implementation preserves the benchmark semantics defined in the specification. The checks cover recipe execution and material provenance, bilateral transfer and exclusivity, coalition formation, information and action restrictions, concurrent execution, geometry constraints, and the separation of training rewards from benchmark reporting. Controlled recipe traces further verify serial, AND, XOR, and joining structures, including correct handoffs and terminal product formation only after a valid recipe path is completed. Appendix Table 18 links the main benchmark invariants to their corresponding executable checks.

Temporary coalition formation.

Deterministic controls realize two-, three-, and four-robot coalitions and preserve pairwise locality, exact matching of the selected execution candidate, the canonical role assignment, and immutable membership during execution. As an interface-feasibility check, all 20 trials at each tested coalition size form in one tick both with and without the aggregate support-fraction cue. These controls show that the coalition mechanism itself does not require the cue in the constructed cases. They do not establish the workload-level causal effect of removing the cue under candidate competition.

The official COALITION scenarios characterize temporary multi-robot coordination under the canonical AssemblyGrid v1 observation interface, which includes the support-fraction cue. Figure 4 therefore reports the two MARL-method diagnostics most directly tied to coalition formation: success rate and failed formation attempts. The broader MARL-reference tables retain a common diagnostic core across workload families, thereby enabling blocking, inactivity, production, and robot use to be compared under one reporting convention; coalition-specific quantities are emphasized only where the corresponding recipes contain multi-robot operations.

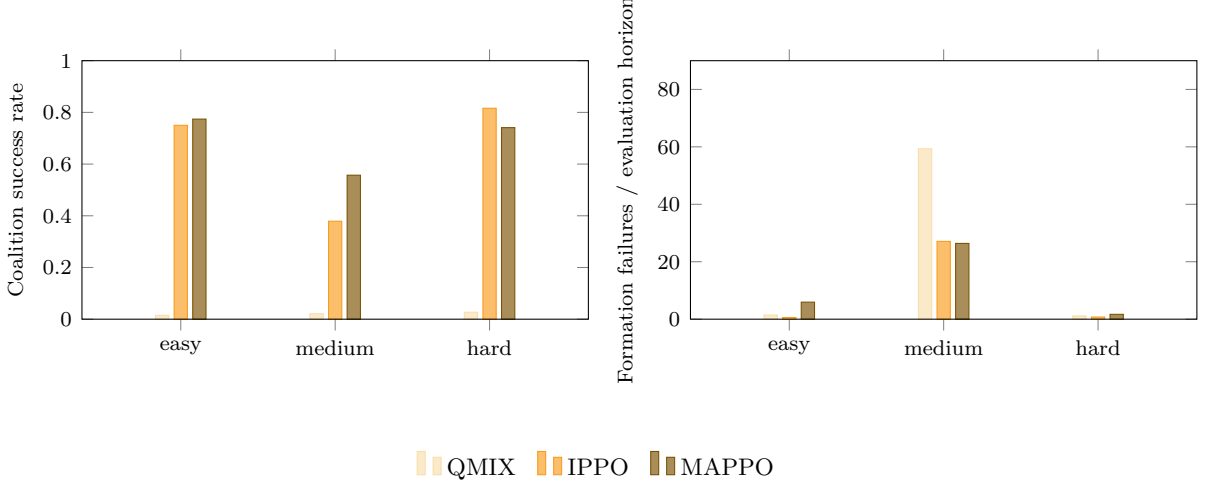


Figure 4: Coalition-formation diagnostics across difficulty levels.

The mechanism diagnostics indicate that coalition behavior differs materially across the three scenarios under the declared support-fraction observation. The MEDIUM scenario produces many more failed formation attempts for IPPO and MAPPO than EASY or HARD, whereas formation success is not monotone in coalition arity. In particular, the WIP cap of one in HARD reduces candidate contention despite its fixed four-robot terminal operation, and MEDIUM permits a variable two-to-four-robot second operation. QMIX exhibits very low coalition success throughout.

Productive concurrency.

A deterministic matched control checks the productive-concurrency semantics independently of controller learning. The operation multiset, durations, precedence, release time, and total productive work are fixed. The serial schedule executes two independent alignment operations consecutively and completes at $C_{\max} = 11$, whereas the productive-parallel schedule allows those same independent alignment operations to overlap and completes at $C_{\max} = 8$, a reduction of 3 ticks or 27.3%. Because the only change in this control is whether the independent productive operations may overlap, the result verifies that the benchmark distinguishes useful simultaneous progress from robot busyness. It does not by itself establish that the three official CONCURRENCY scenarios isolate concurrency as a single causal factor.

Geometric feasibility.

The abstract geometry model is checked through three deterministic cases: a two-robot coalition targeting the midpoint between its members is accepted with minimum reach margin 1.0 and realized duration $D_{\text{approach}} + D_{\text{process}} + D_{\text{retreat}} = 4 + 3 + 1 = 8$ ticks. A singleton target at distance 2.000 against a reach of 1.500 is rejected. Two crossing nominal approach segments are reported as conflicting, whereas two separated segments are not. Together, the deterministic cases demonstrate that the abstract geometry layer classifies the declared task-level reach and conflict conditions as specified.

Matched geometry sensitivity.

A matched intervention then evaluates whether the geometry layer changes the production problem in addition to classifying isolated geometric cases. The production configuration is fixed to a 5×6 grid with a two-robot supported-insertion recipe, WIP cap 4, product release every 5 ticks, and a 400-tick evaluation horizon. For each of 50 random seeds, the same base production realization is evaluated under three matched geometry variants, $\rho_{\text{conf}} \in \{0.12, 0.25, 0.40\}$, with the controller and all remaining settings held fixed. Since the geometry profile is instance-defining, these are matched variants. Increasing ρ_{conf} enlarges the keep-clear region used for simultaneous

abstract motions and therefore changes workspace contention under the declared task-level geometry model.

The graded response for Greedy is reported in Table 9, where the capacity and realized-concurrency entries are arithmetic means of $K_{\text{prod}}^*(s_t)$ and $K_{\text{prod},t}^\pi$ over the states visited at sampled decision epochs. The utilization entry is the mean of $K_{\text{prod},t}^\pi / K_{\text{prod}}^*(s_t)$ over exact opportunity states with positive productive-concurrency capacity, consistent with Section 3.6. Because the controller is fixed within this matched intervention, changes in its mean visited-state capacity are interpreted across geometry variants for that controller, not as a controller-independent scenario constant. As ρ_{conf} rises, geometry-blocked activities increase while productive-concurrency capacity, realized concurrency, deliveries, admitted-product completion, and throughput decline; blocked activities of all causes exceed the geometry-blocked count by about two resource-blocked events per episode, with no robot-blocked events. Utilization stays nearly constant, indicating that the controller keeps using most of the opportunity that remains. The matched geometry sensitivity values below summarize Greedy over 50 matched base production realizations, generated from suite indices 0–49 of the matched control suite and therefore distinct from the final-test indices 1000–1049; the surrounding analysis reports the corresponding endpoint changes for the other structured references.

Table 9: Matched geometry sensitivity.

Metric	$\rho_{\text{conf}} = 0.12$	$\rho_{\text{conf}} = 0.25$	$\rho_{\text{conf}} = 0.40$
Delivered products / evaluation horizon	15.20	14.60	13.94
Admitted-product completion	0.792	0.784	0.776
Throughput	0.038	0.036	0.035
Geometry-blocked activities	267	307	336
Blocked activities, all causes	269	309	338
Mean visited-state concurrency capacity	0.828	0.807	0.774
Mean realized productive concurrency	0.766	0.740	0.707
Concurrency utilization	0.927	0.923	0.916
Robot occupancy	0.189	0.183	0.175

The paired endpoint comparison demonstrates the same mechanism across the three structured controllers. From $\rho_{\text{conf}} = 0.12$ to 0.40, delivered products decrease on average by 1.26, 1.18, and 1.22 for Greedy, Parallel-aware, and Centralized, respectively. Realized productive concurrency decreases by 0.060, 0.037, and 0.068, whereas geometry-blocked activities increase by 69, 70, and 75. All paired 95% intervals over the 50 seeds exclude zero: deliveries $[-1.80, -0.72]$, $[-1.73, -0.63]$ and $[-1.78, -0.67]$; realized productive concurrency $[-0.086, -0.034]$, $[-0.067, -0.007]$ and $[-0.097, -0.039]$; and geometry-blocked activities $[53, 85]$, $[52, 87]$ and $[60, 90]$. The direction is repeated across most matched seed realizations: delivered products decline in 33, 35, and 31 of 50 seed realizations; realized productive concurrency declines in 38, 37, and 39; and geometry-blocked activities increase in 47, 45, and 48. Random masked exhibits no comparable monotonic delivery response (1.22, 1.32, 1.16) and only a small endpoint increase in geometry-blocked activities (18 to 23), consistent with its limited coordinated parallel activity.

The matched intervention supports a mechanism chain in which stronger workspace interference reduces the productive concurrency permitted by the geometry layer, realized concurrency falls with that opportunity, and fewer products are completed within the fixed horizon. The nearly stable utilization ratio separates this loss of geometric opportunity from a large deterioration in how the structured controllers exploit the remaining opportunity.

4.3 Reference behavior across the official workload suite

The deterministic checks above demonstrate that the benchmark mechanisms are executable, after which the controller references provide complementary evidence that the official scenarios admit productive control. Table 10 reports the privileged Centralized method separately to establish attainable productive execution under global information. The MARL campaign then tests the decentralized benchmark interface across all nine scenarios under one common protocol, while Greedy and Parallel-aware provide structured decentralized executability references on the same 50 held-out final-test instances. Table 11 retains the detailed decentralized non-learning delivery record for COALITION, where temporary multi-robot formation is the primary workload mechanism; Random masked supplies a deliberately weak interface reference.

The privileged centralized reference produced a positive mean output in every official scenario over the 50 held-out final-test instances; Table 10 reports the mean delivered products per evaluation horizon and is used only as evidence of centralized productive execution. Its delivery means equal those of Parallel-aware in eight of the nine scenarios. Centralized selects operation candidates globally but uses the Parallel-aware local rule for picks, handoffs, deliveries and idling, causing the two trajectories to coincide whenever the global selection admits the same operations as the local rule. They differ only in COALITION-medium, the one official scenario with a variable-arity operation, where global selection can choose a different coalition size.

Table 10: Privileged Centralized method.

Family	EASY	MEDIUM	HARD
FLOW	21.28	27.68	38.20
COALITION	17.68	15.46	3.86
CONCURRENCY	10.06	17.74	18.50

A positive mean delivery in every official scenario demonstrates productive centralized execution under the declared benchmark dynamics.

Table 11: Non-learning COALITION deliveries.

Reference	EASY	MEDIUM	HARD
Random masked	1.28 ± 0.26	1.26 ± 0.24	0.00 ± 0.00
Greedy	17.34 ± 0.64	14.76 ± 0.54	3.88 ± 0.41
Parallel-aware	17.68 ± 0.65	14.86 ± 0.39	3.86 ± 0.40

Values are mean delivered products per evaluation horizon with 95% normal-approximation interval half-widths $1.96 s / \sqrt{50}$, where s is the sample standard deviation across the 50 benchmark instances; these instance-level intervals are distinct from the MARL-reference uncertainty computed across ten independently trained policies.

Figure 5 shows the three workload families. Each MARL-reference bar is the mean of ten independently trained policies evaluated on the common 50-instance final-test set, whereas Greedy and Parallel-aware summarize those benchmark instances directly and therefore have a different uncertainty source. The privileged Centralized method is reported separately in Table 10, since its purpose is to demonstrate attainable centralized execution. Tables 12–14, the training and optimization figures, and Table 15 provide complementary benchmark and learning diagnostics.

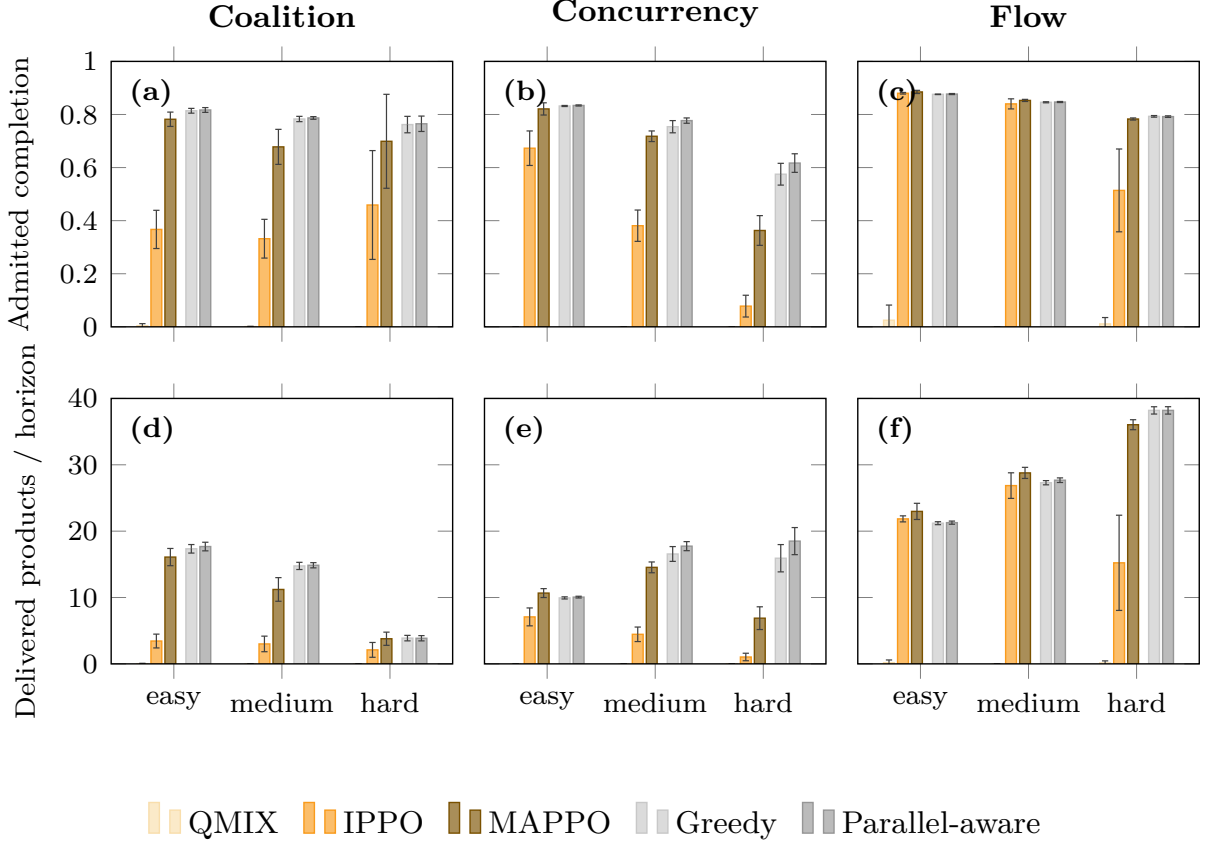


Figure 5: Decentralized reference behavior across workload families.

In Figure 5, whiskers show 95% intervals, across the ten training replicates (Student- t) for the MARL references and across the 50 final-test instances for Greedy and Parallel-aware. The three workload families produce distinct benchmark signatures under the common interfaces. In COALITION, productive execution requires repeated temporary coalition formation; the easy, variable-arity medium, and fixed four-robot hard configurations yield different formation, blocking, and delivery patterns under the declared support-fraction observation. In FLOW, productive MARL execution is obtained across all three levels by the PPO-based methods while the scenario progression jointly changes serial recipe depth, WIP, release pressure, and, at the hard level, grid size. In CONCURRENCY, increasing WIP and release frequency exposes more simultaneously ready work and changes blocked-ready behavior and realized overlap. These workload-family observations are descriptive because the official difficulty levels are complete scenarios with co-varying parameters; causal attribution to any one changed factor requires a matched intervention such as the geometry study above.

These results are interpreted as evidence about the benchmark rather than about the relative merit of the controller algorithms. The structured decentralized methods demonstrate that each workload family supports productive execution under simple task-level rules, while the MARL methods demonstrate that the same local observation and action space can produce nontrivial learned behavior across the official suite. Delivered count and admitted-product completion are read together: delivered count measures absolute production within the fixed horizon, whereas admitted-product completion conditions on the controller-dependent admitted population. The demand-facing definitions in Section 3.5 make scheduled, admitted, and lost releases explicit so that a high completion fraction cannot be mistaken for high service of offered demand. Numerical differences among MARL methods remain descriptive consequences of the fixed reference configurations and are not used to infer an algorithm ordering.

Tables 12–14 complement the outcome figures with a common diagnostic core covering blocking, operational no-progress behavior, production, and robot use. Coalition-formation diagnostics remain central for COALITION. In CONCURRENCY, whose recipe combines two unary branches with a pairwise join, the same counts include singleton admissions and are therefore reported as admission diagnostics (Section 3.5); they are omitted from the FLOW table, as the official FLOW recipes contain only unary operations. Table 13 also reports concurrency diagnostics computed as in Section 4.2: mean visited-state productive-concurrency capacity and mean realized productive concurrency are per-tick means over the states visited by each controller, whereas utilization is the mean ratio over exact-capacity opportunity states and therefore need not equal the ratio of the two means. Because the visited states differ among controllers, mean capacity is not interpreted as a controller-independent property of a scenario. QMIX utilization is undefined in runs without such a state and averages the runs in which it is defined (7, 7, and 5 of 10 for EASY, MEDIUM, and HARD). Capacity was solved exactly at every sampled state except in IPPO HARD and QMIX MEDIUM, where 49 and 112 of the 500 final-test episodes contained approximate-capacity states. The ready-operation feasible-coalition fraction is also omitted since Section 3.5 defines it exactly as $1 - \text{BR}$. MARL-reference uncertainty is computed across independent training replicates, whereas non-learning uncertainty arises from benchmark-instance variation; the two are therefore interpreted separately.

Table 12: COALITION MARL-reference diagnostics.

Metric	Difficulty	QMIX	IPPO	MAPPO
Coalition-formation failures / horizon	easy	1.45	0.59	5.93
	medium	59.31	27.09	26.38
	hard	1.14	0.75	1.69
Coalition success (resolved)	easy	0.015	0.750	0.774
	medium	0.021	0.379	0.557
	hard	0.027	0.816	0.741
Candidate switches / horizon	easy	0.00	0.00	0.00
	medium	225.63	4.39	14.76
	hard	0.00	0.19	0.00
Admission latency, ticks	easy	0.56 [†]	0.16	0.41
	medium	0.71 [†]	0.30	0.63
	hard	0.53 [†]	0.11	0.05 [†]
Blocked-ready-operation rate	easy	0.114	0.828	0.934
	medium	0.192	0.733	0.814
	hard	0.176	0.830	0.762
Operational no-progress ticks, of 400	easy	396.1	327.5	158.8
	medium	379.2	316.6	164.8
	hard	398.3	240.2	219.2
Delivered / horizon	easy	0.03	3.44	16.09
	medium	0.00	3.00	11.21
	hard	0.00	2.11	3.79
Admitted-product completion	easy	0.004	0.367	0.782
	medium	0.001	0.332	0.678
	hard	0.000	0.459	0.699
Robot occupancy	easy	0.003	0.035	0.158
	medium	0.009	0.036	0.128
	hard	0.001	0.036	0.050

[†]Latency is averaged over replicates with at least one admission: QMIX 1, 7 and 1 of 10 in easy, medium and hard, and MAPPO 9 of 10 in hard.

Table 13: CONCURRENCY MARL-reference diagnostics.

Metric	Difficulty	QMIX	IPPO	MAPPO
Admission failures / horizon	easy	1.94	1.07	0.48
	medium	0.00	5.06	10.34
	hard	3.19	2.58	7.06
Admission success	easy	0.298	0.950	0.987
	medium	0.015	0.868	0.848
	hard	0.235	0.866	0.860
Blocked-ready-operation rate	easy	0.100	0.612	0.483
	medium	0.013	0.859	0.884
	hard	0.156	0.904	0.973
Operational no-progress ticks, of 400	easy	389.3	211.7	139.9
	medium	366.7	277.7	92.8
	hard	384.1	319.2	190.9
Delivered / horizon	easy	0.00	7.08	10.67
	medium	0.00	4.45	14.54
	hard	0.00	1.04	6.89
Admitted-product completion	easy	0.000	0.673	0.821
	medium	0.000	0.381	0.718
	hard	0.000	0.078	0.363
Robot occupancy	easy	0.003	0.073	0.104
	medium	0.010	0.052	0.174
	hard	0.004	0.020	0.087
Mean visited-state concurrency capacity	easy	0.919	0.524	0.727
	medium	3.155	0.495	1.114
	hard	0.734	0.580	0.776
Mean realized productive concurrency	easy	0.020	0.472	0.699
	medium	0.000	0.371	1.091
	hard	0.057	0.284	0.766
Concurrency utilization	easy	0.145	0.916	0.965
	medium	0.001	0.875	0.978
	hard	0.202	0.854	0.988

Capacity is exact on 99.6% or more of the evaluated states in every cell except QMIX CONCURRENCY-medium (81.0%); on the remaining states the search exceeds its exact limits and the reported capacity is a greedy lower bound. Utilization is averaged over exact states only.

Table 14: FLOW MARL-reference diagnostics.

Metric	Difficulty	QMIX	IPPO	MAPPO
Blocked-ready-operation rate	easy	0.015	0.006	0.018
	medium	0.003	0.055	0.049
	hard	0.027	0.054	0.038
Operational no-progress ticks, of 400	easy	394.5	156.9	147.8
	medium	392.6	72.4	56.3
	hard	373.8	150.1	21.9
Delivered / horizon	easy	0.18	21.85	22.98
	medium	0.00	26.87	28.78
	hard	0.14	15.23	36.04
Admitted-product completion	easy	0.025	0.880	0.885
	medium	0.000	0.840	0.853
	hard	0.011	0.514	0.783
Robot occupancy	easy	0.005	0.193	0.189
	medium	0.002	0.194	0.194
	hard	0.009	0.108	0.236

4.4 MARL-reference behavior and same-generator held-out evaluation

The training record provides optimization context for the full 270-run MARL campaign over 1.536 million environment steps. Here *milestone return* denotes the logged mean undiscounted milestone return over the rollouts contributing to a recorded training milestone, where each rollout return sums the team terms of Eq. 20 with a delivery coefficient of 10 together with the agent-mean support credit, and excludes the separate **sparse-delivery-v1** environment reward that supplies the remaining delivery unit; it is an optimization diagnostic not a benchmark score. Figures 6 and 7 report the ten-replicate median and interquartile range for milestone return, admissions, and delivery in the COALITION and CONCURRENCY families, whose recipes contain multi-robot operations; admissions are coalition formations in COALITION and also include singleton admissions in CONCURRENCY. Figure 8 reports milestone return and delivery for FLOW; coalition-formation curves are intentionally omitted given the unary structure of the official FLOW recipes. Figures 9, 10, and 11 report actor-critic optimization diagnostics for IPPO and MAPPO, allowing policy and critic learning dynamics to be inspected independently of final production outcomes. All training figures document behavior under the fixed reference protocol and are used for reproducibility and optimization-health context.

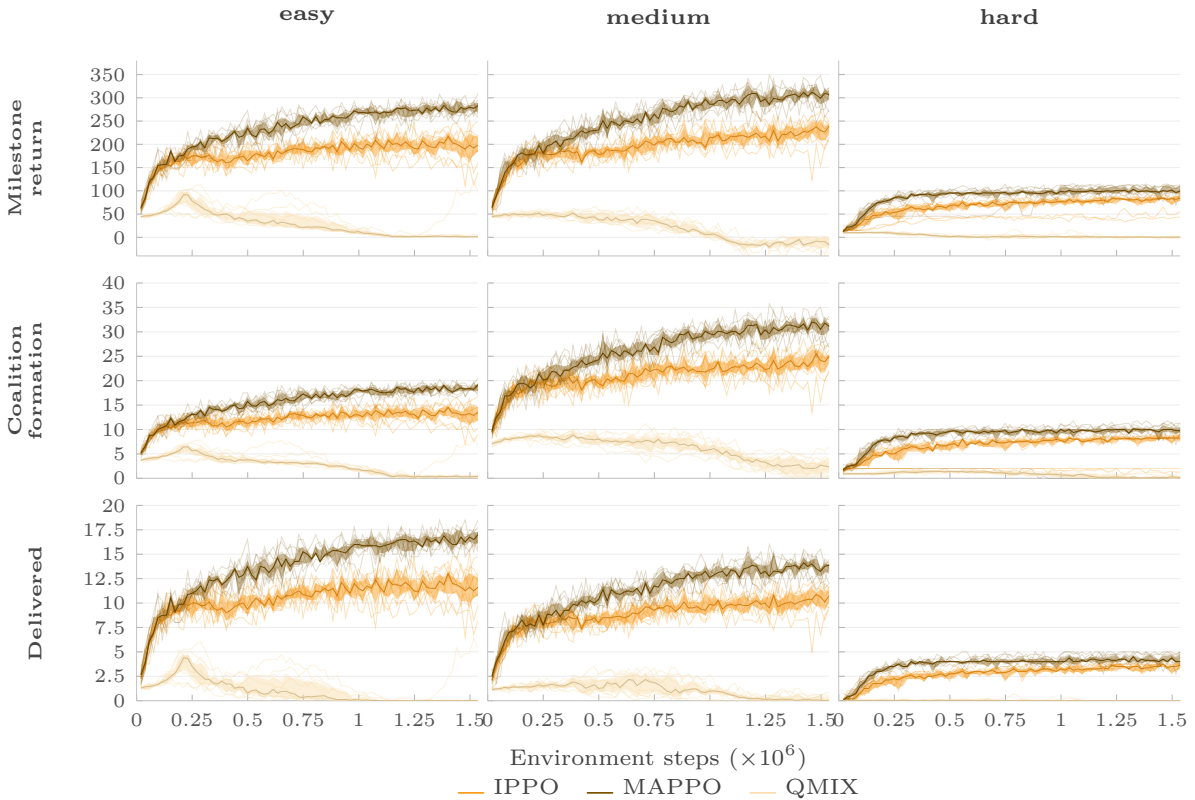


Figure 6: COALITION training curves.

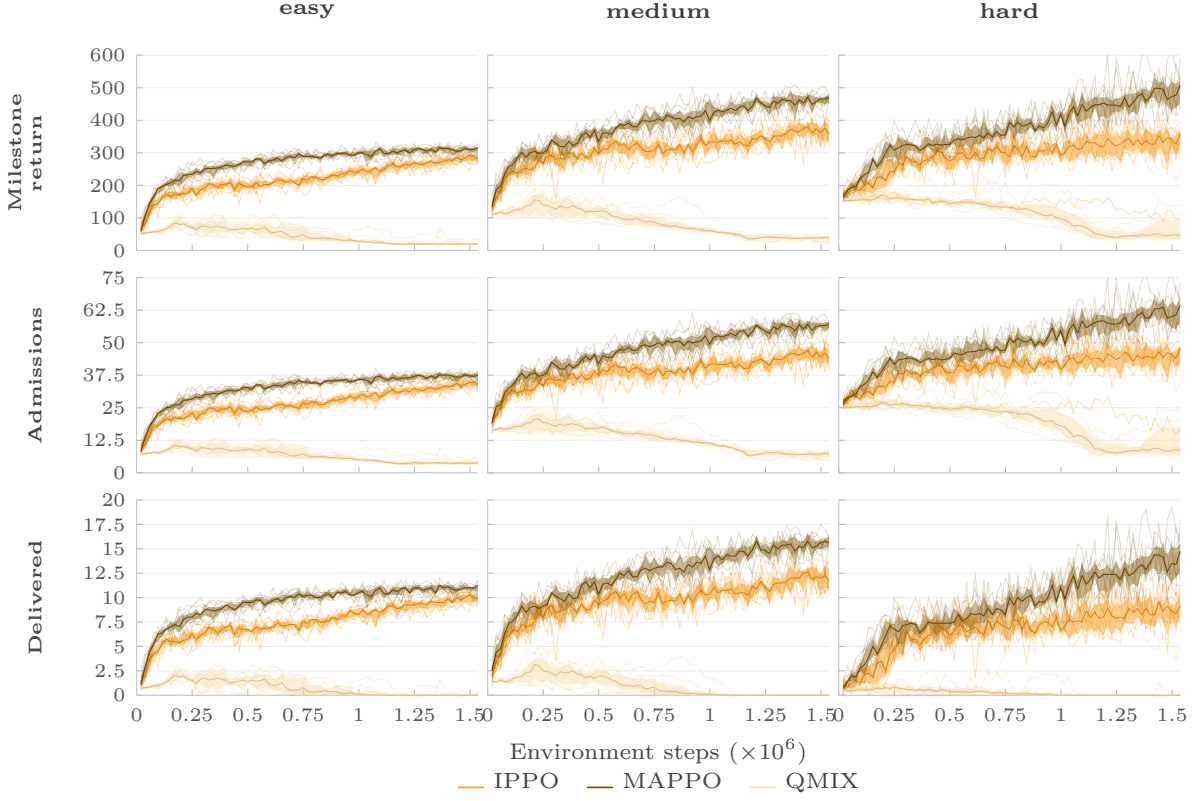


Figure 7: CONCURRENCY training curves.

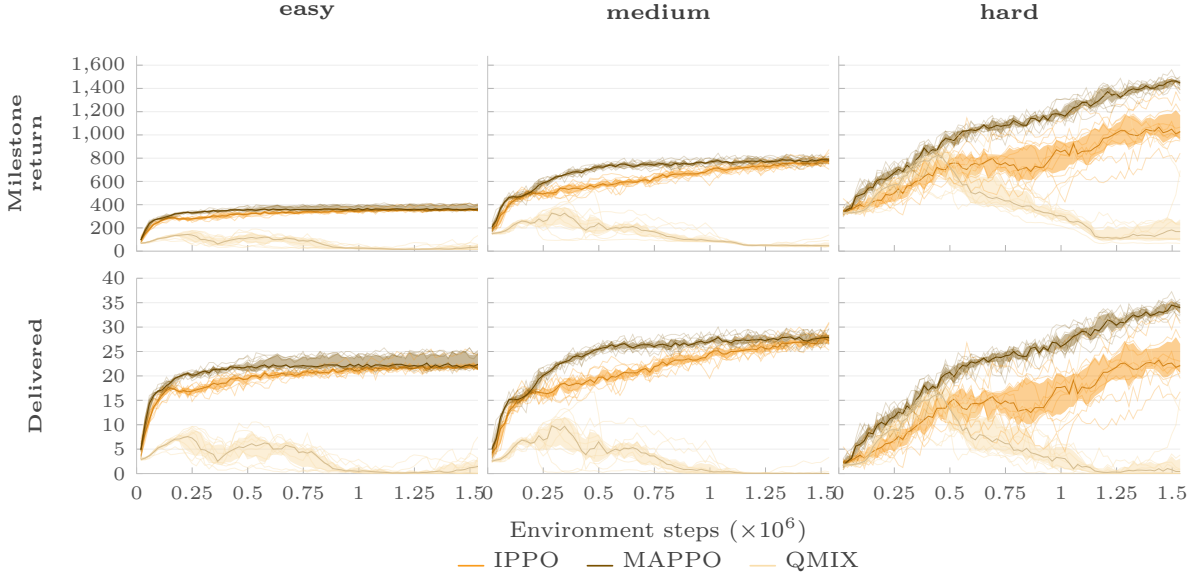


Figure 8: FLOW training curves.

Under the reported implementations, reward specification, exploration settings, and fixed training budget, IPPO and MAPPO develop sustained delivery behavior across all three workload families, although the final operating level remains strongly scenario-dependent. The QMIX reference exhibits transient improvement in several training traces but produces little or no final-test output across the official suite. This pattern characterizes the reported QMIX configuration, including its observation encoding, action representation, exploration schedule, replay settings,

and training budget; it does not support a general claim about QMIX as an algorithm class.

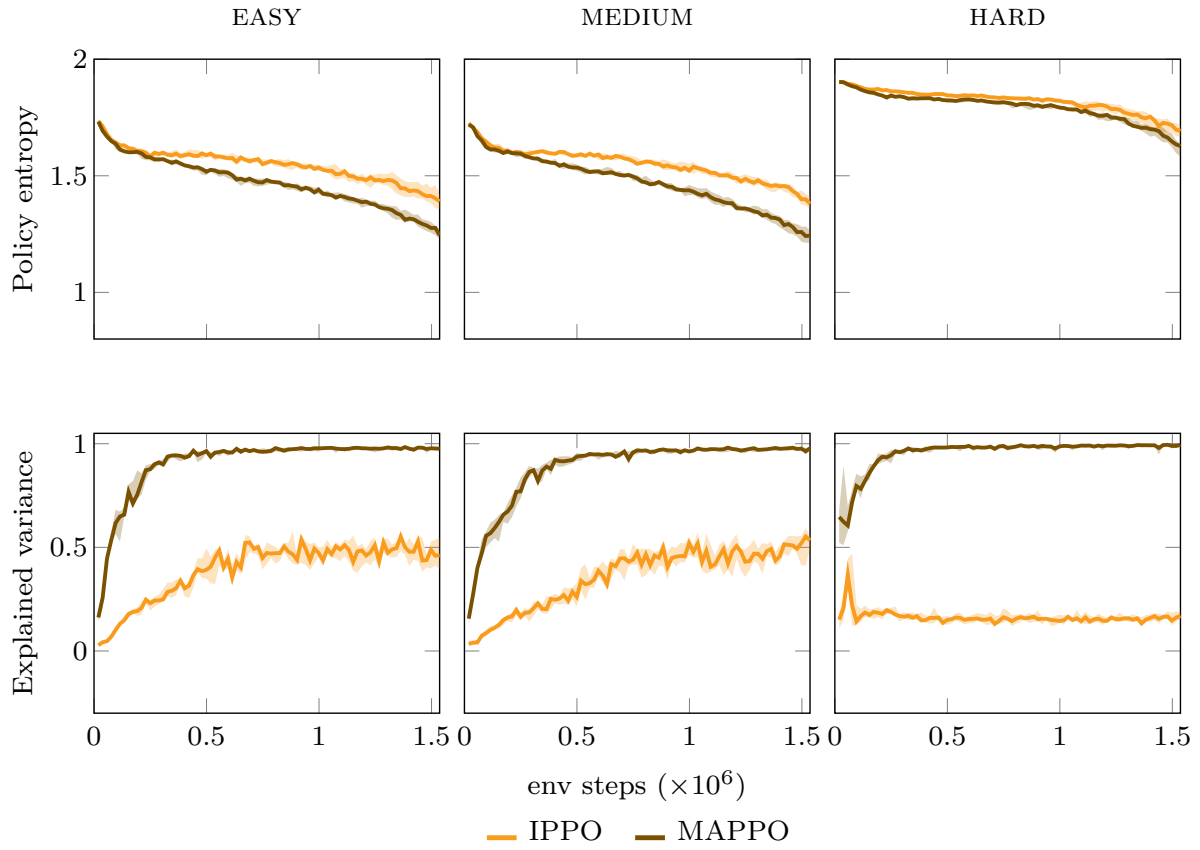


Figure 9: COALITION actor-critic diagnostics.

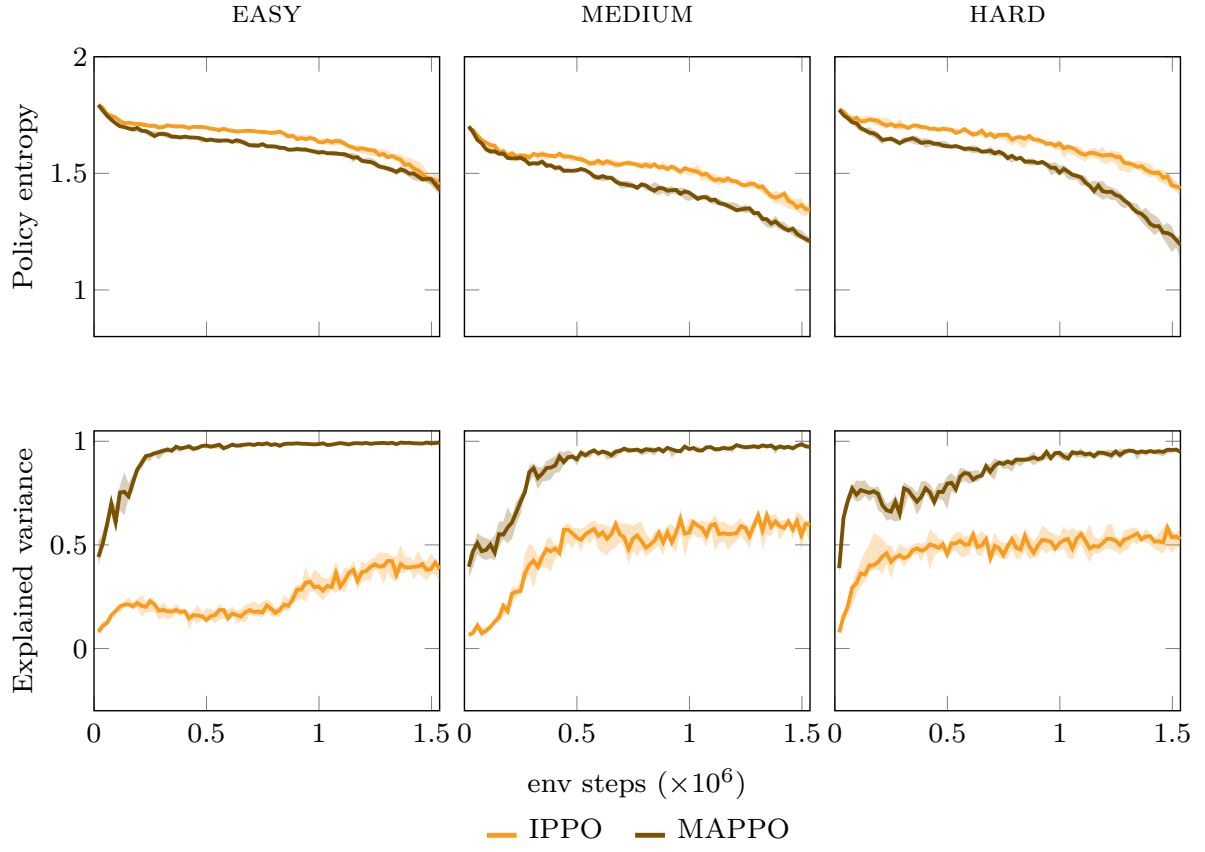


Figure 10: CONCURRENCY actor-critic diagnostics.

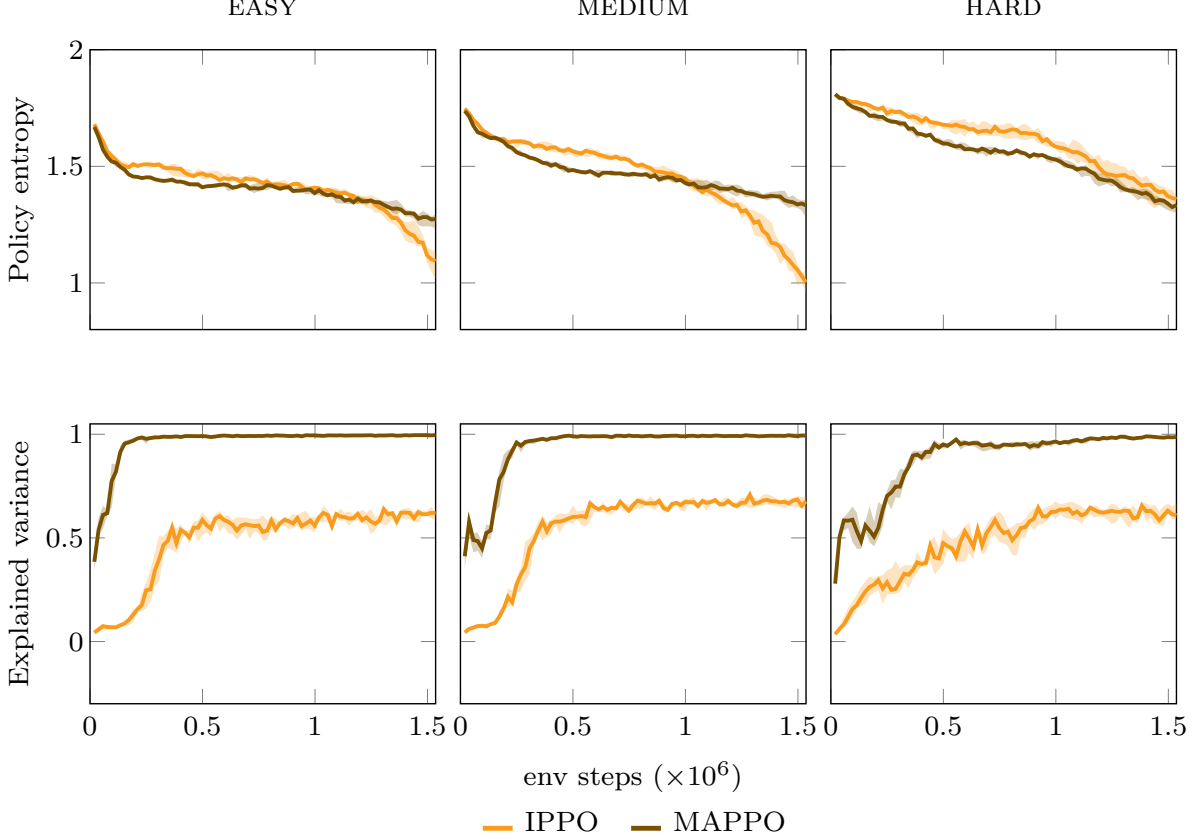


Figure 11: FLOW actor-critic diagnostics.

Policy entropy measures the stochasticity of the decentralized actor, whereas explained variance describes the fraction of variation in the critic target captured by the corresponding value prediction. Across the nine official scenarios, the entropy trajectories of IPPO and MAPPO decrease progressively without collapsing to zero, which is consistent with increasingly concentrated masked action distributions over the visited states while retaining nonzero stochasticity over the reported training budget; since the legal action set is state dependent, raw entropy is interpreted only as an optimization diagnostic. The critic diagnostics separate more strongly: MAPPO reaches high explained variance across the scenarios, whereas IPPO remains markedly lower under its local observation-based critic. The observed distinction is consistent with the different critic-information regimes defined by the two reference implementations, although the diagnostic alone neither identifies the cause of their production differences nor demonstrates algorithmic superiority. Policy entropy and critic explained variance are reported only for IPPO and MAPPO since these actor-critic diagnostics are directly comparable in the logged training record; QMIX uses a different value-decomposition and exploration structure and is therefore characterized through the common training and outcome measures in Figures 6, 7, and 8.

Table 15 consolidates the MARL outcomes for the full official suite. For every family and difficulty level, each entry summarizes ten independently trained final checkpoints, each evaluated on the same 50 held-out instances, and reports the mean delivered count with the 95% Student- t interval half-width $t_{0.975,9} s/\sqrt{10}$, where s is the sample standard deviation of the ten per-policy means.

Table 15: Held-out final-checkpoint deliveries.

Family	Difficulty	QMIX	IPPO	MAPPO
COALITION	easy	0.03 ± 0.06	3.44 ± 1.04	16.09 ± 1.30
	medium	0.00 ± 0.01	3.00 ± 1.17	11.21 ± 1.78
	hard	0.00 ± 0.00	2.11 ± 1.12	3.79 ± 0.99
CONCURRENCY	easy	0.00 ± 0.00	7.08 ± 1.35	10.67 ± 0.67
	medium	0.00 ± 0.00	4.45 ± 1.09	14.54 ± 0.81
	hard	0.00 ± 0.00	1.04 ± 0.57	6.89 ± 1.71
FLOW	easy	0.18 ± 0.41	21.85 ± 0.46	22.98 ± 1.22
	medium	0.00 ± 0.00	26.87 ± 1.93	28.78 ± 0.84
	hard	0.14 ± 0.31	15.23 ± 7.17	36.04 ± 0.75

Held-out deliveries are lower than the late training-curve values in Figures 6, 7, and 8 for IPPO in COALITION easy and medium, CONCURRENCY hard, and FLOW hard, and for MAPPO in CONCURRENCY hard. Training rollouts sample actions from the policy on run-specific training realizations, whereas final-test evaluation selects the highest-scoring legal action on the shared held-out instances. Taken together with the mechanism controls and non-learning methods, these experiments address the four validation questions summarized in Table 16.

Table 16: Summary of validation evidence.

Validation question	Primary evidence	Established result
Recipe and material semantics	203/203 checks and deterministic recipe/material traces	The reference implementation realizes the declared recipe, material, handoff and delivery semantics.
Temporary coalition coordination	deterministic coalition-size controls and COALITION formation diagnostics	Two- to four-robot coalitions execute; under the declared support-fraction observation, the three COALITION scenarios produce distinct formation and failure behavior.
Productive concurrency	matched $C_{\max} = 11$ versus 8 control	Productive concurrent execution is represented separately from robot busyness.
Geometric feasibility	deterministic geometry cases and the 50-instance matched conflict-radius intervention	The geometry layer classifies task-level feasibility as specified, and stronger workspace interference reduces productive concurrency and production under matched conditions.

Taken together, the experiments verify consistency with the benchmark task definition, exercise the declared recipe, coalition, productive-concurrency, and geometry mechanisms at the specified abstraction level, and demonstrate through a matched intervention that geometric interference changes productive opportunity and production. Coalition workload results are conditional on the canonical support-fraction observation, and the official family levels are interpreted as complete workload scenarios. The structured non-learning methods demonstrate productive executability across the official scenarios, while the 270-run MARL campaign characterizes scenario-dependent production, coordination, and optimization behavior under one common protocol.

5 Discussion

This section interprets the results reported in Section 4.

5.1 Benchmark behavior across the official workload suite

The three workload families emphasize different workload structures; accordingly, difficulty labels are meaningful within a family but not across families. The MARL results show that the benchmark supports decentralized policy learning through its local observation and action space: IPPO and MAPPO yield positive mean final-test delivery in every official scenario without centralized action selection at execution time, although 4 of the 90 IPPO replicates and 1 of the 90 MAPPO replicates deliver nothing on the final test. The reported QMIX configuration yields little or no final-test output, with 84 of 90 replicates delivering nothing, which coincided with divergence recorded in the archived training diagnostics: the temporal-difference loss rose more than tenfold in 70 of the 90 runs, and the median absolute mixed value estimate grew by one to two orders of magnitude in every family. In the experiments, QMIX is only implemented via a single-network temporal-difference target with mean-squared-error loss.

5.2 Productive concurrency and geometry

The matched $C_{\max}$ control shows that allowing otherwise identical independent operations to overlap has a concrete effect on completion time. The conflict-radius intervention indicates how geometry reduces productive opportunity for a fixed controller: stronger workspace interference increases geometry-blocked activities, lowers visited-state capacity and realized concurrency, and reduces deliveries for all three structured controllers, while utilization barely changes. These matched controls support the stated mechanism interpretations. By contrast, the easy, medium, and hard CONCURRENCY scenarios are not a one-factor intervention because their load parameters, and at the hard level grid size, co-vary.

5.3 Scope and limitations

The evaluation covers all three official workload families, with the MARL experiments spanning all nine scenarios. AssemblyGrid v1 works at the task level and abstracts lower-level robot details such as inverse kinematics, joint limits, dynamics, and grasp physics. Its geometry model instead captures factors such as reach, clearance, motion time, and workspace conflicts. The main assumptions are listed in Appendix B.9. Coalition results use the standard decentralized observation, including the support-fraction cue, which is kept fixed in the current experiments. Broader settings, such as different layouts, heterogeneous robots, sensing uncertainty, and alternative coordination mechanisms, are left for future extensions. All three MARL methods are trained from scratch for 1,536,000 steps using the reference settings and reward defined in Section 4. The reported results therefore reflect these particular training settings and budget, and different architectures, tuning, or credit-assignment methods could lead to different outcomes. Training uses five realizations per scenario, while evaluation uses 50 unseen realizations from the same generator. The results therefore show generalization within the same scenario.

5.4 Future research enabled by AssemblyGrid v1

Beyond learning, the same environment supports controlled production-system studies in which topology, workspace overlap, routing redundancy, robot density, product mix and coalition-capable regions are varied with controller assumptions fixed. Future controller studies may extend the MARL-reference characterization with method-specific tuning, recurrent or attention-based architectures, and alternative credit assignment, or evaluate how learned MARL policies respond to declared geometry changes using the same matched conflict-radius intervention. Broader extensions should introduce one explicitly defined source of variation at a time. Examples include an explicit communication channel with specified range, bandwidth, latency, loss, and cost; standard generalization splits over recipes, geometry, product mix, teammates, and scale;

disturbances such as robot downtime, blocked workspaces, stockouts, and duration noise; higher-detail geometry with robot-model kinematics and collision checking; and richer process models with rework, rollback, or scrap. Treating these as named extensions keeps the AssemblyGrid v1 workload families, information rules, feasibility conditions, and metrics stable for comparison.

6 Conclusion

AssemblyGrid v1 provides a reproducible task definition for decentralized multi-robot production in which explicit recipe and material progression, temporary coalitions, bounded local information that includes a declared support-fraction coordination cue, productive concurrency, and geometry-dependent admissibility interact within a common environment. The reference implementation satisfies the benchmark conformance checks and realizes the intended recipe, coalition, concurrency, and geometry mechanisms, while the matched geometry intervention supports that workspace interference reduces productive opportunity and production. Across the official suite, the centralized and structured decentralized references achieve productive execution, and IPPO and MAPPO yield positive mean final-test delivery, under distinct information and decision regimes, demonstrating the validity of the proposed benchmark. Physical robot validity, perception, alternative coordination interfaces, learned communication, robustness, and transfer are beyond the scope of this research.

Data and code availability

The AssemblyGrid v1 reference implementation, the nine-scenario official realization catalogue (`assemblygrid-official-realizations-v4`, with the pre-v1 geometry family kept separately as a historical extension), the JSON schemas, the invariant and conformance test suite, the four non-learning reference controllers, the MARL campaign training code, including the shared actor network and the campaign runners of Table 17, and the frozen MARL campaign `aag046-official-suite-canonical-v1` (270 runs: IPPO, MAPPO, and QMIX $\times$ the three workload families $\times$ three difficulty levels $\times$ ten seeds, 1,536,000 environment steps each) are released under version tag `v1.0.0`. The benchmark repository contains the implementation, the catalogues, the schemas, the conformance suite, the reference controllers and the campaign training code; the frozen 270-run campaign records are deposited separately as an archived dataset whose SHA-256 manifest binds every reported table and figure value to its source record. The code is released under the MIT licence. The repository is available at https://github.com/Fouad-Bahrpeyma/AssemblyGrid_v1, and the versioned software release is archived at <https://doi.org/10.5281/zenodo.22734115>.

References

- [1] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. Van Brussel. Reconfigurable manufacturing systems. *CIRP Annals*, 48(2):527–540, 1999. doi:10.1016/S0007-8506(07)63232-6.
- [2] H. A. ElMaraghy. Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems*, 17(4):261–276, 2005. doi:10.1007/s10696-006-9028-7.
- [3] M. Schönemann, C. Herrmann, P. Greschke, and S. Thiede. Simulation of matrix-structured manufacturing systems. *Journal of Manufacturing Systems*, 37:104–112, 2015. doi:10.1016/j.jmsy.2015.09.002.
- [4] M. Wang, Y. Tong, C. Zhuang, and X. Du. Towards Industry 5.0: A framework of reconfigurable matrix-structured manufacturing system. *Journal of Manufacturing Systems*, 82:114–136, 2025. doi:10.1016/j.jmsy.2025.06.002.
- [5] G. Hüttemann, A. F. Buckhorst, and R. H. Schmitt. Modeling and assessing line-less mobile assembly systems. *Procedia CIRP*, 81:724–729, 2019. doi:10.1016/j.procir.2019.03.184.
- [6] R. H. Schmitt, G. Hüttemann, and S. Münker. A priori performance assessment of line-less mobile assembly systems. *CIRP Annals*, 70(1):389–392, 2021. doi:10.1016/j.cirp.2021.04.059.
- [7] A. Avhad, C. Schou, and O. Madsen. A framework for multi-robot control in execution of a Swarm Production System. *Computers in Industry*, 151:103981, 2023. doi:10.1016/j.compind.2023.103981.
- [8] A. Avhad, H. Arnarson, C. Schou, and O. Madsen. Implementing Swarm Production System with multi-robot simulation. *Procedia Computer Science*, 232:934–945, 2024. doi:10.1016/j.procs.2024.01.093.
- [9] C. P. Nielsen, A. Avhad, C. Schou, and E. Ribeiro da Silva. Control system architecture for matrix-structured manufacturing systems. *Computers in Industry*, 146:103851, 2023. doi:10.1016/j.compind.2023.103851.
- [10] W. Li, X. Liu, X. Du, H. Shao, T. Zhu, S. Yang, and X. Yang. Matrix-structured manufacturing systems: literature review and future directions. *International Journal of Production Research*, pages 1–36, 2026. doi:10.1080/00207543.2026.2669883.
- [11] P. Schumacher, C. Weckenborg, Y. Graupner, and T. S. Spengler. Matrix-structured manufacturing systems: integrated planning of design and material supply. *International Journal of Production Research*, 2026. doi:10.1080/00207543.2026.2644564.
- [12] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, 2017.
- [13] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.
- [14] B. Ellis, J. Cook, S. Moalla, M. Samvelyan, M. Sun, A. Mahajan, J. N. Foerster, and S. Whiteson. SMACv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 37567–37593, 2023.

- [15] N. Bard, J. N. Foerster, S. Chandar, et al. The Hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 280:103216, 2020. doi:10.1016/j.artint.2019.103216.
- [16] K. Kurach, A. Raichuk, P. Stańczyk, et al. Google Research Football: A novel reinforcement learning environment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4501–4510, 2020. doi:10.1609/aaai.v34i04.5878.
- [17] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, and Y. Yu. MAgent: A many-agent reinforcement learning platform for artificial collective intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1):8222–8223, 2018. doi:10.1609/aaai.v32i1.11371.
- [18] F. Christianos, L. Schäfer, and S. V. Albrecht. Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 10707–10717, 2020.
- [19] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.
- [20] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, and G. Spigler. Flatland-RL: Multi-agent reinforcement learning on trains. arXiv:2012.05893, 2020.
- [21] M. Carroll, R. Shah, M. K. Ho, T. L. Griffiths, S. A. Seshia, P. Abbeel, and A. D. Dragan. On the utility of learning about humans for human-AI coordination. In *Advances in Neural Information Processing Systems*, volume 32, pages 5175–5186, 2019.
- [22] C. Ruhdorfer, M. Bortoletto, A. Penzkofer, and A. Bulling. The Overcooked Generalization Challenge. arXiv:2406.17949, 2024.
- [23] T. Gessler, T. Dizdarevic, A. Calinescu, B. Ellis, A. Lupu, and J. N. Foerster. OvercookedV2: Rethinking Overcooked for zero-shot coordination. In *International Conference on Learning Representations*, 2025.
- [24] J. Z. Leibo, E. A. Duenez-Guzman, A. Vezhnevets, J. P. Agapiou, P. Sunehag, R. Koster, J. Matyas, C. Beattie, I. Mordatch, and T. Graepel. Scalable evaluation of multi-agent reinforcement learning with Melting Pot. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of PMLR, pages 6187–6199, 2021.
- [25] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, N. Williams, Y. Lokesh, and P. Ravi. PettingZoo: Gym for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.
- [26] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok. VMAS: A vectorized multi-agent simulator for collective robot learning. In *Proceedings of the 16th International Symposium on Distributed Autonomous Robotic Systems*, pages 42–56. Springer, 2022. doi:10.1007/978-3-031-51497-5_4.
- [27] M. Lechner, L. Yin, T. Seyde, T.-H. J. Wang, W. Xiao, R. Hasani, J. Rountree, and D. Rus. Gigastep—One billion steps per second multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi:10.52202/075280-0009.
- [28] A. Rutherford, B. Ellis, M. Gallici, J. Cook, A. Lupu, G. Ingvarsson, T. Willi, R. Hammond, A. Khan, C. Schroeder de Witt, et al. JaxMARL: Multi-agent RL environments and algorithms in JAX. In *Advances in Neural Information Processing Systems*, volume 37, Datasets and Benchmarks Track, 2024. doi:10.52202/079017-1612.

- [29] S. Hu, Y. Zhong, M. Gao, W. Wang, H. Dong, X. Liang, Z. Li, X. Chang, and Y. Yang. MARLlib: A scalable and efficient multi-agent reinforcement learning library. *Journal of Machine Learning Research*, 24(315):1–23, 2023.
- [30] M. Bettini, A. Prorok, and V. Moens. BenchMARL: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024.
- [31] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004. doi:10.1177/0278364904045564.
- [32] E. Nunes, M. Manner, H. Mitiche, and M. Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, 2017. doi:10.1016/j.robot.2016.10.008.
- [33] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006. doi:10.1109/JPROC.2006.876939.
- [34] H. Chakraa, F. Guérin, E. Leclercq, and D. Lefebvre. Optimization techniques for multi-robot task allocation problems: Review on the state of the art. *Robotics and Autonomous Systems*, 168:104492, 2023. doi:10.1016/j.robot.2023.104492.
- [35] S. Alirezazadeh and L. A. Alexandre. A survey on task allocation and scheduling in robotic network systems. *IEEE Internet of Things Journal*, 12(2):1484–1508, 2025. doi:10.1109/JIOT.2024.3491944.
- [36] L. Vig and J. A. Adams. Multi-robot coalition formation. *IEEE Transactions on Robotics*, 22(4):637–649, 2006. doi:10.1109/TRO.2006.878948.
- [37] K. Arjun, D. Parlevliet, H. Wang, and A. Yazdani. Optimizing coalition formation strategies for scalable multi-robot task allocation: A comprehensive survey of methods and mechanisms. *Robotics*, 14(7):93, 2025. doi:10.3390/robotics14070093.
- [38] A. Verma, A. Gautam, A. Dutta, V. S. Shekhawat, et al. CF-HMRTA: Coalition formation for heterogeneous multi-robot task allocation. *Journal of Intelligent & Robotic Systems*, 111:81, 2025. doi:10.1007/s10846-025-02287-4.
- [39] F. Bahrpeyma and D. Reichelt. A review of the applications of multi-agent reinforcement learning in smart factories. *Frontiers in Robotics and AI*, 9:1027340, 2022. doi:10.3389/frobt.2022.1027340.
- [40] W. Xu, J. Gu, W. Zhang, M. Gen, and H. Ohwada. Multi-agent reinforcement learning for flexible shop scheduling problem: a survey. *Frontiers in Industrial Engineering*, 3:1611512, 2025. doi:10.3389/fieng.2025.1611512.
- [41] T. Zhou, D. Tang, H. Zhu, and Z. Zhang. Multi-agent reinforcement learning for online scheduling in smart factories. *Robotics and Computer-Integrated Manufacturing*, 72:102202, 2021. doi:10.1016/j.rcim.2021.102202.
- [42] L. Kaven, P. Huke, A. Göppert, and R. H. Schmitt. Multi agent reinforcement learning for online layout planning and scheduling in flexible assembly systems. *Journal of Intelligent Manufacturing*, 35:3917–3936, 2024. doi:10.1007/s10845-023-02309-8.
- [43] P. Burggräf, J. Wagner, T. Saßmannshausen, D. Ohrndorf, and K. Subramani. Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling. *Procedia CIRP*, 112:57–62, 2022. doi:10.1016/j.procir.2022.09.024.

- [44] J.-D. Zhang, Z. He, W.-H. Chan, and C.-Y. Chow. DeepMAG: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. *Knowledge-Based Systems*, 259:110083, 2023. doi:10.1016/j.knosys.2022.110083.
- [45] L. Zhang, Y. Yan, C. Yang, and Y. Hu. Dynamic flexible job-shop scheduling by multi-agent reinforcement learning with reward-shaping. *Advanced Engineering Informatics*, 62:102872, 2024. doi:10.1016/j.aei.2024.102872.
- [46] J. Jang, D. Klabjan, H. Liu, N. S. Patel, X. Li, B. Ananthanarayanan, H. Dauod, and T.-H. Juang. Scalable multi-agent reinforcement learning for factory-wide dynamic scheduling in semiconductor manufacturing. *Engineering Applications of Artificial Intelligence*, 161:112168, 2025. doi:10.1016/j.engappai.2025.112168.
- [47] K. Bhatta and Q. Chang. Train small, deploy large: Scaling multi-agent reinforcement learning for multi-stage manufacturing lines. *Journal of Manufacturing Systems*, 81:155–168, 2025. doi:10.1016/j.jmsy.2025.04.017.
- [48] L. Lv, J. Fan, C. Zhang, and W. Shen. A multi-agent reinforcement learning based scheduling strategy for flexible job shops under machine breakdowns. *Robotics and Computer-Integrated Manufacturing*, 93:102923, 2025. doi:10.1016/j.rcim.2024.102923.
- [49] Z. Qin and Y. Lu. Knowledge graph-enhanced multi-agent reinforcement learning for adaptive scheduling in smart manufacturing. *Journal of Intelligent Manufacturing*, 36:5943–5966, 2025. doi:10.1007/s10845-024-02494-0.
- [50] G. Boschetti, M. Bottin, M. Faccio, and R. Minto. Multi-robot multi-operator collaborative assembly systems: a performance evaluation model. *Journal of Intelligent Manufacturing*, 32:1455–1470, 2021. doi:10.1007/s10845-020-01714-7.
- [51] A. Keshvarparast, D. Battini, O. Battaia, and A. Pirayesh. Collaborative robots in manufacturing and assembly systems: literature review and future research agenda. *Journal of Intelligent Manufacturing*, 35:2065–2118, 2024. doi:10.1007/s10845-023-02137-w.
- [52] Z. Feng, G. Hu, Y. Sun, and J. Soon. An overview of collaborative robotic manipulation in multi-robot systems. *Annual Reviews in Control*, 49:113–127, 2020. doi:10.1016/j.arcontrol.2020.02.002.
- [53] J. A. Marvel, R. Bostelman, and J. Falco. Multi-robot assembly strategies and metrics. *ACM Computing Surveys*, 51(1), Article 14, 2018. doi:10.1145/3150225.
- [54] Y. Jiang, Z. Huang, B. Yang, and W. Yang. A review of robotic assembly strategies for the full operation procedure: planning, execution and evaluation. *Robotics and Computer-Integrated Manufacturing*, 78:102366, 2022. doi:10.1016/j.rcim.2022.102366.
- [55] J. Chen, J. Li, Y. Huang, C. Garrett, D. Sun, C. Fan, A. Hofmann, C. Mueller, S. Koenig, and B. C. Williams. Cooperative task and motion planning for multi-arm assembly systems. arXiv:2203.02475, 2022.
- [56] J. Collins, M. Robson, J. Yamada, M. Sridharan, K. Janik, and I. Posner. RAMP: A benchmark for evaluating robotic assembly manipulation and planning. *IEEE Robotics and Automation Letters*, 9(1):9–16, 2024. doi:10.1109/LRA.2023.3330611.
- [57] Y. Qin, L. Kang, X. Song, Z. Yin, X. Liu, X. Liu, R. Zhang, and L. Bai. RoboFactory: Exploring embodied agent collaboration with compositional constraints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10075–10085, 2025.

- [58] K. Brown, D. M. Asmar, M. Schwager, and M. J. Kochenderfer. Large-scale multi-robot assembly planning for autonomous manufacturing. *Robotics and Autonomous Systems*, 194:105179, 2025. doi:10.1016/j.robot.2025.105179.
- [59] P. Huang, R. Liu, S. Aggarwal, C. Liu, and J. Li. APEX-MR: Multi-Robot Asynchronous Planning and Execution for Cooperative Assembly. In *Robotics: Science and Systems (RSS)*, 2025.
- [60] Y. Tian, J. Jacob, Y. Huang, J. Zhao, E. L. Gu, P. Ma, A. Zhang, F. Javid, B. Romero, S. Chitta, S. Sueda, H. Li, and W. Matusik. Fabrica: Dual-Arm Assembly of General Multi-Part Objects via Integrated Planning and Learning. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 406–419, 2025.
- [61] F. J. Huertos, O. Bañales, P. Alvarez, and I. Cabanes. Multi-Agent Reinforcement Learning for Cooperative Manipulation in Industrial Robotics: A Systematic Review of Trends, Gaps and Research Drivers. *Robotics*, 15(8):156, 2026. doi:10.3390/robotics15080156.
- [62] F. A. Oliehoek and C. Amato. *A Concise Introduction to Decentralized POMDPs*. Springer, 2016. doi:10.1007/978-3-319-28929-8.
- [63] C. Schroeder de Witt, T. Gupta, D. Makoviichuk, V. Makoviyshuk, P. H. S. Torr, M. Sun, and S. Whiteson. Is independent learning all you need in the StarCraft Multi-Agent Challenge? arXiv:2011.09533, 2020.
- [64] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv:1707.06347, 2017.
- [65] C. Yu, A. Velu, E. Vinitisky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of PPO in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.
- [66] T. Rashid, M. Samvelyan, C. Schroeder de Witt, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of PMLR, pages 4295–4304, 2018.
- [67] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018. doi:10.1609/aaai.v32i1.11694.
- [68] C. Colas, O. Sigaud, and P.-Y. Oudeyer. How many random seeds? Statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.
- [69] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

A Reproducibility, versioning, and reference implementation

Appendix A records release-level information needed to reproduce and interpret the benchmark without changing the AssemblyGrid v1 task definition. The appendix separates the abstraction-level definition, experimental archive, and software-interface details from the formal benchmark definition that follows.

A.1 Geometry profile and release identity

The released recipe schema is `assemblygrid-recipe-v1`, and the default geometry profile is `abstract-v1`. A dimensional mapping of the normalized reference quantities to a particular manipulator, including a UR10 interpretation, does not change the AssemblyGrid v1 task-level abstraction. Evaluations incorporating robot-model inverse kinematics, joint-limit checks, synchronized link-level collision checking, dynamics, contact, or hardware execution should be reported as higher-detail extensions.

A.2 Release identifiers used for reproducibility

The main paper uses semantic names instead of software identifiers. The released implementation uses the following identifiers where exact software traceability is required: candidate-support fraction `proposal_support`, admitted-product completion fraction `completion_rate`, admission success rate `coalition_success_rate`, admission failures `team_formation_failures`, the legacy software key `deadlock_ticks` for the operational no-progress-tick diagnostic, the legacy aggregate blocking keys `trajectory_conflicts` and `geometry_conflict_count` together with the cause-specific keys `geometry_block_count`, `resource_block_count` and `robot_block_count`, the archived count-capacity key `K_feasible_star` for the canonical K_{prod}^* quantity (the implementation also exports `productive_concurrency_capacity_mean` as the semantic alias), Random masked reference `random_feasible`, shared actor class `StructuredActor`, and the fixed MARL training reward `coalition_milestone_v1`. The semantic reward terms Δ_{coal_t} and c_t^{sup} correspond to the legacy implementation concepts for candidate-admission events and proposal-support credit, respectively. These identifiers are implementation details and do not alter the corresponding benchmark definitions.

The official scenario recipes are implemented under the following identifiers: FLOW easy, medium, and hard use `solo_inspection`, `flow_serial_3`, and `flow_serial_5`; COALITION easy, medium, and hard use `pair_assembly`, `supported_insert`, and `standard_abc`; and all three CONCURRENCY difficulty levels use `parallel_branch_light`, with the corresponding semantic recipe structures given in Table 5.

A.3 Experimental archive and evaluation records

The reported controller campaign uses five training realizations, five disjoint validation instances (indices 200–204), and 50 held-out final-test instances (indices 1000–1049) in every scenario. Within each scenario, the scenario configuration, generator, and active geometry profile are held fixed across these splits. The two evaluation splits are the canonical suite instances, generated once from the released instance index and therefore identical across every run, method, and algorithm seed. The training split is constructed differently: each run takes the scenario parameters of suite indices 0–4 and regenerates five realizations under run-specific generation and execution seeds derived from the algorithm seed, therefore every replicate trains on its own five realizations of the same scenario. The training budget is divided evenly across those five realizations, which are trained as consecutive blocks in a fixed order. A realization changes only the generation and execution seeds: within a scenario the grid, topology, recipe graph, WIP limit, release interval, geometry profile and horizon are fixed, and the generation seed determines only which additional input robots may reach each shelf, which shelf row supplies each raw material, and the product masses and due slack. Due slack does not enter v1 dynamics; product mass enters payload feasibility, which never binds under the official mass and payload ranges, and it appears in the privileged global state. Training and final-test realizations are therefore drawn from the same generator with the same scenario parameters and differ only in their seeds; accordingly, a policy always trains on the same layout and recipe and sees five distinct access and material-source patterns, each revisited over many episodes of the 1,536,000-step

budget. This matched split was chosen to give each run its own reproducible five-realization training set per scenario. The same final-test instances are used for IPPO, MAPPO, QMIX, and the reported non-learning methods. Periodic training checkpoints were not retained; the reported policies are therefore evaluated at their final training checkpoint. The validation split was used only for online monitoring during training, not for early stopping, reward adjustment, post-hoc model selection, or final-test selection. The final-test figures reported in Section 4.3 and Table 15 are the per-run final-checkpoint evaluation means recorded in each run’s result record, `aag042_result.json`.

The MARL campaign contains 270 completed training runs, comprising ten independent statistical replicates for each method-scenario configuration: IPPO, MAPPO, and QMIX $\times$ the three workload families $\times$ three difficulty levels $\times$ ten independent algorithm seeds, each trained from scratch for 1,536,000 environment steps. The archived campaign identifier is `aag046-official-suite-canonical-v1`. Each valid run contributes one final checkpoint, its optimization history, and its evaluations on the validation and final-test splits. Algorithmic zero-production outcomes remain valid replicates and are retained in aggregate summaries. Pilot and tuning runs, deterministic mechanism checks, non-learning evaluations, geometry-control evaluations, infrastructure failures, incomplete runs, and reruns are not counted as additional MARL replicates and are tracked separately in the archive. Non-learning reference behavior is taken from the frozen non-learning reference evaluation record of the four reference controllers evaluated on the same instances. Earlier exploratory campaigns, protocol-development runs, and the pre-AssemblyGrid v1 geometry-family records are retained in the archive for traceability only and are excluded from every reported aggregate.

Each released instance records its complete runtime configuration and content identity. The archived evaluation package is accompanied by a SHA-256 checksum manifest that binds the released records to their archived counterparts. The archive retains the complete training histories of all 270 runs.

A.4 Reference implementation interface

The reference implementation exposes the decentralized AssemblyGrid v1 observation and candidate-specific action space, deterministic reset and step semantics, action masks, separate generation and execution seeds, benchmark KPI logging, and a distinct privileged global-state interface for centralized references and centralized-training methods. Candidate identity and realization follow the canonical key and realization rule of Section 2, and candidates are generated at most once per key. A candidate identifier is the software reference to that identity, while a candidate encoding is the controller-side representation used by a particular implementation.

For candidate construction, the engine first identifies logically ready operation occurrences and their required material holders, enumerates manipulation-local robot subsets that satisfy declared arity and role capability, constructs the corresponding task-level target assignment, and rejects structurally or task-level-geometrically inadmissible realizations. For decentralized exposure, robot u receives only candidates that contain u . The candidate list is deliberately not filtered by hidden remote active-trajectory conflicts or globally contended resources, as such filtering would leak joint feasibility through action availability. Those conflicts are resolved during capacity diagnosis or joint admission. Candidate ordering uses aggregate support and stable canonical identity. The reference implementation provides 32 pick-option slots and 64 operation-option slots per agent and raises an error on overflow. No candidate-addressability errors occurred in the official scenarios or reported experiments; the release addressability audit, which runs Parallel-aware for 180 ticks on each of the 45 official realizations, observed at most ten visible pick options per agent (limit 32), in CONCURRENCY-hard, and 24 operation candidates (limit 64), in COALITION-medium.

Optional Gymnasium and PettingZoo adapters and algorithm-layer reward helpers may be supplied for convenience; they do not alter the underlying AssemblyGrid v1 semantics. Instance

records retain the active geometry profile or declared variant and all problem-defining parameters, while AssemblyGrid v1 decentralized execution retains aggregate local candidate-support fraction without adding a free-form or addressable communication channel.

A.5 Reference MARL configurations

Table 17 lists the MARL-controller configurations used for the reported full-suite campaign and records the method-level settings needed to reproduce the 270 reference training runs. These settings define the reference implementations used in the reported MARL characterization.

Table 17: MARL methods’ configurations.

Field	IPPO	MAPPO	QMIX
Execution network	shared actor: LayerNorm, linear encoder to width 256, two tanh residual blocks, and separate action-family and action-slot heads whose logits are summed; orthogonal initialization	same shared actor as IPPO	shared Q network: three-layer tanh multilayer perceptron (MLP) with two hidden layers of width 256
Training value model	local critic on the agent’s own observation; same residual trunk at width 256	privileged agent-conditioned global critic on the concatenated global state and agent observation; same trunk	monotonic QMIX mixer over the privileged global state, mixing width 64, hypernetwork hidden width 128, non-negative first-layer weights
Optimizer	AdamW ($\epsilon = 10^{-5}$)	AdamW ($\epsilon = 10^{-5}$)	AdamW ($\epsilon = 10^{-5}$)
Learning rate	actor 10^{-4} , critic 3×10^{-4}	actor 10^{-4} , critic 3×10^{-4}	5×10^{-4}
Discount	$\gamma = 0.99$	$\gamma = 0.99$	$\gamma = 0.99$
On-policy update	generalized advantage estimation (GAE) $\lambda = 0.95$; policy clip 0.10; value clip 0.20; 1,600 environment steps per update; four epochs; minibatch 4,096; gradient-norm clip 0.5; target Kullback–Leibler (KL) 0.02	same as IPPO	not applicable
Exploration / regularization	entropy coefficient annealed linearly $0.015 \rightarrow 0.001$ over the training budget	same as IPPO	ϵ -greedy, $1.0 \rightarrow 0.05$ linearly over the first 75% of the budget, then held at 0.05
Replay / target	not applicable	not applicable	uniform replay buffer 50,000; batch 64; one update every 4 environment steps; hard target sync every 1,600 steps; single-network temporal-difference (TD) target with mean-squared-error loss (neither Double Q nor Huber loss was enabled)
Warm-up	3,200-step critic warm-up (actor frozen)	3,200-step critic warm-up (actor frozen)	3,200-transition replay warm-up before the first update
Training curriculum	one declared difficulty per run; the budget is split evenly over five realizations regenerated from the scenario parameters of suite indices 0–4 under run-specific seeds		
Initialization	random weights; no demonstrations, behavior cloning, or pretraining		
Training budget	1,536,000 environment steps		

A.6 Executable conformance mapping

The benchmark invariants in Appendix B.12 are linked to executable checks in the released reference implementation. The mapping makes the correspondence between the formal benchmark definition and the simulator explicitly traceable without making Python function names part of the benchmark semantics. Table 18 gives the principal mapping used by the conformance suite reported in Section 4.

Table 18: Invariant-to-check mapping.

Invariant	Benchmark property	Principal executable check
Conservation	Material conservation and unique ownership	<code>check_holding_uniqueness</code> ; duplication tests in <code>check_support_vs_ownership</code>
Support	Support does not create a second material owner	<code>check_support_vs_ownership</code>
Progress	Product progress is monotone across state transitions	<code>test_progress_is_monotone</code> , together with the construction that completed operations are only added
Recipe	Recipe-state consistency	<code>check_recipe_state</code>
Locality	Coalition locality	<code>check_coalitions</code>
Lifetime	Coalition membership and roles remain fixed while committed	<code>check_coalition_lifetime</code>
Arity	Coalition arity and the canonical role assignment are admissible	<code>check_coalitions</code>
Exclusivity	Robot, reservation, and finite-resource exclusivity	<code>check_parallelism</code> , <code>check_reservations</code> , <code>check_shelves</code>
Observation	Decentralized information locality	observation-constructor conformance tests
Masking	Action-mask admissibility does not reveal global joint feasibility	<code>check_action_mask_respected</code> and encoding tests
Accounting	Released and delivered accounting is monotone	<code>check_monotonic_counts</code>

Additional implementation-consistency checks verify WIP accounting, candidate-support table consistency, and agreement between realized concurrency and the declared diagnostic capacity. The exact capacity solver is checked against hand-derived optima on small constructed cases, including a case in which greedy selection is suboptimal. For auditability, the public release record binds the reported 203-check suite to the benchmark version, an immutable source or archive identifier, the exact test command, the Python and dependency environment, and the invariant-to-test mapping in Table 18. These checks support reproducibility but do not define additional benchmark objectives.

B Formal AssemblyGrid v1 definition

Appendix B collects the formal elements that define an AssemblyGrid v1 problem independently of any particular controller. The reference implementation distinguishes state, instance parameters, transition rules, hard constraints, structural relations, feasibility conditions, metrics, and invariants. Implementation identifiers and release information are kept in Appendix A, whereas experimental evidence remains in Section 4.

B.1 Core notation

Symbol	Meaning
$u_{i,j}$	Robot at fixed grid position (i, j)
$\mathcal{U}$	Complete robot population
$\mathcal{N}(u)$	AssemblyGrid v1 local neighborhood of robot u
$\mathcal{W}(u)$	Abstract reachable workspace of robot u
$\mathcal{H}(u, v)$	Shared handoff region of neighboring robots u and v
$\mathcal{G}_{\text{obs}}, \mathcal{G}_{\text{handoff}}, \mathcal{G}_{\text{manip}}$	Local observation, direct handoff, and direct manipulation relations
$\Gamma_r = (\mathcal{T}_r, \mathcal{E}_r, \mathcal{X}_r)$	Recipe structure of product type r ; $(\mathcal{T}_r, \mathcal{E}_r)$ is its precedence DAG
$\mathcal{X}_r$	Mutually exclusive operation alternatives for XOR recipe structure
z	Product unit; $r(z)$ denotes its product type

q	Material item, representing a part or subassembly
τ	Recipe operation
Δ_τ	Atomic material-state transition applied when operation τ completes successfully
(z, τ)	Operation occurrence for product unit z
c	Execution candidate for a ready operation occurrence
e	Activity created after an execution candidate is admitted
$C(e)$	Temporary coalition committed to activity e
$k_\tau^{\min}, k_\tau^{\max}$	Minimum and maximum coalition sizes allowed by operation τ
$k_{\text{topo}}^{\max}$	Largest coalition supported by the declared manipulation topology
s_t	Global benchmark state at decision epoch t
$p_z(t)$	Monotone product-progress state of product z
G_t^{op}	Conflict graph over ongoing and newly admissible concrete activities at decision epoch t
$\mathcal{E}_t^{\text{run}}$	Activities already committed and still executing at the start of decision epoch t
$\mathcal{J}_t$	Conflict-free activity sets that preserve all activities in $\mathcal{E}_t^{\text{run}}$
$K_{\text{prod}}^*(s_t)$	Canonical maximum number of mutually compatible recipe-operation activities that can productively occupy that interval while preserving existing commitments

B.2 Entities and state variables

The minimum state-bearing entities are robots, product units, material items, recipe operations, execution candidates, admitted activities, temporary coalitions, and physical or resource reservations. The global state s_t records robot availability and held material; part and subassembly location and ownership; recipe readiness, execution, completion, and resolved XOR choices; execution candidates and pending support records, including the lifecycle timing needed for expiry and switching eligibility; roles, activity commitments, and ongoing activities; fixtures, buffers, abstract workspaces, and other capacity-limited resources; product release, WIP, completion, and output status; and benchmark time. No separate communication-state component is included in AssemblyGrid v1, consistent with the absence of any agent-authored or addressable communication channel; only the benchmark-defined aggregate candidate-support cue is exposed.

A product carries a recipe identity and an evolving product-progress state. A robot likewise has task-level status in addition to its fixed grid position. Richer privileged or extension state may contain joint configuration, detailed robot-model features, skills, due dates, or economic values, but implementation-level storage alone does not make those variables AssemblyGrid v1 observations or instance requirements. Table 20 lists the semantic elements held fixed across all AssemblyGrid v1 instances.

B.3 Configuration parameters and instance identity

AssemblyGrid v1 separates fixed semantic choices from scenario parameters: the fixed semantics define how observations, handoffs, coalitions, recipes, geometry, and success are interpreted, whereas the official scenarios select problem quantities such as grid size, recipe, WIP limit, and release interval. The nine scenarios in Table 5, together with the constants in Table 21 and the recipe parameters in Table 22, are the complete workload definitions; no separate generic grid, WIP, release-interval, or recipe default supersedes them.

Table 20: Fixed semantics of AssemblyGrid v1.

AssemblyGrid v1 element	Fixed interpretation
Local topology	Moore radius one for the AssemblyGrid v1 observation, handoff, and manipulation relations; the relations remain semantically distinct.
Coalition upper bound	$k_{\text{topo}}^{\max} = \omega(\mathcal{G}_{\text{manip}})$, which equals four for the official AssemblyGrid v1 grids under the Moore-radius-one manipulation topology.
Geometry profile	Versioned abstract-v1 task-level geometry. Reach, spacing, clearance, timing, and conflict constants are owned by the declared profile.
Official conflict setting	The nine official scenarios use normalized workspace-conflict radius $\rho_{\text{conf}} = 0.25$; matched geometry studies use explicitly named variants.
Handoff semantics	Bilateral sender and receiver commitment is required.
Coalition semantics	Required members independently select the same candidate-specific action; the environment does not assign teammates.
Support lifecycle	Each robot holds support for at most one execution candidate. Support created at t_0 is admission-eligible through $t_0 + 3$ and first expires at $t_0 + 4$; repeated support does not refresh the timeout or lock, and switching is permitted from $t_0 + 1$.
Information structure	Local task information plus the aggregate candidate-support fraction; supporter identities, global feasibility, and free-form or addressable messages are not exposed.
Objective definition	Task success and benchmark metrics are defined by AssemblyGrid v1; learning reward and discount are method-level quantities.

An instance identity records every problem-defining choice required to reproduce the dynamics: grid and topology; active geometry profile or named variant; recipe and product mix; material, buffer, fixture, and other resource settings; release process and WIP limit; operation and transfer durations; coalition and handoff rules; observation definition; initial condition; and generation seed. Evaluation mode and any finite-batch timeout belong to the evaluation-execution record. Algorithm-specific quantities such as optimizer, network architecture, training reward, learning rate, batch size, entropy coefficient, or exploration schedule are method configuration.

Table 21: Constants shared by the official configurations.

Group	Constant	Value
Geometry	Cell spacing	1.0
	Reference reach and robot reach radius	1.5
	Workspace-conflict radius ρ_{conf}	0.25
	Robot clearance	0.10
	Role-target offset	0.12
	Retreat fraction	0.25
Timing	Reference time; robot speed; joint speed	1.0; 1.0; 1.0
	Pick, delivery, and handoff durations	2, 2, and 1 ticks
Coordination	Support timeout	3 ticks (admission-eligible through $t_0 + 3$)
Supply and access	Candidate-switching lock	1 tick
	Shelf capacity per raw material type	6
	Shelf replenishment interval	5 ticks
	Extra shelf-access probability	0.30
Disturbances	Robot payload; product mass range	10.0; 0.5 to 2.0
	Robot, operation, and handoff failure; duration noise; stockout	0 (disabled)
Evaluation	Horizon T (evaluation protocol)	400 ticks

Table 21 lists the constants shared by all nine official configurations; distances are in units

of cell spacing and durations in ticks. The extra shelf-access probability determines, once per instance and from the generation seed, which additional input robots may pick from each shelf (Appendix B.7). Payload feasibility never binds in the official scenarios, since product mass lies between 0.5 and 2.0 while the payload is 10.0, and generated due dates enter only the urgency term of the optional weighted productive value, which is used by optional weighted diagnostics and by the weighted coalition selection of the privileged centralized reference; neither therefore affects AssemblyGrid v1 dynamics or benchmark measures.

Table 22: Official recipe parameters.

Recipe	Operation	Inputs $\rightarrow$ out-put	Predecessors	$[k_{\tau}^{\min}, k_{\tau}^{\max}]$	Roles	d_{τ}
solo_inspection	inspect	A $\rightarrow$ FINAL	none	[1, 1]	inspector	2
flow_serial_3	prepare	A $\rightarrow$ P1	none	[1, 1]	operator	2
	process	P1 $\rightarrow$ P2	prepare	[1, 1]	operator	3
	inspect	P2 $\rightarrow$ FINAL	process	[1, 1]	inspector	2
flow_serial_5	prepare	A $\rightarrow$ P1	none	[1, 1]	operator	2
	process_1	P1 $\rightarrow$ P2	prepare	[1, 1]	operator	3
	process_2	P2 $\rightarrow$ P3	process_1	[1, 1]	operator	3
	finish	P3 $\rightarrow$ P4	process_2	[1, 1]	operator	2
	inspect	P4 $\rightarrow$ FINAL	finish	[1, 1]	inspector	2
pair_assembly	pair_join	A, B $\rightarrow$ FI-NAL	none	[2, 2]	holder_A, holder_B	4
supported_insert	align_ab	A, B $\rightarrow$ AB	none	[2, 2]	holder_A, holder_B	3
	supported_insert	AB $\rightarrow$ FI-NAL	align_ab	[2, 4]	holder, inserter, stabilizer, inspector	4
standard_ab	align_ab	A, B $\rightarrow$ AB	none	[2, 2]	holder_A, holder_B	3
	stabilize_insert	AB, C $\rightarrow$ FI-NAL	align_ab	[4, 4]	holder_AB, holder_C, operator, stabilizer	5
parallel_branch_light	prepare_a	A $\rightarrow$ PA	none	[1, 1]	operator_a	3
	prepare_b	B $\rightarrow$ PB	none	[1, 1]	operator_b	3
	join_branches	PA, PB $\rightarrow$ FI-NAL	prepare_a, prepare_b	[2, 2]	holder_PA, holder_PB	4

Table 22 lists every operation of the seven official recipes, with the arity interval $[k_{\tau}^{\min}, k_{\tau}^{\max}]$ and the nominal productive duration d_{τ} in ticks; recipe identifiers map to scenarios as listed in Appendix A. In AssemblyGrid v1 an operation takes place at a common target point computed from the base positions of its input holders, and each member’s abstract target is offset from that point by the role-target offset. Role skills are also recorded in the recipe files but are inactive as skill semantics are disabled. Productive weights enter only the optional weighted productive value, which is used by optional weighted diagnostics and by the weighted coalition selection of the privileged centralized reference; decentralized agents never observe it, and it affects neither AssemblyGrid v1 dynamics nor benchmark measures.

B.4 Rules and transition logic

The following rules define AssemblyGrid v1.

1. **Release.** A product enters only when the declared release process and WIP or input-capacity constraints permit it; a scheduled release that is not admitted is lost and not queued.
2. **Recipe readiness.** An operation is logically ready only when it is neither completed nor already executing, all mandatory predecessor conditions hold, and it remains enabled under the product’s resolved XOR choices. Material availability is checked separately. Completing one XOR alternative closes its mutually exclusive siblings as encoded by $\mathcal{X}_{\tau}$ for that product.
3. **Local coalition structure.** Direct-manipulation coalition members are pairwise adjacent in $\mathcal{G}_{\text{manip}}$; handoff uses $\mathcal{G}_{\text{handoff}}$; observation uses $\mathcal{G}_{\text{obs}}$.

4. **Decentralized commitment.** A cooperative operation starts only after all required members independently support the same uniquely identified execution candidate and the joint admission check succeeds. The environment does not assign coalition partners. After admission, the participating robots become committed to the resulting activity.
5. **Bilateral handoff.** Transfer requires neighboring robots u and v to select matching sender and receiver actions, with $v \in \mathcal{N}(u) \setminus \{u\}$. Specifically,

$$a_t^u = \text{handoff}(v, q), \quad a_t^v = \text{receive}(u, q).$$

The transfer point $p \in \mathcal{H}(u, v)$ must satisfy the operation geometry, local transfer feasibility, and the absence of a conflicting committed resource.

6. **Execution.** A committed activity occupies its actual robots, material inputs, exclusive resources, and abstract workspace or motion reservation for its declared duration.
7. **Completion.** Successful recipe-operation completion atomically applies the declared transition Δ_τ and strictly extends the product-progress state. Rejected attempts do not fabricate progress.
8. **Delivery.** A product succeeds only when its terminal recipe output has been produced and successfully delivered through an admissible output interface.

B.5 Hard constraints

Hard constraints govern both local candidate construction and later joint admission; they are not reward penalties. For an execution candidate c associated with operation occurrence (z, τ) and proposed robot set U_c , the structural conditions are

$$k_\tau^{\min} \leq |U_c| \leq \min(k_\tau^{\max}, k_{\text{topo}}^{\max}), \quad (22)$$

$$\text{ready}_t(z, \tau) = 1, \quad (23)$$

$$\text{InputsPresent}(z, \tau, s_t) = 1, \quad (24)$$

$$x_u(c) \in \mathcal{W}(u) \quad \forall u \in U_c. \quad (25)$$

Here $k_\tau^{\min}$ and $k_\tau^{\max}$ are the operation's coalition-size bounds, $\text{ready}_t(z, \tau)$ is logical recipe readiness, $\text{InputsPresent}(z, \tau, s_t)$ requires the declared material inputs, and $x_u(c)$ is the role-specific abstract target for robot u . For variable-arity operations with explicit roles, the ordered AssemblyGrid v1 role declaration covers $k_\tau^{\max}$ members, while the canonical assignment rule instantiates exactly $|U_c|$ roles for the realized candidate. Together with the declared role and manipulation-locality requirements, these conditions determine whether a candidate can be exposed to an eligible robot when they are available from that robot's permitted local information.

A fully supported candidate is admitted only if the joint transition additionally satisfies

$$\text{ResourceOK}(c, s_t) = 1, \quad \text{AbstractMotionOK}(c, s_t) = 1, \quad (26)$$

where the first predicate resolves finite and exclusive auxiliary resources and the second resolves the active task-level geometry, including conflicts with already committed or simultaneously considered activities. A robot, material item, fixture, buffer slot, or other exclusive resource cannot be simultaneously committed to incompatible activities, and material conservation and declared capacity limits must hold at every transition. Thus hidden nonlocal resource or concurrency conflicts are checked at joint admission. If c is admitted, the resulting activity $e(c)$ inherits the product unit, recipe operation, robot set, roles, operation site, auxiliary resources, and geometric assignment specified by the candidate. Table 23 summarizes the principal bindings these constraints create.

Feasibility. A generated benchmark instance is feasible only if a policy-independent successful execution exists: raw material can reach every operation that consumes it, a feasible handoff chain connects required sources to an output interface, every required operation satisfies $k_\tau^{\min} \leq k_{\text{topo}}^{\max}$ and admits at least one admissible execution candidate under the active geometry profile, and declared resource capacities do not make a required operation permanently infeasible. An instance that violates these conditions is invalid.

B.6 Bindings and dependencies

Table 23: AssemblyGrid v1 bindings and dependencies.

Binding	Formal or operational form	Consequence
Product to recipe	$z \mapsto \Gamma_r(z)$	Product achievement and legal future operations are recipe-specific.
Operation to material	$\tau \leftrightarrow (\text{inputs}_\tau, \Delta_\tau)$	Logical readiness alone is insufficient unless the required material inputs are present; successful completion applies the operation’s declared atomic material transition.
Operation to coalition	$\tau \leftrightarrow [k_\tau^{\min}, k_\tau^{\max}]$	Required simultaneous participation is operation-specific and limited by the topology.
Coalition to role	candidate member-to-role assignment	A candidate is admissible only if its proposed members can fill the roles declared by the operation.
Robot to capability	$u \leftrightarrow \text{declared task-level capabilities}$	Interchangeability is conditional under the active geometry profile.
Robot to workspace	$u \leftrightarrow \mathcal{W}(u)$	Base location and geometry profile determine abstract reach and direct transfer or manipulation opportunities.
Operation to resource	$\tau \leftrightarrow \text{res}_\tau$	Fixtures, buffers, tools, or shared zones can couple otherwise independent products.
Coalition to time	$C \leftrightarrow [t, t + D)$	Committed robots are unavailable to incompatible concurrent work.

B.7 Structural relations and interaction graphs

Several graphs coexist and are not interchangeable:

- $\mathcal{G}_{\text{obs}}$: which neighbors’ task-level dynamic state is locally observable;
- $\mathcal{G}_{\text{handoff}}$: which pairs may perform direct bilateral material transfer;
- $\mathcal{G}_{\text{manip}}$: which pairs may jointly participate in one direct-manipulation coalition;
- $\Gamma_r = (\mathcal{T}_r, \mathcal{E}_r, \mathcal{X}_r)$: the recipe structure for product type r , with mandatory predecessor relations in $\mathcal{E}_r$ and explicit XOR alternatives in $\mathcal{X}_r$;
- G_t^{op} : the state-dependent conflict graph over concrete ongoing and prospective activities.

The three robot relations coincide geometrically in the default Moore-radius-one configuration but remain semantically distinct. The maximum coalition size is derived specifically from $\mathcal{G}_{\text{manip}}$ through $k_{\text{topo}}^{\max} = \omega(\mathcal{G}_{\text{manip}})$.

Grid and boundary sets. For an $M \times N$ grid, $\mathcal{U} = \{u_{i,j} : 1 \leq i \leq M, 1 \leq j \leq N\}$ with $|\mathcal{U}| = MN$. The closed Moore neighborhood is

$$\mathcal{N}(u_{i,j}) = \{u_{i',j'} \in \mathcal{U} : \max(|i' - i|, |j' - j|) \leq 1\}, \quad (27)$$

and it induces $\mathcal{G}_{\text{obs}}$, $\mathcal{G}_{\text{handoff}}$, and $\mathcal{G}_{\text{manip}}$ in the default configuration. Input shelves $\mathcal{S}_{\text{in}}$ with pick regions c_s feed the left column, and the boundary robot sets are

$$\begin{aligned} \mathcal{U}_{\text{in}} &= \{u_{i,1}\}_{i=1}^M, & \mathcal{U}_{\text{out}} &= \{u_{i,N}\}_{i=1}^M, \\ \mathcal{U}(s) &= \{u \in \mathcal{U}_{\text{in}} : c_s \cap \mathcal{W}(u) \neq \emptyset, \text{row}(u) = \text{row}(s) \vee \beta_{u,s} = 1\}, \end{aligned} \quad (28)$$

where $\mathcal{U}_{\text{in}}$ and $\mathcal{U}_{\text{out}}$ are the left-boundary input and right-boundary output robot sets, respectively, and $\mathcal{U}(s)$ is the subset of input robots with pick access to c_s . An input robot in the shelf's own row has access whenever its workspace reaches c_s ; any other input robot whose workspace reaches c_s has access when the instance-generation indicator $\beta_{u,s} \sim \text{Bernoulli}(0.30)$, drawn once from the generation seed, equals one. Internal routing need not be monotone. Each robot u has an abstract reachable workspace $\mathcal{W}(u) = \{x : \|x - b(u)\|_2 \leq R_u\}$ about its base $b(u)$ with declared reach radius R_u , and a bilateral handoff between neighbors u, v is possible only in the shared region $\mathcal{H}(u, v) = \mathcal{W}(u) \cap \mathcal{W}(v)$.

B.8 State-dependent feasibility

Let c denote an execution candidate and let s_t denote the state at decision epoch t . The candidate-level structural predicate is

$$\begin{aligned} \text{CandidateBase}(c, s_t) &= \text{Ready}(c, s_t) \wedge \text{Material}(c, s_t) \wedge \text{Arity}(c) \wedge \text{Local}(c) \\ &\quad \wedge \text{Capability}(c) \wedge \text{Reach}(c). \end{aligned} \quad (29)$$

The predicates require, respectively, a logically ready operation, present material inputs, a coalition size within recipe and topology bounds, a valid manipulation-local robot set, role-compatible robots, and role-specific targets within the corresponding abstract workspaces. Robot u can be exposed to candidate c only if $u \in U_c$ and these candidate-level conditions are satisfied using information permitted by the local interface.

For a fully supported candidate, joint admission additionally requires

$$\text{JointAdmissible}(c, s_t) = \text{CandidateBase}(c, s_t) \wedge \text{Resource}(c, s_t) \wedge \text{AbstractMotion}(c, s_t), \quad (30)$$

where $\text{Resource}(c, s_t)$ resolves finite and exclusive resources and $\text{AbstractMotion}(c, s_t)$ resolves the active approach, retreat, clearance, and concurrency-conflict conditions against activities already executing and other candidates considered for the same interval. These nonlocal conditions are evaluated by the joint transition and are not used to leak hidden global state through the decentralized candidate list or action mask.

Joint feasibility and concurrency capacity. At decision epoch t , let $\mathcal{E}_t^{\text{run}}$ be the committed recipe-operation activities already executing and let $\mathcal{C}_t^{\text{new}}$ be the individually admissible execution candidates available in the current state, independent of whether they are fully supported by the controller. Running nonproductive logistics reservations remain fixed feasibility constraints through the resource and workspace checks but are not counted as capacity activities. For capacity analysis, each $c \in \mathcal{C}_t^{\text{new}}$ is represented by the prospective activity it would create if admitted; let $\mathcal{E}_t$ contain those prospective activities together with $\mathcal{E}_t^{\text{run}}$. Each $e \in \mathcal{E}_t$ therefore fixes a product unit, recipe operation, coalition, roles, operation site, and abstract-motion assignment. Actual joint admission is evaluated separately and is restricted to the fully supported subset $\mathcal{C}_t^{\text{sup}} \subseteq \mathcal{C}_t^{\text{new}}$. The conflict graph used for capacity analysis is

$$G_t^{\text{op}} = (\mathcal{E}_t, E_t^{\text{conf}}), \quad (e, e') \in E_t^{\text{conf}} \Leftrightarrow e, e' \text{ cannot share the execution interval}, \quad (31)$$

where E_t^{conf} is the set of pairwise execution-conflict edges. The feasible concurrent activity sets that preserve existing commitments are

$$\mathfrak{I}_t = \left\{ I \subseteq \mathcal{E}_t : I \text{ independent in } G_t^{\text{op}}, \mathcal{E}_t^{\text{run}} \subseteq I, |\{e \in I : \rho \in \text{res}(e)\}| + n_\rho^{\text{ext}}(t) \leq \text{cap}(\rho) \quad \forall \rho \right\}. \quad (32)$$

In canonical AssemblyGrid v1, every activity in $\mathcal{E}_t$ is a recipe-operation activity whose successful completion advances product progress, thus $\mathcal{E}_t^{\text{prod}} = \mathcal{E}_t$. The feasibility calculation preserves all committed operation teams and enforces conflicts from nonproductive logistics reservations through the resource and workspace checks used to construct admissible prospective activities; those logistics reservations constrain capacity but are not counted as capacity activities. The resource condition is stated separately from the conflict graph because a pairwise relation cannot express it: three activities sharing a resource of capacity two are pairwise compatible yet cannot all execute. Here $\text{res}(e)$ are the exclusive resources of activity e , $\text{cap}(\rho)$ is the declared capacity of resource ρ , and $n_\rho^{\text{ext}}(t)$ counts only committed users of ρ outside I , such as ongoing logistics reservations, thus committed productive activities are counted once through their membership in I . The single canonical count capacity is therefore $K_{\text{prod}}^*(s_t) = \max_{I \in \mathfrak{I}_t} |I|$. Alternative realizations of the same product-operation occurrence are mutually exclusive in the conflict/selection construction, and nonproductive logistics reservations constrain this maximum without contributing to it. In the reference implementation, the group-aware search is exact when the candidate set contains at most 64 candidates, at most 6 product-operation groups, and at most 100,000 grouped include-or-skip combinations. States outside these limits use deterministic greedy selection as an explicitly flagged lower bound; exactness is recorded for each sample. Canonical utilization averages are computed only on exact-capacity opportunity states, while approximate-state quantities are reported separately as bounds. For cooperative-only analysis the counted activities may be restricted to operations with $k_\tau^{\text{min}} \geq 2$. The product-progress state $p_z(t)$ is the triple of completed operations, resolved XOR choices, and delivery status; it is monotone in AssemblyGrid v1 (no rework or scrap), and an activity is productive iff its completion strictly extends $p_z(t)$ for some product.

B.9 Assumptions and abstractions

AssemblyGrid v1 uses synchronized discrete time; fixed robot bases; perfect local task-level sensing under $\mathcal{G}_{\text{obs}}$; aggregate local candidate-support fraction with no free-form or addressable messages; discrete material items with unique ownership; abstracted part pose, orientation, grasp, and contact; deterministic operation durations together with the default abstract geometric approach and retreat model; monotone product progress with no rework or scrap; a homogeneous robot population under the default **abstract-v1** profile; and task-level cooperation with no adversarial agent-specific objectives. The geometry model uses nominal task-level reach, clearance, timing, and abstract motion segments.

Kinematic or task-and-motion planning, physics and perception, disturbances, heterogeneous robot-capability studies, skills, due-date or value semantics, and standardized out-of-distribution generalization splits over recipes, layouts, geometry profiles, or scale are explicit extensions. If used, they must be named and reported as extensions rather than silently changing AssemblyGrid v1.

B.10 Structural instance descriptors and experimental factors

The following descriptors summarize structural properties of an instance or a matched intervention. They are not official workload-family labels and do not define a universal scalar difficulty:

$$D_{\text{size}} = (MN, \text{WIP}_{\text{max}}, |\mathcal{T}_r|, |\mathcal{R}|), \quad (33)$$

$$D_{\text{coalition}} = (f_{\text{coal}}, \mathcal{K}_r), \quad (34)$$

$$D_{\text{geometry}} = (r_{\text{space}}, \rho_{\text{conf}}, \nu_{\text{geom}}), \quad (35)$$

$$D_{\text{process}} = (L_r, w_r, n_{\text{join}}, n_{\text{xor}}), \quad (36)$$

$$D_{\text{topology}} = (\mathcal{G}_{\text{obs}}, \mathcal{G}_{\text{handoff}}, \mathcal{G}_{\text{manip}}), \quad (37)$$

where MN is the robot count, WIP_{max} is the configured WIP cap, $|\mathcal{T}_r|$ is the number of operations in recipe r , and $|\mathcal{R}|$ is the number of active product types. The coalition fraction is $f_{\text{coal}} = |\{\tau \in \mathcal{T}_r : k_{\tau}^{\min} > 1\}|/|\mathcal{T}_r|$, and $\mathcal{K}_r$ is the multiset of minimum coalition requirements $k_{\tau}^{\min}$ in recipe r . The spacing ratio is $r_{\text{space}} = d_{\text{base}}/R_{\text{ref}}$, where d_{base} is nominal center-to-center robot-base spacing and R_{ref} is reference reach. The quantity ρ_{conf} is the declared normalized workspace-conflict radius, and ν_{geom} is the active geometry-profile or variant identifier. Finally, L_r is recipe DAG depth, w_r is maximum antichain width, and n_{join} and n_{xor} count join and XOR structures respectively.

Topology and required coalition size are coupled, since every required operation must satisfy $k_{\tau}^{\min} \leq k_{\text{topo}}^{\max}$; consequently, scenario design cannot vary coalition demand independently of the active manipulation topology. Communication noise, disturbances, skill learning, and held-out transfer splits remain extension factors.

B.11 Performance and diagnostic metrics

Metrics do not alter environment dynamics and are independent of any training reward. Throughput, admitted-product completion, release acceptance, offered-demand fulfillment, lost releases, WIP, restricted mean flow time, admission and coalition success, occupancy, blocked-ready rates, blocked-activity events with their cause-specific subsets, and the operational no-progress-tick diagnostic are defined in Section 3.5; statewise productive-concurrency capacity, realized productive concurrency, and utilization are defined in Section 3.6. Execution-averaged capacity is a visited-state diagnostic and can differ across controllers because their trajectories visit different states.

For a completed finite batch $\mathcal{Z}$, the makespan is $C_{\text{max}} = \max_{z \in \mathcal{Z}} C_z$; if the declared finite-batch timeout T_B is reached first, the execution is reported as timed out and no C_{max} is assigned. Candidate switching counts robot switches from pending candidate support, and geometry reporting additionally uses normalized approach or retreat delay ticks. The legacy implementation key for the operational no-progress-tick diagnostic is `deadlock_ticks`. Further optional diagnostics include handoffs per product and role imbalance; abstract activity duration, path length, and reach or clearance margin are optional geometry diagnostics. Joint-space path length, inverse-kinematic failure, link collision, dynamics, contact, and grasp metrics require a declared higher-detail extension. Wall-clock speed, memory use, and diagnostic-solver cost are implementation instrumentation.

Reference bounds. Conservative algorithm-independent quantities give validation checks and small-instance references. If $n_k(t)$ operations requiring exactly k robots run concurrently, robot exclusivity gives $\sum_{k=1}^4 k n_k(t) \leq MN$, hence $n_k(t) \leq \lfloor MN/k \rfloor$ for a batch of operations with one coalition size before locality, reachability, precedence, and motion constraints. For product z released at a_z ,

$$C_z - a_z \geq \max\{L_{\text{crit}}(z), L_{\text{transport}}(z), L_{\text{motion}}(z), L_{\text{resource}}(z)\}, \quad (38)$$

where L_{crit} is a critical-path bound, $L_{\text{transport}}$ is a required-transfer bound, L_{motion} a necessary-abstract-motion bound, and L_{resource} an exclusive-resource bound; the terms need not be independent. The robot-time workload below sums the declared operations of a recipe and is therefore a valid lower bound for recipes whose operations all execute, which covers every official recipe; for a recipe with exclusive alternative routes the sum must be restricted to the operations of the realized route. For a finite batch $\mathcal{Z}$, let $D_{\min}(z, \tau)$ denote the minimum admissible realized duration of operation τ for product z under the instance. The lower-bound robot-time workload is $W_R^{\text{LB}} = \sum_{z \in \mathcal{Z}} \sum_{\tau \in \mathcal{T}_{\tau(z)}} k_{\tau}^{\min} D_{\min}(z, \tau)$,

$$C_{\max} \geq \max\{\max_z L_{\text{crit}}(z), W_R^{\text{LB}}/MN, L_{\text{resource, batch}}\}. \quad (39)$$

where $L_{\text{resource, batch}}$ is the batch-level lower bound induced by exclusive-resource workloads. These bounds hold across changing coalition sizes and do not assume independent 2×2 machines.

B.12 Formal invariants

A conforming simulator must preserve the following properties. These are semantic consistency conditions not performance objectives, and each is referred to by its name in the executable mapping of Appendix A.

Conservation. Every material item has a single consistent location and ownership state; no transition creates a duplicate material item except through an explicitly declared recipe output transition.

Support. Cooperative support for a held item does not create additional material ownership or duplicate the material item.

Progress. AssemblyGrid v1 completion never removes an already completed recipe operation from a product’s progress state.

Recipe. Completed and executing operation sets are disjoint, predecessor and XOR constraints are respected, and delivery occurs only after terminal recipe completion.

Locality. Every pair of members of a committed direct-manipulation coalition is connected in $\mathcal{G}_{\text{manip}}$.

Lifetime. Coalition membership, roles, and operation identity remain fixed while the activity is committed or executing and change only at a terminal event.

Arity. Realized coalition size lies within the operation’s declared interval and does not exceed $k_{\text{topo}}^{\max}$; assigned roles cover exactly the participating members.

Exclusivity. A robot or other exclusive resource cannot be committed to incompatible overlapping activities, and finite-capacity resources cannot exceed their declared capacity.

Observation. A decentralized observation is a function only of the robot’s own state, declared local dynamic state, and explicitly public benchmark information.

Masking. Local action masking removes declared local impossibilities but does not expose a global joint-feasibility oracle; at least one legal action remains available under every reachable decision state.

Accounting. Cumulative released and delivered counts never decrease, and reported WIP remains consistent with released but undelivered products.

Appendix A provides the executable mapping for these invariants and links each declared consistency condition to the corresponding implementation-level check. Exact diagnostic solvers must additionally agree with independently derived reference optima, obtained by hand or by exhaustive search, on the small validation instances for which exactness is claimed, providing a direct consistency check between the optimized diagnostic procedures and their reference calculations.

C Benchmark scope and extensions

Table 24 consolidates the scope of AssemblyGrid v1 by distinguishing mechanisms represented by the benchmark from properties introduced only through declared extensions.

C.1 Reading the scope summary

Each row names one mechanism, interface, or evaluation control and answers three independent questions about the named property: *Modeled* asks whether AssemblyGrid v1 represents the property at all, that is, whether it appears in the state, the transition model, the information structure, or the evaluation protocol. *Varied* asks whether the official suite or a declared matched control changes the property across evaluated instances, which is what makes it an experimental variable. *Measured* asks whether the benchmark reports a metric for the property, thereby making its effect visible in results.

The three questions are kept separate, since they represent distinct properties of the benchmark. Action masking is modeled and constrains every step, yet it is held fixed across the official suite and is not itself an outcome measure. Temporary coalition formation is modeled, varied across the three COALITION scenarios, and reported through formation success and failure counts. The privileged global state is modeled and used by centralized-training methods, but it is never a reported outcome, since the global state serves as a training-time interface not a production result. Reading the three columns together therefore shows not only what the benchmark contains, but which parts of it are currently exercised as experimental variables and which are observable in the reported measures.

A filled mark records that the property holds directly, and an open mark records that it holds partially, at a coarser abstraction, or only inside a matched control. A dash records that the property is part of AssemblyGrid v1 but is not varied or measured in that column; “n/a” is reserved for properties outside AssemblyGrid v1.

C.2 Scope entries and extensions

A row carrying “n/a” in all three columns identifies a property requiring a named extension with its own specification and evaluation protocol. Incorporating such a property into the canonical task definition constitutes a benchmark-version change, whereas varying an already modeled property extends the experimental design without changing the underlying task.

Two limits of this summary should be kept in mind. The summary records the state of the released benchmark; consequently, a controller that ignores a modeled mechanism is still evaluated against that mechanism. The summary also describes AssemblyGrid v1 only and does not compare it with other benchmarks; Table 1 provides that comparison.

Table 24: AssemblyGrid v1 scope and extensions.

Aspect	Modeled	Varied	Measured	Specified in
<i>Product and material progression</i>				
Recipe graph with multi-stage transformation	✓	✓	✓	Sec. 2.2, App. B.4
Sequence, AND, and XOR precedence	✓	○	○	Sec. 2.2
Material conservation and unique ownership	✓	–	✓	App. B.5, App. B.12
Bilateral synchronized handoff	✓	✓	✓	Sec. 2.3, App. B.4
Terminal delivery and task success	✓	✓	✓	Sec. 3.4
Rework, rollback, or scrap semantics	n/a	n/a	n/a	App. B.9
<i>Temporary cooperation</i>				
Single-robot operations or temporary coalitions of two to four robots	✓	✓	✓	Sec. 2.3, Sec. 3.3
Declared manipulation roles per member	✓	○	○	Sec. 2.2
Membership fixed while an operation is committed	✓	–	✓	Sec. 2.3, App. B.12
Coalition formation success, failure, and latency	✓	✓	✓	Sec. 4.2
Aggregate support-fraction coordination cue	✓	○	○	Sec. 2.4
Persistent teams or standing alliances	n/a	n/a	n/a	Sec. 2.3
Free-form or addressable communication	n/a	n/a	n/a	Sec. 5.3
<i>Decentralized information</i>				
Bounded local observation on the Moore neighborhood	✓	–	○	Sec. 2.4
Action masking of hard impossibilities detectable from permitted local information	✓	–	○	Sec. 2.4
Privileged global-state access (training for MAPPO and QMIX; evaluation-time action selection for Centralized)	✓	○	–	Sec. 2.4; Sec. 4
Privileged global joint-feasibility oracle exposed to the controller	n/a	n/a	n/a	Sec. 2.4
Sensing noise or perception layer	n/a	n/a	n/a	App. B.9
<i>Geometry and motion</i>				
Abstract reach and workspace overlap	✓	✓	✓	Sec. 2.1, App. B.7
Clearance and workspace-conflict predicates	✓	○	✓	Sec. 2.1, Sec. 4.2
Approach and retreat motion delay	✓	○	✓	Sec. 2.3
Kinematics, link collision, contact, and grasp dynamics	n/a	n/a	n/a	Sec. 5.3
<i>Concurrency and resources</i>				
Productive concurrency distinguished from occupancy	✓	✓	✓	Sec. 3.6
Productive-concurrency capacity	✓	○	✓	Sec. 3.6, App. B.8
Exclusive resources, fixtures, and buffers	✓	✓	○	App. B.5, App. B.6
WIP cap and product release process	✓	✓	○	Sec. 3.3
Offered, admitted, and lost releases / demand fulfillment (derivable, not exported)	✓	✓	○	Sec. 3.5
<i>Robots and topology</i>				
Fixed-base homogeneous manipulator lattice	✓	○	–	Sec. 2
Grid size	✓	✓	–	Sec. 3.3
Moore-radius-one interaction topology	✓	–	–	Sec. 2, App. B.7
Heterogeneous robot capability	n/a	n/a	n/a	App. B.9
Mobile robots or conveyor transport	n/a	n/a	n/a	Sec. 5.3
Robot downtime and failure recovery	n/a	n/a	n/a	App. B.9
<i>Evaluation protocol</i>				
Task success scored independently of learning reward	✓	–	✓	Sec. 2.4, Sec. 3.4
Instance identity, seed separation, and versioning	✓	✓	✓	Sec. 3.1, App. A
Fixed validation and final-test instance splits	✓	✓	✓	Sec. 4
Held-out generalization split over recipes or scale	n/a	n/a	n/a	App. B.9
Standardized robustness or disturbance protocol	n/a	n/a	n/a	Sec. 5.3

✓ property holds directly; ○ holds partially, at a coarser abstraction, or only in a matched control; – means not varied or measured in that column; n/a means outside AssemblyGrid v1 and reserved for a named extension.