MOSAIC: The Right Modules for Each Task in Embodied Agents

Kevin Wang [*1] Dweep Trivedi[*1] Vincent Ha[*1] Albert Jiang[1] Christian Ellis [1]
Ufuk Topcu[1] Swarat Chaudhuri[1] Zhangyang Wang[1]
[1]The University of Texas at Austin [*]Equal Contribution
{kevinwang.1839, utopcu, atlaswang}@utexas.edu
{dweept, swarat}@cs.utexas.edu

Abstract

Embodied agents demand a heterogeneous mix of capabilities: spatial precision for manipulation, persistent mapping for long-horizon navigation, and multi-step planning under natural-language instruction. The dominant remedy folds every such capability into one foundation model through spatial pre-training and geometry-aware fine-tuning, shipping a single fixed pipeline for every task in the benchmark. We contend that the question is not how to grow capability into the model, but how to compose, per task, the context that lets the model's existing capability bear on the task at hand. We formulate embodied agent design as per-task module-graph synthesis*: a joint search over which modules to include, which concrete tool implements each, and how each tool's parameters bind to environment observations and upstream outputs. Solving this search by full self-evolution is infeasible in embodied control, where validating each candidate requires a multi-step rollout and credit arrives only at episode end. We therefore introduce MOSAIC, a memory-conditioned agent that retains the load-bearing slice of self-evolution and drops the rest: the typed module catalog, the meta-agent, and the model weights stay fixed, while a memory of past pipeline designs and their outcomes evolves with deployment. A frontier meta-agent runs once per episode to compose a pipeline conditioned on retrievals from this memory, and a small executor runs every step.On EB-Nav, MOSAIC reaches 50.0% average success across five capability subsets, an 18.4 pp gain over the default EmbodiedBench agent at the same Qwen3-VL-8B backbone and +5.3 pp over a baseline that activates every module in the same catalog without per-task selection.*

1. Introduction

Embodied agents demand a heterogeneous mix of capabilities: spatial precision for manipulation, persistent mapping for long-horizon navigation, multi-step planning under natural-language instruction, and recovery from execution failure [14, 16, 39]. Each of these demands places a different burden on the agent's underlying foundation model. A pick-and-place task asks for geometric precision but rarely for long-horizon memory; a household navigation task asks for the opposite mix; an instruction-following task adds a third axis of demand. No single configuration of capabilities is uniformly right for every embodied task.

The dominant remedy is to fold every such capability into a single foundation model. Spatial pre-training [3, 15], geometry-aware fine-tuning [2], and multi-modal continued pre-training enlarge the model's internal world representation, hoping to produce a sufficiently capable agent by enlarging the agent's brain. However, even after substantial training cost, the resulting models still trail specialized perception modules in precision. A 335M-parameter depth specialist reaches under 5% relative error out of the box [38], while training a frontier vision-language model to a comparable level of geometric fidelity remains expensive and partial [2, 19].

Why grow capability into the model when specialized capability already exists outside it? An embodied agent has the reasoning to plan and act, but it can only reason over the context it is given. We argue the question is not how to enlarge the agent's internal world model, but how to compose, per task, the context that lets the agent's existing capability bear on the task at hand.

In adjacent settings, per-task selection has been shown to beat all-in-one alternatives. FAMA selects helper agents per customer-service conversation rather than fixing the ensemble for the whole benchmark [25]. Multimodal agents are also susceptible to irrelevant context, with even strong agents misled by environmental distractions [18]. The same asymmetry shows up sharply in embodied control: a SLAM module that amortizes its scene-graph cost across a 50-step navigation episode pays full token cost for negligible benefit on a 5-step pick-and-place, and an object detector that grounds manipulation floods the context of a long-horizon search. The right unit of pipeline design is the task, not the benchmark.

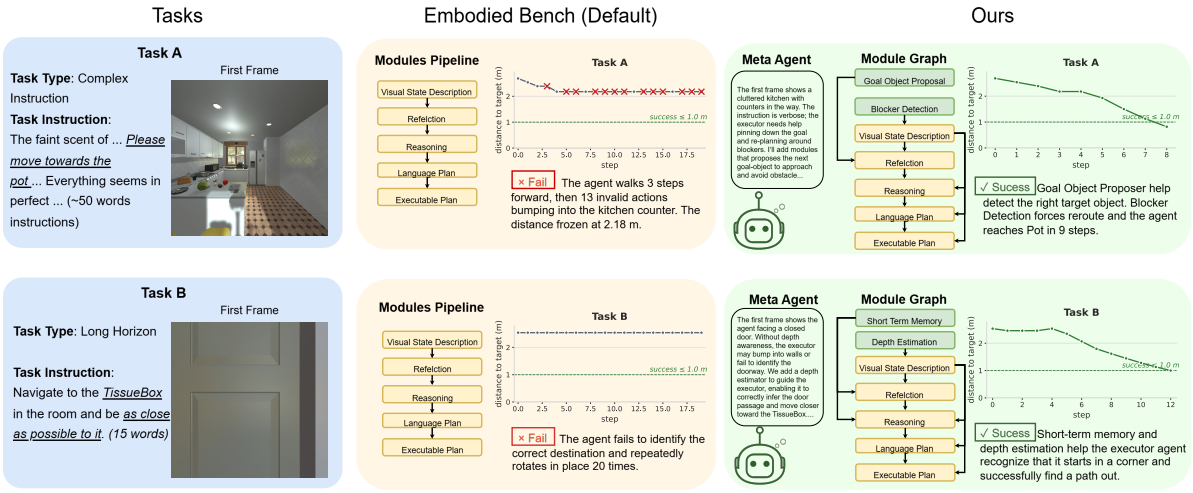


Figure 1. **MOSAIC overview.** For each task, MOSAIC constructs a task-specific module graph by selecting the necessary perception, memory, and planning modules, then routing input and output information between them at the task level. This enables the same executor to adapt its pipeline to different embodied task requirements without retraining.

A natural question follows: how should an agent actually pick a pipeline for each new task? One aspirational answer is a fully self-evolving agent [35, 44, 45]: as new tasks arrive, it updates its module library and its parameter-binding policy so that each subsequent design improves on the last. However, this ideal is out of reach in embodied control for three coupled reasons: (i) every candidate update has to be validated through a multi-step rollout that can stretch to 20 steps per episode; (ii) the search jointly couples module selection, tool selection, and parameter binding; and (iii) the credit signal arrives only at episode end, leaving each candidate with a single noisy scalar. Unlike static prompt or program optimization, where an inner-loop search over candidates is cheap, in our setting such a search is prohibitively expensive. We contend that the right move is to retain self-evolution's most load-bearing component and drop the rest. Concretely, we study *memory evolution alone*, a narrow but tractable slice in which the typed module library, the meta-agent, and the model weights stay fixed, and only a memory of past pipeline designs and their outcomes grows with experience. This narrowing defines the central object of the paper: *per-task module-graph synthesis*, in which a per-task pipeline is composed by a meta-agent conditioned on retrievals from this evolving memory.

In this work, we introduce MOSAIC, a memory-conditioned agent that performs *per-task module-graph synthesis* over a typed module library (Figure 1). Unlike scale-and-train recipes that ship one fixed pipeline across every task in the benchmark [2, 3, 15], MOSAIC composes a different module graph for each task and reuses the same frozen executor throughout. A frontier meta-agent runs once per episode: conditioned on retrievals from a growing memory of past designs, it proposes N candidate pipelines that jointly resolve module inclusion, tool variants, and parameter bindings. Each candidate is raced through K replicated rollouts, and a structured per-design summary of mean reward, stability, lesson, and critic confidence is written back to memory for retrieval on later tasks. A small executor then runs every step within each rollout. This split is what makes per-task design affordable: episodes stretch up to 20 steps, so a once-per-task meta-call dilutes to almost nothing per step, and typing reduces pipeline synthesis to slot-filling that a frontier model prompted with the catalog and a few retrieved exemplars can drive directly [27].

Contributions. **(1)** We formulate embodied agent design as *per-task module-graph synthesis*, where the agent composes context per task instead of carrying a fixed pipeline across the benchmark.

(2) We introduce MOSAIC, a memory-conditioned agent that synthesizes a per-task pipeline over a typed module library, improving its selections through an accumulated memory of past designs.

(3) We show on EB-Nav that modularity and per-task selection both matter, with accumulated memory load-bearing specifically on long-horizon tasks (§5).

2. Related Work

Modular agents and pipeline search. Two long lines each collapse one of MOSAIC's three decision layers. Tool-calling agents fix the module structure and pick ⟨tool, arguments⟩ at inference time [23, 26, 40], collapsing module selection; neural module networks and declarative LM-program frameworks fix one tool per module and search only over parameter bindings [1, 13], collapsing tool se-

lection. Agentic architecture search lifts both restrictions and searches all three layers offline: AgentSquare [27] recombines Planning, Reasoning, Tool, and Memory slots; AFlow [43] runs MCTS over workflow DAGs; ADAS [11] grows an archive of meta-agent-invented designs; and a sibling line treats pipelines as differentiable graphs to be back-propagated through [5, 37, 42, 47]. All of them commit to a single fixed pipeline for the entire benchmark, the constant-output special case of our formulation. MOSAIC exploits within-benchmark heterogeneity: the architecture that wins on long-horizon search is Pareto-dominated on short approach tasks and conversely, so the right unit of design is the task, not the benchmark.

Memory-augmented agents. MOSAIC conditions design on retrieved cases from an evolving memory, a mechanism with deep precedent. Reflexion [29] conditions the next attempt on a buffer of verbal reflections; ExpeL [46] retrieves text lessons extracted from trajectories; Generative Agents [22] synthesize reflections from a chronological memory stream; workflow-level analogues cache reusable task recipes [31, 34]; and a parallel literature studies the memory architecture itself [7, 9, 10, 21]. In all of these, memory parameterizes the policy that emits the next *action*. MOSAIC instead treats memory as a conditioning signal for *pipeline design*: each entry is a structured per-design record (mean reward, stability, lesson, critic confidence) that retrieves exemplars to condition a fresh meta-agent call, not exemplars that condition the next step. A related family pursues fuller self-evolution by editing code or weights [8, 32, 33, 35, 36, 44, 45]; we hold the catalog, meta-agent, and weights fixed because in embodied control each candidate edit costs a multi-step rollout with a single noisy episode-end signal.

Per-query routing and inference-time orchestration. A separate family adapts at the granularity of an individual query. Cost-aware routers cascade between models or workflow depths per query [4, 6, 20, 24, 25, 30, 41], while inference-time orchestrators compose specialized tools per query for single-shot multimodal reasoning [12, 17, 19, 28, 48]. Both treat the query as the design unit, an episode of length one. In multi-step embodied control, per-step synthesis is unaffordable but a once-per-episode design call dilutes to almost nothing per step; MOSAIC is the per-task, multi-step counterpart to these single-shot orchestrators.

MLLM specialization for embodied perception. The dominant alternative line addresses the same problem at a different layer, folding spatial and geometric capability *into* the model itself via spatially-augmented training and geometry-aware fine-tuning [2, 3, 15], at substantial cost and still trailing specialized perception modules in geometric precision. We hold model weights fixed and ask instead which capability already exists outside the model, and how to invoke it per task; this line is therefore complementary to ours, not competitive. The benchmarks against which it is evaluated [14, 16, 39] almost universally assume a fixed monolithic agent across the entire benchmark, an assumption MOSAIC directly challenges.

3. Problem Setup

We formulate embodied-agent design as memory-conditioned per-task module-graph *synthesis*. The agent operates over a fixed typed library of modules and tools, and accumulates a memory bank of past task–pipeline–outcome records as it runs. Before each episode, a meta-agent synthesizes a feasible pipeline from this library, conditioned on both the current task and the current memory bank. Section 3.1 fixes the embodied task setting; Section 3.2 defines the typed design space, the memory bank, and states the central optimization problem.

3.1. Embodied Task Setup

We consider a distribution $\mathcal{D}$ over embodied tasks. Each task $g = (L, I_0, C)$ specifies a natural-language instruction L, an initial RGB observation I_0, and a capability category C (e.g., spatial reasoning, long-horizon planning) drawn from the benchmark's taxonomy. The agent interacts with the environment over a partially observed episode of horizon H [39].

At step t, the agent receives a visual observation I_t, conditions on the instruction L and the interaction history $h_t = (I_0, a_0, I_1, a_1, \ldots, a_{t-1}, I_t)$, and samples an action

$$a_t \sim \pi_{\text{act}}(\cdot \mid L, h_t). \tag{1}$$

The resulting trajectory $\zeta = (I_0, a_0, I_1, a_1, \ldots, a_{H-1}, I_H)$ is scored by a benchmark-provided task-success function $R(\zeta; g) \in [0, 1]$ (e.g., binary success or subgoal completion), which is the only feedback the environment provides. We measure agents by their expected score $\mathbb{E}_{g\sim\mathcal{D}}[R(\zeta; g)]$.

In what follows, we take π_{act} to be *modular*: its actions are produced by composing a small number of typed modules (e.g., perception, reasoning, planning) drawn from a fixed library, rather than by a single monolithic network. A separate *meta-agent* synthesizes this composition on each new task. Prior work fixes one π_{act} for the entire benchmark; we instead synthesize one per task, conditioned on a memory of past designs. Section 3.2 formalizes the module library, the meta-agent, and the synthesis problem.

3.2. Memory-Conditioned Module-Graph Synthesis

We formalize the constrained self-evolution setup motivated in Sec. 1: only the agent's memory bank evolves across

episodes, while the module library, tool implementations, meta-agent, and model weights remain fixed. Throughout, $\mathcal{M}$ denotes the universe of candidate modules, $\mathcal{T}$ the universe of candidate tools, Env the typed schema of environment channels (e.g., RGB, depth, gripper state), and Cfg_t the set of design-time configuration values (e.g., prompts, hyperparameters) attached to each tool $t \in \mathcal{T}$. Both $\mathcal{M}$ and $\mathcal{T}$ are given as part of the problem instance; we study the question of synthesizing pipelines from this library.

Definition 1 (Tool)**.** *A* tool t *has typed inputs* $\mathrm{In}(t)$*, output type* Y_t*, and an implementation kind (e.g., LLM, neural network, vision model, classical CV routine, code, shell command). For simplicity, the math in this section treats every input in* $\mathrm{In}(t)$ *as required; Sec. 4 describes the implementation-level handling of optional inputs.*

Definition 2 (Module)**.** *A* module m *has a natural-language role description* role_m*, an output type* Y_m *that downstream consumers bind against, and an allowed-consume set* $U_m \subseteq \mathcal{M} \setminus \{m\}$ *of modules whose outputs* m *is permitted to consume.*

Definition 3 (Module–Tool Compatibility)**.** *The tools that may instantiate module* m *are*

$$\mathrm{Tools}(m) = \Big\{\, t \in \mathcal{T} \;\Big|\; Y_t = Y_m \;\wedge\; \forall p \in \mathrm{In}(t) : \mathrm{type}(p) \in \mathrm{Env} \cup \{Y_{m'} : m' \in U_m\} \cup \mathrm{Cfg}_t \,\Big\}. \tag{2}$$

Definition 4 (Pipeline)**.** *A* pipeline *is a triple* $\pi = (M', \tau, \rho)$ *consisting of:*

- *a* module selection $M' \subseteq \mathcal{M}$*;*
- *a* tool assignment $\tau : M' \to \mathcal{T}$ *with* $\tau(m) \in \mathrm{Tools}(m)$*;*
- *an* input wiring ρ *that, for each* $m \in M'$*, binds every* $p \in \mathrm{In}(\tau(m))$ *to an environment channel, an upstream module output, or a config value.*

A pipeline is feasible *when every binding type-matches and the consume relation* $\{(m, m') : m, m' \in M',\ m' \in U_m\}$ *is acyclic. We write* Π *for the set of feasible pipelines.*

Memory bank. The agent maintains a memory bank $\mathcal{B}$ that stores records from past episodes. After episode e, the bank is updated as

$$\mathcal{B}_{e+1} = \mathcal{B}_e \cup \{(g_e, \pi_e, r_e, \ell_e)\}, \tag{3}$$

where $\pi_e \in \Pi$ is the deployed pipeline, $r_e = R(\zeta_e; g_e)$ the terminal score, and ℓ_e a post-episode lesson summarizing what worked or failed; the trajectory ζ_e exists transiently during scoring and is not retained. We write $\mathfrak{B}$ for the space of possible memory banks.

Problem 1. Memory-Conditioned Module-Graph Synthesis

Given. $\mathcal{M}$, $\mathcal{T}$, $\{\mathrm{Tools}(m)\}_{m\in\mathcal{M}}$, Env, $\{\mathrm{Cfg}_t\}_{t\in\mathcal{T}}$, memory bank $\mathcal{B}$, task distribution $\mathcal{D}$, and task-success function $R(\zeta; g)$.

Specify. A memory-conditioned synthesis policy $f \in \mathcal{F}$, where $\mathcal{F} = \{f : \mathcal{G} \times \mathfrak{B} \to \Pi\}$, that emits the modular action policy $\pi_{\mathrm{act}} = f(g, \mathcal{B})$ for each task — structured as a pipeline per Definition 4. We measure f by

$$J(f) = \mathbb{E}\big[R(\zeta; g)\big], \quad g \sim \mathcal{D},\ \zeta \sim \mathrm{rollout}(\pi_{\mathrm{act}}, g), \quad \text{s.t.} \quad \mathrm{Feasible}(\pi_{\mathrm{act}})\ \ \forall g.$$

We do not search over $\mathcal{F}$. In Sec. 4 we fix f as a frontier meta-agent; only the memory bank $\mathcal{B}$ evolves across episodes (Eq. 3), and improvement in $J(f)$ comes from retrieval over a growing $\mathcal{B}$, not from updates to f.

The feasible search space Π is exponential: there are $2^{|\mathcal{M}|}$ subsets M' alone, and within each subset every module can be filled by any of its compatible tools with any of its valid input wirings. Exhaustive search is therefore out of reach, motivating a learned synthesis policy.

We realize this policy as MOSAIC in Sec. 4, whose proposal–evaluation–memory loop we detail next.

4. Method

MOSAIC realizes the synthesis policy f from Problem 1 as a closed-loop system that proposes pipelines from the typed catalog, evaluates them in embodied trials, and writes structured per-design summaries into memory for future retrieval. Figure 2 shows one cycle. §4.1 describes how the meta-agent conditions on $(g_e, \mathcal{B}_e)$ to propose a set of N candidate pipelines. §4.2 describes how each candidate is evaluated through K replicated rollouts and consolidated into a per-design summary that updates $\mathcal{B}$. §4.3 grounds these abstractions in the concrete module catalog used in our experiments, and §4.4 illustrates how the framework extends to new tools and modules.

4.1. Memory-Conditioned Proposal

Given a new task $g_e = (L_e, I_{0,e}, C_e)$ at episode e, MOSAIC first retrieves a memory context

$$\mathcal{R}_e = \mathrm{Retrieve}(g_e, \mathcal{B}_e). \tag{4}$$

Retrieval is restricted to records whose capability category matches C_e and ranked by a multimodal similarity

$$s(g, r) = \alpha\, s_{\text{text}}(g, r) + (1-\alpha)\, s_{\text{image}}(g, r), \tag{5}$$

where s_{text} compares an embedding of the instruction L in g against the stored instruction embedding in r, s_{image} compares an embedding of the first frame I_0 in g against the

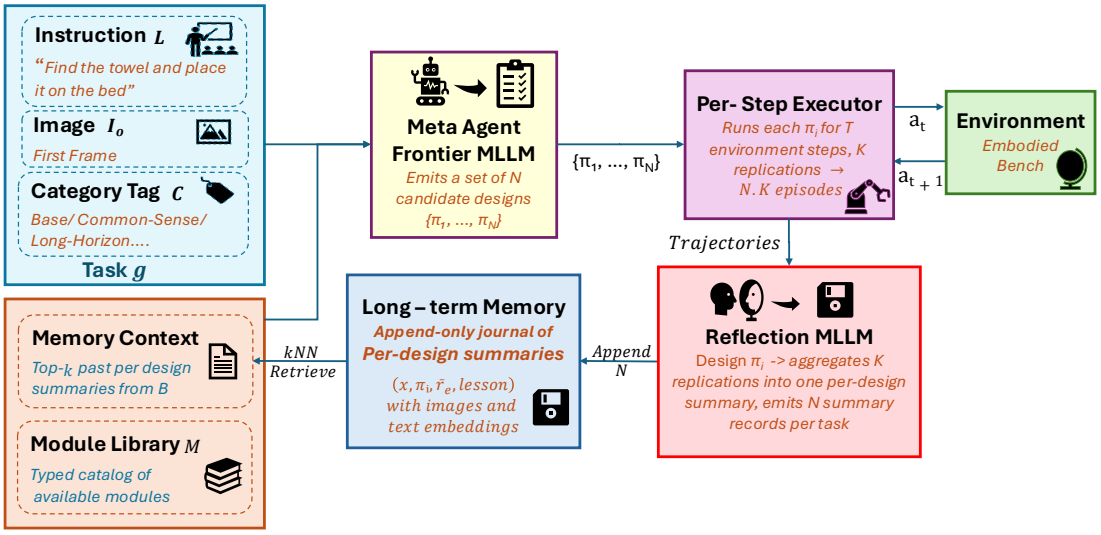


Figure 2. **MOSAIC Optimization pipeline** One task drives one cycle: the task input (light blue) and the module catalog with retrieved memory context (orange) are provided to the meta-agent (yellow), which proposes N candidate pipelines. The per-step executor (purple) evaluates each candidate through K replicated rollouts in the environment (green). The resulting trajectories are aggregated by a reflection critic (red) into N per-design summaries, which are appended to the long-term memory $\mathcal{B}$ (blue). The catalog $\mathcal{M}$, meta-agent, executor, and model weights stay fixed; only $\mathcal{B}$ grows.

stored first-frame embedding in r, and we take $\alpha = 0.7$. The top records are rendered into the meta-agent prompt as a compact summary of past designs, outcomes, and lessons for similar tasks.

The meta-agent, a frontier MLLM, receives $(g_e, \mathcal{M}, \mathcal{R}_e)$ and emits a set of N candidate pipelines

$$\{\pi_e^{(1)}, \ldots, \pi_e^{(N)}\}, \quad \pi_e^{(i)} = (M_e^{(i)}, \tau_e^{(i)}, \rho_e^{(i)}), \qquad (6)$$

each in the format of Definition 4. Pipelines are emitted as JSON objects specifying enabled modules, variant choices, and input bindings, constrained by a schema over admissible module and channel names. A validator canonicalizes the output, auto-fills required inputs, closes dependencies, and rejects malformed designs; if a proposal cannot be repaired, the meta-agent retries or falls back to a high-confidence design retrieved from $\mathcal{B}_e$.

4.2. Race-Based Evaluation and Memory Consolidation

Both LLM-driven design and embodied execution are stochastic, so a single rollout of a single design is too noisy to rank candidates reliably. MOSAIC therefore evaluates each of the N proposed designs through K replicated rollouts, yielding $N \cdot K$ episodes per task; replicates run in parallel because designs do not interact during execution.

We score each rollout with a shaped reward $q_e^{(i,k)}$ that combines binary task success with penalties for invalid actions, parse errors, fallback use, and step inefficiency. The shaped reward stays local to the race.

After all K replicates of a design have landed, a frontier MLLM critic aggregates them into a per-design summary $z_e^{(i)} = (\bar{q}_e^{(i)}, \sigma_e^{(i)}, \ell_e^{(i)}, c_e^{(i)})$, where $\bar{q}_e^{(i)} = \frac{1}{K}\sum_k q_e^{(i,k)}$ is the mean shaped reward, σ a stability indicator, ℓ a natural-language lesson describing what worked or failed across the replicates, and c the critic's confidence in that lesson. The race winner $i_e^\star = \arg\max_i \bar{q}_e^{(i)}$ is recorded as a flag on the corresponding summary.

The memory update appends N records, one per design,

$$\mathcal{B}_{e+1} = \mathcal{B}_e \cup \left\{ \left(g_e, \pi_e^{(i)}, z_e^{(i)}, [i = i_e^\star]\right) \right\}_{i=1}^{N}, \qquad (7)$$

each keyed by text and image embeddings of $(L_e, I_{0,e})$ to support the retrieval defined in Eq. 5. Storing both winners and losers, rather than only $i_e^\star$, lets the next task's retrieval surface contrastive evidence, for example that a particular variant choice consistently underperforms on long-horizon tasks. We additionally maintain cross-design comparison statistics (paired deltas between designs in the same race, factor effects across tasks, and per-design rollups) which the meta-agent can consume when proposing future designs; these are described in the supplementary.

Across episodes, the same fixed meta-agent emits different pipelines as $\mathcal{B}$ grows: this is how MOSAIC drives $J(f)$ upward under Problem 1 without modifying any module implementation, tool, or model weight. When $N{=}1$ and $K{=}1$, the race collapses and MOSAIC reduces to a one-shot proposer with deterministic memory write; we use $N{>}1, K{>}1$ in our training §5.

Configuration	Avg	Base	Common	Complex	Visual	Long
Original EmbodiedBench	31.56 ± 14.19	48.89 ± 5.09	33.33 ± 3.34	32.22 ± 3.85	35.56 ± 5.09	7.78 ± 1.92
EmbodiedBench (Decompose)	43.11 ± 7.18	52.22 ± 7.70	40 ± 5.77	41.11 ± 5.09	37.78 ± 1.92	44.45 ± 6.94
GPT-4o-mini (Decompose)	30 ± 15.94	37.78 ± 5.09	27.78 ± 5.09	34.44 ± 10.18	46.67 ± 0	3.33 ± 3.34
All-modules	44.67 ± 5.01	48.89 ± 1.92	44.45 ± 6.94	42.22 ± 5.09	41.11 ± 1.92	46.67 ± 5.77
MOSAIC (ours)	**50.00** ± 1.33	**60.00** ± 8.82	**46.67** ± 0.00	**45.56** ± 10.18	**51.11** ± 5.09	**46.67** ± 3.33

Table 1. **Task success rates on EB-Nav (test split).** Per-subset success rate (%) and the five-subset mean *Avg*. All rows use Qwen3-VL-8B as executor except GPT-4o-mini (Decompose); see §5.1 for baseline definitions.

4.3. Module Catalog

Our experiments instantiate $\mathcal{M}$ with a 12-module catalog covering short-term memory, blocker detection, geometric reasoning, perception, reflection, reasoning, and planning. Five modules are deterministic helpers (e.g., `object_detector`, `depth_estimator`); seven are LLM-based and execute in dependency order, terminating in `executable_plan` which emits the machine-readable action list executed by the environment. Several modules admit named variants (e.g., `stm`.full vs `stm`.lean) that the meta-agent may select per design, and a handful admit *optional inputs*, auxiliary signals such as prior episode context or coarse hints, that the meta-agent wires through ρ when a type-matching source exists and ignores otherwise. Each module's role, available variants, and input sources are reported in Table 4, with Figure 4 laying out the catalog by execution tier with its dependency DAG.

4.4. Extensibility

The typed protocol of Definitions 1–4 admits two orthogonal axes of extension without modifying the meta-agent interface or Problem 1; both are illustrated in Figure 5 in the supplementary. **(a) Growing** $\mathrm{Tools}(m)$**.** A new tool whose output type matches Y_m is drop-in. For example, the LLM-based tool currently used for `visual_state_description` can be replaced by a YOLO+template pipeline or a fine-tuned scene-graph VLM by changing only $\tau(m_{\mathrm{VSD}})$, while every downstream module that consumes `visual_state_description`'s output is unaffected. **(b) Growing** $\mathcal{M}$**.** A new module that publishes a novel output type extends the source space available to every downstream module. For example, adding a SLAM module m_{SLAM} that publishes a scene-graph tensor lets downstream planning modules bind a new `scene_graph` input via ρ without changing Π's definition; the depth-estimator module already in our catalog (Table 4) was added the same way. Neither change requires modifying the meta-agent prompt or the formal objective: only the catalog and the validator schema are extended.

5. Experiments

We evaluate MOSAIC along three questions that map directly to the design of §4. **Q1 (aggregate value).** Does per-task pipeline selection over a fixed typed catalog beat fixed-pipeline baselines at matched executor cost? **Q2 (per-subset structure).** Where across capability subsets does the gain actually land, and what does that tell us about which problems per-task selection solves? **Q3 (memory mechanism).** How does the long-term memory bank $\mathcal{B}$ contribute, at what layer of the design space does it work, and which retrieval signal carries it?

5.1. Experimental Setup

Environments and metrics. We run on the EB-Nav track of EmbodiedBench [39]. Each environment is partitioned into five capability subsets: Base, Common Sense, Complex Instruction, Visual Appearance, and Long Horizon. We use a 50/50 train/test split per environment, yielding 30 training tasks and 30 test tasks; MOSAIC makes a single pass over the training tasks (one episode per task — the meta-agent's race produces $N \cdot K$ rollouts and writes N structured records per task) so the case bank $\mathcal{B}$ accumulates one round of records before evaluation. Table 1 and Table 2 report on the test split; Table 3 reports on the train split. The headline metric is per-subset task success rate, and the aggregate metric *Avg* is the unweighted mean across the five subsets within each environment.

Models. MOSAIC uses two models per episode (cf. §4.1). The frontier *meta-agent* runs once per episode to propose the N candidate pipelines and to write the per-design summaries; we use Claude Sonnet 4.6 by default. The *executor* runs every step inside each rollout; unless otherwise noted the executor is Qwen3-VL-8B, with an additional experiment using GPT-4o-mini as the executor (Table 1).

MOSAIC configuration. Unless stated otherwise we use N=3 candidate designs per task, K=3 replicate rollouts per design, and α=0.7 for retrieval (Eq. 5). The catalog $\mathcal{M}$ is the 12-module catalog of §4.3. The case bank $\mathcal{B}$ is initialized empty and accumulated online. For testing we use N=1 and K=1, collapsing it into one design per episode.

Baselines. Four configurations bracket the baseline space (Table 1). **(i) Original EmbodiedBench.** The stock five-module agent shipped with EmbodiedBench [39], run on the same Qwen3-VL-8B executor as MOSAIC. **(ii) EmbodiedBench (Decompose).** The same EmbodiedBench planner with a decomposition prompt that splits the instruc-

Configuration	Avg	Base	Common	Complex	Visual	Long
MOSAIC (text-only retrieval)	49.34 ± 10.55	64.45 ± 3.85	45.55 ± 6.94	44.44 ± 12.62	48.92 ± 5.03	43.33 ± 8.82
MOSAIC (w/o long-term memory)	48.00 ± 2.67	**66.67** ± 3.33	**54.44** ± 1.92	**51.11** ± 5.09	**55.56** ± 6.94	12.22 ± 7.70
MOSAIC (ours)	**50.00** ± 1.33	60.00 ± 8.82	46.67 ± 0.00	45.56 ± 10.18	51.11 ± 5.09	**46.67** ± 3.33

Table 2. **Ablation of MOSAIC components on EB-Nav (test split).** *Text-only retrieval* sets α=1 in Eq. 5; *w/o long-term memory* disables retrieval over $\mathcal{B}$ while keeping the per-episode race. All rows use Qwen3-VL-8B.

Configuration	Avg	Base	Common	Complex	Visual	Long
Original EmbodiedBench	29.11 ± 13.42	41.11 ± 6.94	28.89 ± 10.18	35.56 ± 8.39	32.22 ± 6.94	7.78 ± 3.85
MOSAIC (no LT memory)	49.11 ± 2.78	**61.11** ± 7.70	**55.56** ± 6.94	**62.22** ± 8.39	**56.67** ± 3.33	10.00 ± 8.82
MOSAIC (ours)	**52.89** ± 5.55	56.67 ± 6.67	48.89 ± 3.85	58.89 ± 8.39	51.11 ± 5.09	**48.89** ± 11.71

Table 3. **Train-split success rates on EB-Nav.** Same LTM ablation as Table 2, on the 30 training tasks; improvement of MOSAIC over the no-memory row is attributable to memory accumulation alone.

tion into sequential subgoals, again on Qwen3-VL-8B. **(iii) GPT-4o-mini (Decompose).** The Decompose pipeline of (ii) with the executor swapped to GPT-4o-mini; the only change relative to (ii) is the executor model, isolating executor capacity. **(iv) All-modules.** Every module in our 12-module catalog activated simultaneously, with no per-task selection. This is the same catalog MOSAIC uses; it isolates the value of per-task selection against catalog richness alone.

5.2. Main Results

Table 1 reports per-subset success rates on EB-Nav. Holding the executor fixed at Qwen3-VL-8B, three baseline comparisons isolate what MOSAIC (50.00%) contributes. Against Original EmbodiedBench's stock five-module agent (31.56%), MOSAIC gains 18 pp — the combined value of decomposition, a richer typed catalog, and per-task selection, all of which Original EmbodiedBench lacks. Against EmbodiedBench (Decompose) (43.11%), which adds a decomposition prompt to the same five-module agent, MOSAIC gains 7 pp: decomposition alone closes most of the headline gap, leaving a residual that the richer catalog and per-task selection together account for. Against All-modules (44.67%), which activates the entire 12-module catalog MOSAIC also uses but disables per-task selection, MOSAIC gains 5.3 pp: same catalog, same executor, different rule for choosing what to run — isolating the value of *selection* itself.

A separate ablation isolates executor capacity. Holding the Decompose pipeline fixed and replacing Qwen3-VL-8B with GPT-4o-mini drops aggregate SR by 13 pp (43.11% → 30.00%): the executor contributes a substantial share of any agent's performance. MOSAIC's gains, however, are obtained at fixed Qwen3-VL-8B, so they are attributable to the pipeline rather than the executor model.

5.3. The Long-Horizon Failure Mode

Decomposing the aggregate gain by subset reveals what MOSAIC's per-task selection actually solves. Original EmbodiedBench's stock five-module agent collapses to 7.78% on Long Horizon, while every baseline with a planning stage — EmbodiedBench (Decompose), All-modules, and MOSAIC— recovers into the mid-40s on the same subset, clustered within a few points. MOSAIC's 5.3-point advantage over All-modules therefore comes not from Long Horizon (where the two are tied), but from the four shorter subsets, where MOSAIC selects a leaner pipeline per task while All-modules runs all 12 modules on every episode.

5.4. Diversity at the Right Layer

Two ablations isolate the contribution of the long-term memory bank $\mathcal{B}$ and the retrieval signal that drives it (Tables 2, 3). Both ablations preserve the per-episode race of N=3 candidate pipelines with K=3 replicates each, and differ only in whether the meta-agent retrieves from $\mathcal{B}$, and how.

Long-term memory rescues Long Horizon. The aggregate gain from memory is modest (Table 2), but it decomposes into a large rescue on Long Horizon balanced against small losses on each of the four shorter subsets; the same pattern reproduces, more strongly, on the training split (Table 3). On Long Horizon, the no-memory variant underperforms even the static All-modules baseline despite enabling the same 12-module set. The cause is parameter jitter: lacking prior evidence about which parameter bindings work, the meta-agent varies them per task (Fig. 3, left); on long episodes this per-step inconsistency compounds, and the no-memory variant frequently exhausts the 20-step budget. With memory, the meta-agent retrieves prior records of winning bindings and applies them, recovering performance on this subset.

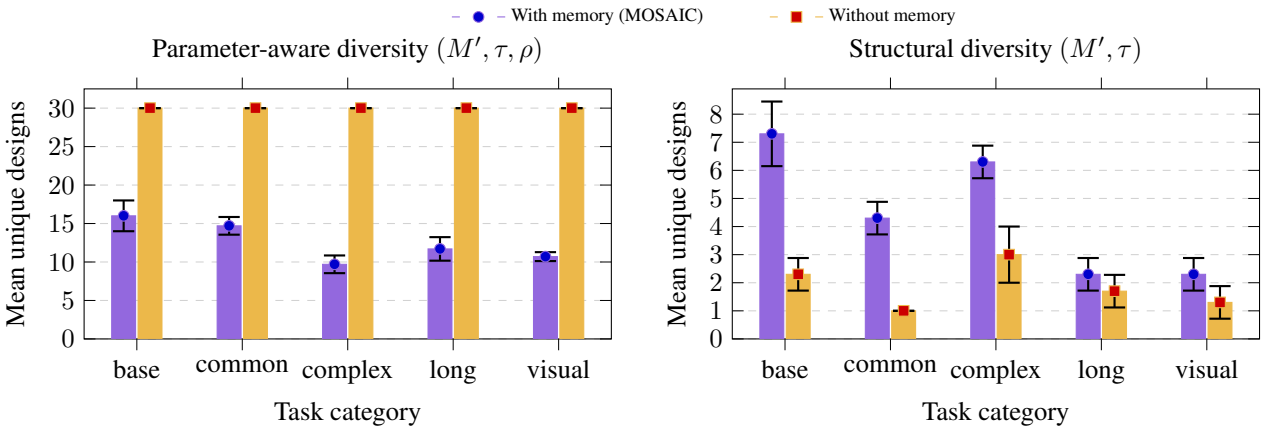


Figure 3. **Memory relocates diversity** from the parameter level to the structural level.

The per-subset losses are a fixable artifact. The losses are reproducible across both splits and all replicates: they are not noise. They trace to our reflection prompt — it overemphasized the parse-error term in the shaped reward, so modules included in designs with high parse-error spikes were labeled losers even when they were not the cause, causing the meta-agent to over-prune those modules on test scenes where they would have helped. A reflection prompt that attributes parse-error pressure more carefully to specific modules should reduce this overfitting; the net effect of the current configuration remains positive.

Memory relocates diversity to the structural layer. Figure 3 characterizes what changes about the meta-agent's behavior with memory. We measure design diversity at two granularities: *parameter-aware* (M', τ, ρ) counts unique full configurations, and *structural* (M', τ) counts unique (modules, tool-choice) shapes with parameter bindings stripped. Without memory, the meta-agent emits a fresh parameter-level configuration on essentially every episode but commits to a small set of structural shapes (it jitters bindings around a fixed skeleton). With memory, parameter-level diversity contracts while structural diversity expands by a comparable factor. Memory does not shrink the meta-agent's search space; it relocates the search from the parameter level, where empirical signal is weak, to the structural level, where past episodes provide direct evidence about which module combinations earn their tokens. Text similarity carries most of this signal: text-only retrieval (α_{img}=0) trails full retrieval by less than 1 pp on aggregate (Table 2).

6. Conclusion

We formulated embodied-agent design as memory-conditioned per-task module-graph synthesis: a joint search over which modules to include, which concrete tool implements each, and how each tool's inputs bind to environment observations and upstream outputs. Solving this search by full self-evolution is infeasible in embodied control, where validating any candidate requires a multi-step rollout and the credit signal arrives only at episode end. MOSAIC retains the load-bearing slice of self-evolution and drops the rest: the typed module catalog, the meta-agent, and the model weights stay fixed, while a memory of past designs and their outcomes evolves with deployment.On EmbodiedBench Navigation, MOSAIC achieves 50.0% average success across five capability subsets, showing that performance gains come from both the module catalog and task-specific module selection.

Limitations. We flag four limitations of the present work. First, MOSAIC's improvement loop is deliberately narrow: only memory evolves online, while the catalog, the meta-agent, and the model weights stay fixed. Second, all validation is on the EmbodiedBench simulator family (EB-Nav only); transfer to other domains or to physical robots remains to be shown. Third, the race used at training time costs $N \cdot K$ rollouts per task — roughly $9\times$ a single-policy baseline at training time — though deployment cost is matched. Finally, the per-task synthesis cost amortizes over the episode horizon; for tasks much shorter than a typical episode, the once-per-task design call no longer dilutes meaningfully and the benefit shrinks.

Outlook. Two natural extensions follow. First, online catalog growth would let MOSAIC add new tools or modules in response to recurring failure modes, narrowing the gap to fully self-evolving agents. Second, richer per-design summaries — cross-design comparison statistics, factor effects across tasks, calibrated critic confidence — could further sharpen the meta-agent's proposals as the memory bank grows.

References

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016. 2

[2] Zhipeng Cai, Ching-Feng Yeh, Hu Xu, Zhuang Liu, Gregory Meyer, Xinjie Lei, Changsheng Zhao, Shang-Wen Li, Vikas Chandra, and Yangyang Shi. Depthlm: Metric depth from vision language models. *arXiv preprint arXiv:2509.25413*, 2025. 1, 2, 3

[3] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14455–14465, 2024. 1, 2, 3

[4] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023. 3

[5] Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024. 3

[6] Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. Learning to route llms with confidence tokens. *arXiv preprint arXiv:2410.13284*, 2024. 3

[7] Pengfei Du. Memory for autonomous llm agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026. 3

[8] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025. 3

[9] Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024. 3

[10] Kostas Hatalis, Despina Christou, and Vyshnavi Kondapalli. Review of case-based reasoning for llm agents: theoretical foundations, architectural components, and cognitive integration. *arXiv preprint arXiv:2504.06943*, 2025. 3

[11] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, pages 21344–21377, 2025. 3

[12] James Y Huang, Sheng Zhang, Qianchu Liu, Guanghui Qin, Tinghui Zhu, Tristan Naumann, Muhao Chen, and Hoifung Poon. Be my eyes: Extending large language models to new modalities through multi-agent collaboration. *arXiv preprint arXiv:2511.19417*, 2025. 3

[13] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023. 2

[14] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li E Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024. 1, 3

[15] Xinghang Li, Peiyan Li, Long Qian, Minghuan Liu, Dong Wang, Jirong Liu, Bingyi Kang, Xiao Ma, Xinlong Wang, Di Guo, Tao Kong, Hanbo Zhang, and Huaping Liu. What matters in building vision-language-action models for generalist robots, 2026. 1, 2, 3

[16] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Song XiXuan, Yifan Xu, Shudan Zhang, Hanyu Lai, Jiadai Sun, Xinyue Yang, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. In *International Conference on Learning Representations*, pages 95650–95707, 2025. 1, 3

[17] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36:43447–43478, 2023. 3

[18] Xinbei Ma, Yiting Wang, Yao Yao, Tongxin Yuan, Aston Zhang, Zhuosheng Zhang, and Hai Zhao. Caution for the environment: Multimodal llm agents are susceptible to environmental distractions. *arXiv preprint arXiv:2408.02544*, 2024. 1

[19] Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic ai for spatial reasoning with a dynamic api. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19446–19455, 2025. 1, 3

[20] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024. 3

[21] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023. 3

[22] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023. 3

[23] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024. 2

[24] Cheng Qian, Zuxin Liu, Shirley Kokane, Akshara Prabhakar, Jielin Qiu, Haolin Chen, Zhiwei Liu, Heng Ji, Weiran Yao, Shelby Heinecke, et al. xrouter: Training cost-aware llms orchestration system via reinforcement learning. *arXiv preprint arXiv:2510.08439*, 2025. 3

[25] Amir Saeidi, Venkatesh Mishra, Souradeep Mukhopadhyay, Gaowen Liu, Ali Payani, Jayanth Srinivasa, and Chitta

Baral. Fama: Failure-aware meta-agentic framework for open-source llms in interactive tool use environments. *arXiv preprint arXiv:2604.25135*, 2026. 1, 3

[26] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023. 2

[27] Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, pages 3841–3865, 2025. 2, 3

[28] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023. 3

[29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023. 3

[30] Jinwei Su, Yinghui Xia, Qizhen Lan, Xinyuan Song, Yang Jingsong, Lewei He, and Tianyu Shi. Difficulty-aware agent orchestration in llm-powered workflows. *arXiv e-prints*, pages arXiv–2509, 2025. 3

[31] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245, 2023. 3

[32] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. 3

[33] Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-g\" odel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. *arXiv preprint arXiv:2510.21614*, 2025. 3

[34] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. 3

[35] Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing. *arXiv preprint arXiv:2602.04837*, 2026. 2, 3

[36] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025. 3

[37] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, pages 12028–12068, 2024. 3

[38] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *Advances in Neural Information Processing Systems*, 37:21875–21911, 2024. 1

[39] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025. 1, 3, 6

[40] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 2

[41] Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, 2025. 3

[42] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. 3

[43] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, pages 34040–34077, 2025. 3

[44] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents, 2026. 2, 3

[45] Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026. 2, 3

[46] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19632–19642, 2024. 3

[47] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. 3

[48] Filippo Ziliotto, Tommaso Campari, Luciano Serafini, and Lamberto Ballan. Tango: training-free embodied ai agents for open-world tasks. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24603–24613, 2025. 3

MOSAIC: The Right Modules for Each Task in Embodied Agents

Supplementary Material

A. Module Catalog

Table 4 enumerates the 12-module catalog $\mathcal{M}$ used in our experiments (referenced from §4.3). For each module we list its kind (deterministic helper or LLM-based), role and output type, and the variants from which the meta-agent may choose when composing a per-task pipeline $\pi = (M', \tau, \rho)$.

Module	Kind	Role / output	Variants
`stm`	deterministic	Projects the episode short-term memory into structured action, pose, target, and seen-object state.	full / lean
`blocker_state`	deterministic	Summarizes recent failed or blocked motions, collisions, and repeated action patterns.	–
`stm_bearing`	deterministic	Converts memory targets into bearing/recency features; optionally uses geometry.	geometric / non-geometric
`goal_object_proposer`	LLM	Proposes likely target object types from the instruction; gives perception a task-conditioned search list.	narrow / default / wide
`object_detector`	deterministic	Produces annotated object detections and a legend from instance detections.	detector choice
`depth_estimator`	deterministic	Provides a depth image or depth description as an additional perceptual channel.	estimator choice
`visual_state_description`	LLM	Describes the current scene from visual and structured inputs.	prompt/tool choice
`perception_reconciler`	LLM	Reconciles visual state, blocker signals, and detector/depth evidence.	prompt/tool choice
`reflection`	LLM	Explains what worked or failed in the recent interaction history.	prompt/tool choice
`reasoning`	LLM	Produces high-level task reasoning and next-step strategy.	prompt/tool choice
`language_plan`	LLM	Produces a natural-language action plan.	prompt/tool choice
`executable_plan`	LLM	Terminal module; emits machine-readable action candidates.	prompt/tool choice

Table 4. Concrete MOSAIC module catalog (referenced from §4.3). The meta-agent selects which modules run, which variants are active, and which input channels each module receives; execution order is fixed by the catalog and dependency closure.

B. Pipeline Layout and Dependency Structure

Figure 4 lays out the same catalog by execution tier and shows the dependency DAG that constrains pipeline composition. Tier 1 is deterministic helpers (no LLM call); Tiers 2 and 3 are LLM-based perception and reasoning/planning modules respectively. Each box reports the module's kind, available variants, output type, and the source classes for its required parameters — environment observation, upstream module output, or design-time configuration. A per-task pipeline $\pi = (M', \tau, \rho)$ selects which subset $M' \subseteq \mathcal{M}$ runs, which variant of each (τ), and which input channels are wired (ρ), subject to the DAG.

Per-call inputs (shared, all from environment / config): `rgb_image`[E] `action_list`[E] `task_goal`[E] `system_prompt_role`[C] `json_warning`[C] `action_descriptions`[C] `module_examples`[C] `strategy_*`[C] `action_budget`[C] *(LLM modules attach `rgb_image` as an image; deterministic helpers consume it as PNG.)*

Tier 1 — deterministic helpers (no LLM call)

stm
kind: det
var: full / lean
out: `MemorySnapshot`
srcs: [E] (history)

blocker_state
kind: det
var: —
out: `BlockerState`
srcs: [U] (stm)

stm_bearing
kind: det
var: geometry / no-geom
out: `Bearings`
srcs: [U] (stm)

object_detector
kind: det
var: (detector)
out: `AnnotPNG+legend`
srcs: [E] (rgb,obj_list)

depth_estimator
kind: det
var: (estimator)
out: `DepthPNG+desc`
srcs: [E] (rgb)

Tier 2 — LLM perception

goal_object_proposer
kind: LLM
var: narrow / default / wide
out: `list[str]`
srcs: [E] (task) [C] (prompt)

visual_state_description
kind: LLM
var: (prompt/tool)
out: `str` (scene desc.)
srcs: [E] [U] (stm,gop,obj,dpt)

perception_reconciler
kind: LLM
var: (prompt/tool)
out: `str` (reconciled)
srcs: [U] (blk, vsd)

Tier 3 — LLM reasoning & planning

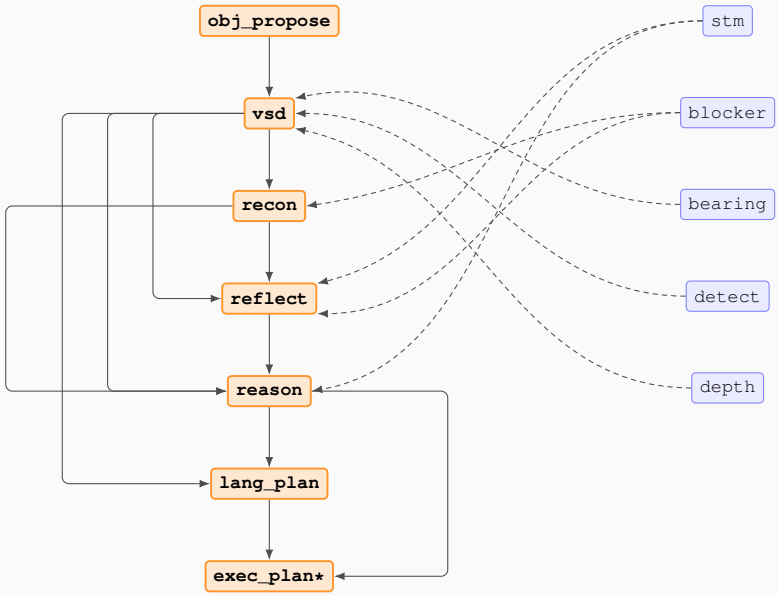
reflection
kind: LLM
var: (prompt/tool)
out: `str` (lessons)
srcs: [U] (stm,blk,sbg,vsd,prc)

reasoning
kind: LLM
var: (prompt/tool)
out: `str` (next plan)
srcs: [U] (vsd,prc,gop,rfl)

language_plan
kind: LLM
var: (prompt/tool)
out: `str` (NL plan)
srcs: [U] (vsd,rsn,rfl) [C]

executable_plan*
kind: LLM*
var: (prompt/tool)
out: `list[Action]`
srcs: [U] (vsd,rsn,rfl,lpn)

Dependency DAG.
Edges show the consume-from relation U_m for the configured pipeline; unlisted edges from per-call broadcast inputs are omitted.

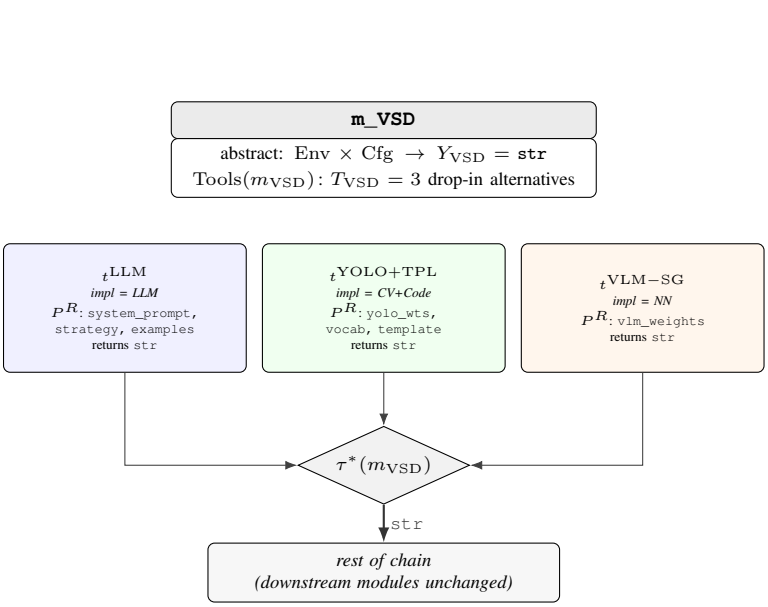


Per-design choice: any subset $M' \subseteq \mathcal{M}$ that contains `executable_plan` and respects U_m.

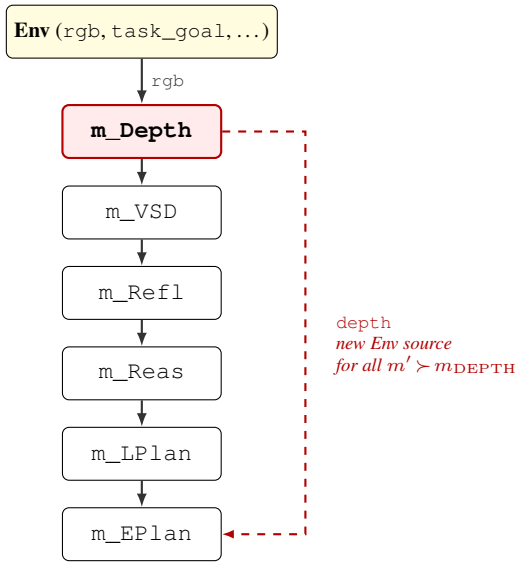
Figure 4. **The 12-module catalog as an instance of the framework (§4.3).** Modules are grouped by execution tier and by the implementation kind of their tools: deterministic helpers (blue, no LLM call), LLM perception, and LLM reasoning & planning. Each box names a module m with the tool(s) that may implement it: its output type Y_m — the interface downstream modules bind against (Def.2) — the tools the meta-agent may assign through $\tau(m) \in \text{Tools}(m)$ (boxes marked `(prompt/tool)` vary by prompt or backbone, i.e. by configuration Cfg_t rather than by enumerated tools), the tool's kind (Def.1), and the source classes of the tool's required inputs $\text{In}(\tau(m))$: [E] environment channel, [U] output of an allowed-consume module $m' \in U_m$, [C] design-time configuration. A per-task pipeline $\pi = (M', \tau, \rho)$ selects (a) which subset $M' \subseteq \mathcal{M}$ to enable, (b) a tool $\tau(m)$ for each, and (c) the wiring ρ binding every $p \in \text{In}(\tau(m))$ to an [E]/[U]/[C] source, subject to the dependency DAG. `executable_plan` is the only terminal module and is always enabled.

C. Framework Extensibility

Figure 5 illustrates the two axes along which the design space of §3.2 can be extended without altering Problem 3.2. **Panel (a)** grows $|\text{Tools}(m)|$ for a single module: three drop-in implementations of `m_VSD` — a frozen-LLM call, a YOLO-plus-template classical CV pipeline, and a fine-tuned vision-language scene-graph model — share the same output type, so only the tool selector τ changes downstream. **Panel (b)** grows $|\mathcal{M}|$ itself: a new depth-synthesis module m_{DEPTH} inserted upstream of m_{VSD} adds `depth` as a new valid source for every downstream module, with no change to existing module signatures.



(a) Growing $|\text{Tools}(m)|$. Three drop-in alternatives share Y_m = `str`; only $\tau^*(m_{\text{VSD}})$ changes.



(b) Growing $|\mathcal{M}|$. m_{DEPTH} extends Env for all downstream m': src(`depth`) : Env → Env ∪ $\{m_{\text{DEPTH}}.\text{out}\}$.

Figure 5. **Two axes of extension of the framework of §3.2. (a)** Growing $|\text{Tools}(m)|$. Three alternative implementations of `m_VSD` — a frozen-LLM call, a YOLO-plus-template classical CV pipeline, and a fine-tuned vision-language scene-graph model — all satisfy $Y_{\text{VSD}} =$ `str` and therefore slot into the downstream chain interchangeably; only $\tau^*(m_{\text{VSD}})$ changes under Problem 1. **(b)** Growing $|\mathcal{M}|$. A new depth-synthesis module m_{DEPTH} consuming `rgb` from the environment is inserted upstream of m_{VSD}. Its output becomes a new valid source for the `depth` parameter of every downstream module — no downstream signature changes. Both (a) and (b) are *expansions* of the search space of §3.2, not alterations to Problem 1.

D. Design Diversity Across Train and Test Splits

Figure 6 complements the with/without-memory comparison of Figure 3 (main paper, §5.4) with a train/test consistency check. Both bars in this figure use MOSAIC with long-term memory enabled: blue bars report design diversity on the training tasks (where memory is built by a single pass over the curriculum), and orange bars report the same metrics on held-out test tasks (where memory is queried). Structural diversity is essentially identical across splits, indicating that MOSAIC's design profile does not depend on whether memory was *accumulated* on the encountered tasks or only *retrieved* for them; the residual gap on parameter-aware counts reflects the larger pool of training-time exploration before memory consolidates.

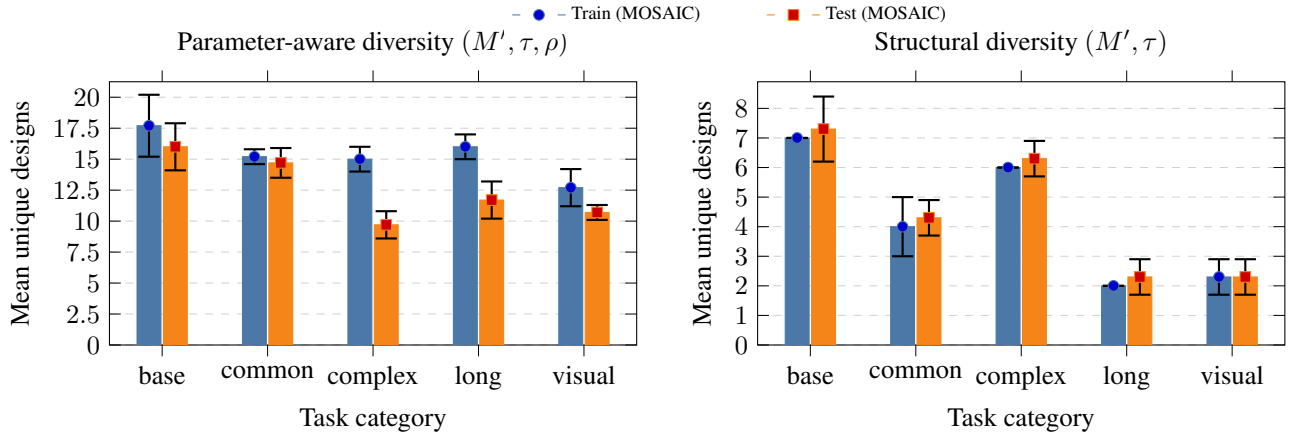


Figure 6. **Train/test consistency of MOSAIC's design diversity.** Unique-design counts on the five EB-Nav subsets (Base, Common Sense, Complex Instruction, Long Horizon, Visual Appearance), reported at the two granularities used in the main paper (Fig. 3): *left* – parameter-aware (M', τ, ρ) counts unique full configurations; *right* – structural (M', τ) counts unique (modules, tool-choice) shapes with parameter bindings stripped. Blue bars report MOSAIC on the training tasks; orange bars report MOSAIC on the held-out test tasks. Both are MOSAIC with long-term memory enabled. Error bars are one standard deviation across $n = 3$ replicates.

E. Depth Augmentation Does Not Replace Per-Task Selection

Table 5 probes whether explicit depth information can substitute for MOSAIC's per-task selection. Two fixed-pipeline configurations are added as references: *Depth Estimation* enables an estimated-depth channel, and *GT Depth* substitutes the simulator's ground-truth depth. Both are evaluated against MOSAIC, which decides per task whether to include a depth-bearing module. Even GT Depth, which removes all estimation error, lands 3.67 pp below MOSAIC on aggregate; the estimated-depth variant lands 6 pp below. Two takeaways follow: (i) higher-fidelity depth alone does not close the gap to MOSAIC— the gain is not driven by a missing perceptual channel that can be patched by giving more information; (ii) on Long Horizon specifically, both depth variants underperform MOSAIC, consistent with §5.4's finding that the dominant lever on long episodes is parameter binding stability rather than additional perceptual input.

Configuration	Avg	Base	Common	Complex	Visual	Long
Estimation Depth	44.00	56.67	31.67	41.67	46.67	43.33
GT Depth	46.33	50.00	46.67	50.00	43.34	41.67
MOSAIC (ours)	50.00 $\pm$ 1.33	60.00 $\pm$ 8.82	46.67 $\pm$ 0.00	45.56 $\pm$ 10.18	51.11 $\pm$ 5.09	46.67 $\pm$ 3.33

Table 5. **Depth-augmentation ablation on EB-Nav (test split).** Per-subset success rate and *Avg* across the five EB-Nav subsets (Base, Common Sense, Complex Instruction, Visual Appearance, Long Horizon). *Depth Estimation* enables an estimated-depth channel; *GT Depth* substitutes ground-truth depth from the simulator. Due to time-constraints, we only report an average over two runs for GT and estimation depth. The *Avg* column is the mean across runs and subsets. MOSAIC (ours) uses per-task module selection and reports mean $\pm$ standard deviation over $n = 3$ replicates. Executor is Qwen3-VL-8B for all rows; meta-agent (for MOSAIC) is Claude Sonnet 4.6.