CoWA: Learning Compact Decision Policies from Controller-Executed Workflows

Anonymous authors
Paper under double-blind review

Abstract

Large language models (LLMs) are moving from open-ended reasoning toward constrained system control, where agents transform structured states into controller-executable actions that optimize system utility under operational constraints. This shift calls for compact LLM agents that can operate under tight latency, cost, and privacy requirements without repeatedly querying cloud models. Existing methods study tool interaction, trajectory supervision, and post-training adaptation, but leave a key design question unresolved: which parts of a domain workflow should be executed by the controller, and which should be learned by the model? We propose **Co**ld-start **W**orkflow **A**lignment (**CoWA**), a framework that separates fixed domain computation from final-action learning. CoWA performs task and interface alignment, tool-augmented cold-start supervision over a fixed controller-executed workflow, and GRPO-based objective alignment. The compact policy learns to synthesize the final action from tool-derived observations, while the controller retains responsibility for tool execution and post-generation checks. This design supports constraint-aware decision-making without iterative per-instance utility search. We instantiate CoWA in a representative Wi-Fi channel-allocation case study. Using 750 training trajectories, a CoWA-trained Llama-3.1-8B achieves an average throughput of 293.5 Mbps, compared with 291.8 Mbps for a prompt-engineered GPT-5.2 baseline, with an observed $2.69\times$ end-to-end wall-clock speedup in our evaluation setup. CoWA also approaches the performance of the search-based baseline without using its search budget.

1 Introduction

Large language models (LLMs) are increasingly used in agents that interact with tools and external environments Xi et al. (2025); Wang et al. (2024). This development raises a practical question for resource management: *can an LLM turn structured system observations into useful control actions?* In these applications, an action must satisfy operational constraints, and its quality is measured by system performance. For example, a wireless channel allocation must provide each station with a valid link while accounting for contention among stations. Repeated decisions also make inference cost and latency relevant. Compact models offer a path toward local decision-making by learning high-quality actions from domain-specific inputs Zheng et al. (2025); Qu et al. (2025).

Prior work provides several foundations for this goal. ReAct, Toolformer, and Voyager study how models interact with tools Yao et al. (2023); Schick et al. (2023); Wang et al. (2023), while FireAct and AgentTuning investigate training on agent trajectories Chen et al. (2023); Zeng et al. (2023). Parameter-efficient adaptation and reward-based post-training provide practical means to specialize a pretrained model Hu et al. (2022); Dettmers et al. (2023); Shao et al. (2024). In wireless control, prior work on LLM-assisted Wi-Fi multi-link operation (MLO) channel allocation combines domain knowledge with search Lian et al. (2025). This line of work provides a concrete decision problem and a search-based comparison for studying learned allocation policies.

However, a remain issue in this setting is which parts of the decision workflow a compact model needs to learn. Domain tools can compute distances, identify conflicts, and construct initial assignments through a fixed procedure. These operations need not be selected or reproduced by the model at every decision. However, their outputs do not by themselves resolve the final allocation: improving one's assignment can change contention for others, and the complete action must satisfy

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

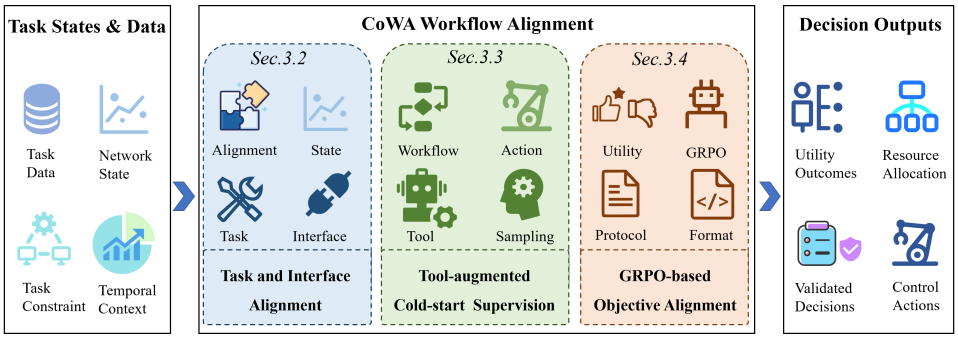

Figure 1: The overview of CoWA. Task and interface alignment converts task data, structured system states, and constraints into a controller-compatible input and action schema. The controller then executes a fixed tool workflow, and tool-augmented cold-start supervision trains the compact policy to synthesize the final action from the resulting observations. GRPO-based objective alignment further optimizes task utility, protocol feasibility, and output-format validity.

allocation constraints. Training only on state–action pairs leaves these intermediate calculations implicit, whereas learning the full tool interaction also requires modeling operations that the controller can already execute. We therefore ask: *can a compact model learn effective constrained decisions by conditioning on a fixed, controller-executed workflow and learning only the final action?*

We study this division of tool execution and decision learning through **Co**ld-start **W**orkflow **A**lignment (**CoWA**), shown in Fig. 1. Here, *workflow alignment* means training the final-action policy on the tool observations supplied by the controller, with an output schema and objective matched to the target task. CoWA first specifies the state, constraints, tools, and required action format. The controller then executes a fixed tool workflow, and supervised fine-tuning trains the model to generate the final action from the resulting history. This supervision provides the cold-start initialization for group relative policy optimization (GRPO) Shao et al. (2024), which uses task utility, format validity, and action feasibility as reward signals. The model learns the final decision; the controller retains responsibility for intermediate tool execution.

This design exposes domain calculations to the policy and focuses learning on how to use their results. The transition from a tool workflow to a learned decision remains non-trivial: demonstrations provide limited coverage of possible allocations, and matching their output format does not ensure high utility or feasibility. CoWA therefore combines supervised initialization with reward-based refinement. At inference time, it generates a final allocation from the tool context without iterative per-instance utility search. The tools still incur computation, and validity rewards encourage constraint satisfaction rather than guarantee it.

We evaluate CoWA in simulated Wi-Fi 7 MLO channel allocation Chen et al. (2022); Carrascosa-Zamacois et al. (2023). With 750 cold-start trajectories, CoWA with Llama-3.1-8B achieves 293.5 Mbps average throughput, compared with 291.8 Mbps for prompt-engineered GPT-5.2 and 297.2 Mbps for BAI-MCTS under the reported search budgets. Removing tool augmentation or GRPO reduces throughput by 29.5 or 30.7 Mbps, respectively. These ablations support the usefulness of tool context and reward-based refinement in this task. End-to-end runtime averages 36.50 s for CoWA and 98.15 s for GPT-5.2, an observed 2.69× speedup in our setup. The GPT-5.2 baseline does not receive CoWA's tool history, and the runtime comparison is not hardware-normalized; these results compare the evaluated systems rather than isolate model capability or compute efficiency.

Our contributions are: 1) We study a division of computation and learning for constrained system control: a controller executes a fixed domain workflow, and a compact LLM learns the final action conditioned on its observations. 2) We develop CoWA to implement this design through a controller-compatible task interface, tool-conditioned supervised initialization, and reinforcement learning with utility and validity rewards. The resulting policy generates final actions without iterative per-instance utility search. 3) We provide a Wi-Fi 7 MLO case study examining throughput, allocation omissions and conflicts, cross-scale behavior, and runtime. The results show competitive throughput against the evaluated frontier-model and search baselines, while ablations quantify the effects of tool augmentation, reward-based refinement, and validity rewards.

2 SYSTEM MODEL

We consider a constrained system where a controller observes the current state and selects a resource-allocation action. The structured state $s \in \mathcal{S}$ contains the entities, available resources, and operating conditions needed for the decision. The constraints Ψ define the feasible action set $\mathcal{A}(s)$. A domain evaluator assigns utility $U(s, a)$ to an allocation, such as aggregate network throughput. We study the allocation at each decision epoch, with the per-instance objective

$$a^\star(s) \in \arg\max_{a \in \mathcal{A}(s)} U(s, a). \tag{1}$$

For a distribution $\mathcal{D}$ of system states, the corresponding policy objective is defined as

$$\max_{\pi \in \Pi_{\text{feas}}} \mathbb{E}_{s \sim \mathcal{D}}\big[U(s, \pi(s))\big], \tag{2}$$

where Π_{feas} denotes decision rules satisfying $\pi(s) \in \mathcal{A}(s)$ for states under consideration. This objective specifies the desired system behavior.

The decision system consists of the controller, a domain tool set $\mathcal{F}$, and a compact LLM. A formatter g describes the state, constraints, tools, and required output schema Σ:

$$x = g(s; \Psi, \mathcal{F}, \Sigma), \qquad \tau = \mathrm{Rollout}(x; \mathcal{F}). \tag{3}$$

Here, $\mathrm{Rollout}$ is a fixed controller-executed workflow, and τ records its tool calls and observations before the final decision. The model with parameters θ generates a candidate action, i.e.,

$$\hat{a} \sim \pi_\theta(\cdot \mid x, \tau). \tag{4}$$

The controller checks both compliance with Σ and membership in $\mathcal{A}(s)$. The system's goal is to train this final-action policy to achieve high utility and satisfy the constraints, while retaining fixed tool execution and avoiding iterative per-instance utility search. The training reward uses a normalized utility $\bar{U}(s, a) \in [0, 1]$, distinguished from the physical utility $U(s, a)$ reported in evaluation.

3 THE COWA FRAMEWORK

3.1 OVERVIEW

The system model separates fixed domain computation from the learned final decision. CoWA realizes this separation by addressing three gaps between a pretrained LLM and a constrained control policy. First, an *interface gap* arises because raw system states and free-form responses do not directly match a controller's input and output requirements. Second, a *workflow gap* arises because a compact model may not reliably reproduce domain calculations. Third, an *objective gap* arises because next-token supervision does not directly optimize system utility or action validity.

As shown in Fig. 2, CoWA assigns one training stage to each gap. Task and interface alignment first constructs $x = g(s; \Psi, \mathcal{F}, \Sigma)$, preserving the state, constraints, available tools, and output schema in a controller-compatible representation. Starting from this interface, the controller executes a fixed workflow and records its observations in τ. Tool-augmented cold-start supervision then trains the policy to map (x, τ) to a demonstration action. Finally, objective alignment refines the initialized policy using system utility and validity feedback. Next, we explain them in details.

3.2 TASK AND INTERFACE ALIGNMENT

For a state s_i, the task interface is defined as $x_i = g(s_i; \Psi, \mathcal{F}, \Sigma)$. The constraint specification Ψ determines $\mathcal{A}(s_i)$, while Σ defines how an action is serialized and parsed. These two objects serve different purposes: schema compliance makes an output readable by the controller, whereas feasibility requires the action to belong to $\mathcal{A}(s_i)$. The tool registry is $\mathcal{F} = \{f_1, f_2, \ldots, f_K\}$, where each f_k is a deterministic or externally implemented domain operation, such as state analysis, candidate construction, or constraint checking. The interface exposes the tool definitions to the controller and serializes their returned observations for the model. As a result, the model should learn how to use the resulting observations when producing the final action.

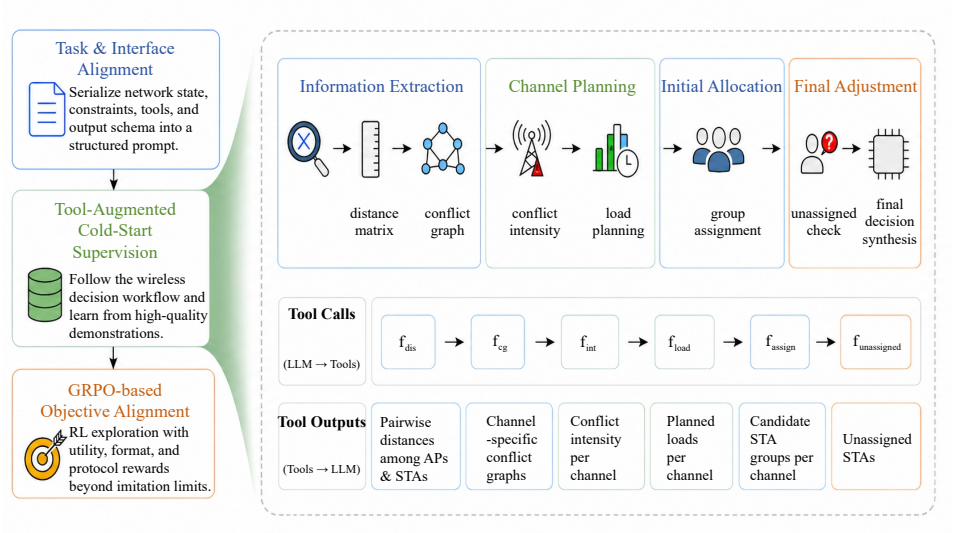

Figure 2: The CoWA pipeline. A controller-compatible interface defines the task, a fixed tool workflow produces the history τ, and a compact policy learns the final action. The right panel shows the workflow used in the Wi-Fi MLO case study.

3.3 Tool-Augmented Cold-Start Supervision

Given interface x_i, the controller runs an ordered workflow defined for the target domain:

$$\tau_i = \mathrm{Rollout}(x_i; \mathcal{F}). \tag{5}$$

The history τ_i contains the ordered tool calls and observations and stops before the final decision. CoWA therefore treats the workflow as input rather than as a sequence of actions to be imitated. Each training example is $(x_i, \tau_i, a_i^{\mathrm{dem}})$, where $a_i^{\mathrm{dem}} \in \mathcal{A}(s_i)$ is a retained demonstration expressed under Σ. We use a_i^{dem} to distinguish the training target from the per-instance optimum $a^\star(s_i)$.

Because a feasible demonstration can still have poor utility, we apply utility-based rejection sampling. Let U_i^{ref} and U_i^{base} denote the utilities of a reference strategy and a task-specific baseline. When $U_i^{\mathrm{ref}} > U_i^{\mathrm{base}}$, we retain a demonstration only if its normalized utility gain satisfies

$$q_i(a_i^{\mathrm{dem}}) = \frac{U(s_i, a_i^{\mathrm{dem}}) - U_i^{\mathrm{base}}}{U_i^{\mathrm{ref}} - U_i^{\mathrm{base}}} \geq \eta, \tag{6}$$

where $\eta \in (0, 1]$ is the filtering threshold. This form measures how much of the reference improvement over the baseline is preserved by the demonstration. The retained dataset is $\mathcal{D}_{\mathrm{cs}} = \{(x_i, \tau_i, a_i^{\mathrm{dem}})\}_{i=1}^{N}$. Supervised fine-tuning minimizes

$$\mathcal{L}_{\mathrm{cs}}(\theta) = -\sum_{(x_i, \tau_i, a_i^{\mathrm{dem}}) \in \mathcal{D}_{\mathrm{cs}}} \log \pi_\theta(a_i^{\mathrm{dem}} \mid x_i, \tau_i). \tag{7}$$

This objective teaches the relation between tool observations and complete actions, but its performance remains limited by the quality and coverage of the demonstrations. We denote the resulting parameters by θ_{cs} and use them to initialize objective alignment.

3.4 GRPO-Based Objective Alignment

The final stage optimizes generated actions using feedback from the same evaluator and constraints that define the system model. For a candidate $\hat{a}$ under state s, CoWA uses

$$r(s, \hat{a}) = \omega_1 \bar{U}(s, \hat{a}) + \omega_2 V_{\mathrm{fmt}}(\hat{a}; \Sigma) + \omega_3 V_{\mathrm{act}}(s, \hat{a}), \tag{8}$$

where $\omega_1, \omega_2, \omega_3 \geq 0$ are weighting coefficients, $\bar{U}(s, \hat{a}) \in [0, 1]$ is the normalized task utility score,

$$V_{\mathrm{fmt}}(\hat{a}; \Sigma) = \mathbf{1}[\mathrm{Format}(\hat{a}; \Sigma) = 1] \tag{9}$$

checks whether the output satisfies the required schema, and

$$V_{\text{act}}(s, \hat{a}) = \mathbf{1}[\hat{a} \in \mathcal{A}(s)] \tag{10}$$

checks whether the parsed action belongs to $\mathcal{A}(s)$. Notice that the weighted reward encourages utility, parseability, and feasibility. For each context (x_i, τ_i), an old policy $\pi_{\theta_{\text{old}}}$ samples a group of candidates,

$$\{\hat{a}_i^{(k)}\}_{k=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x_i, \tau_i), \tag{11}$$

and each candidate receives $r_i^{(k)} = r(s_i, \hat{a}_i^{(k)})$.

We compute group-relative advantages by normalizing rewards within each sampled group, with a small constant ϵ added to the denominator for numerical stability:

$$A_i^{(k)} = \frac{r_i^{(k)} - \text{mean}_{k'}(r_i^{(k')})}{\text{std}_{k'}(r_i^{(k')}) + \epsilon}. \tag{12}$$

Let

$$\rho_i^{(k)} = \frac{\pi_\theta(\hat{a}_i^{(k)} \mid x_i, \tau_i)}{\pi_{\theta_{\text{old}}}(\hat{a}_i^{(k)} \mid x_i, \tau_i)} \tag{13}$$

denote the importance ratio for the sampled batch. The cold-start policy remains the reference in the KL term. The objective is written as

$$\mathcal{L}_{\text{grpo}}(\theta) = -\mathbb{E}_{i,k}\left[\min\left(\rho_i^{(k)} A_i^{(k)}, \text{clip}\Big(\rho_i^{(k)}, 1-\epsilon_c, 1+\epsilon_c\Big) A_i^{(k)}\right)\right] + \beta D_{\text{KL}}\Big(\pi_\theta \,\|\, \pi_{\theta_{\text{cs}}}\Big). \tag{14}$$

The three stages thus share one decision interface: x_i defines the task, τ_i supplies controller-computed evidence, and the reward evaluates the generated action. The following case study instantiates every object in this interface for Wi-Fi MLO channel allocation.

4 Case Study

4.1 Wi-Fi 7 MLO Channel Allocation

We instantiate CoWA for centralized Wi-Fi 7 MLO channel allocation. The purpose of the case study is to test whether a compact policy can use a fixed wireless-analysis workflow to produce competitive allocations without iterative per-instance utility search. MLO allows a station (STA) to establish links across multiple frequency bands. This flexibility creates a combinatorial allocation problem because stations sharing a channel contend with one another, so a locally useful assignment may reduce aggregate throughput. Search-based methods (*e.g.*, BAI-MCTS) address this coupling by evaluating many candidate allocations Lian et al. (2025). CoWA instead amortizes decision-making across training states and uses one fixed tool rollout followed by final-action generation.

As illustrated in Fig. 3, the controlled system consists of a centralized controller and M basic service sets (BSSs). Each BSS contains one access point multi-link device (AP-MLD) and multiple station multi-link devices (STA-MLDs). Let $\mathcal{M} = \{1, \ldots, M\}$ denote the AP set, $\mathcal{N}_m$ denote the STA set associated with AP m, and the global STA set is $\mathcal{N} = \bigcup_{m \in \mathcal{M}} \mathcal{N}_m$. The available spectrum consists of bands $\mathcal{B} = \{2.4, 5, 6\}$ GHz, where band b has channel pool $\mathcal{C}_b$. Each channel c has carrier-sensing radius R_c^{cs}, and each STA may activate at most L links. The state is defined as

$$s_t = \Big(\{\mathbf{p}_m^{\text{AP}}\}_{m \in \mathcal{M}}, \{\mathbf{p}_n^{\text{STA}}\}_{n \in \mathcal{N}}, \{\mathcal{N}_m\}_{m \in \mathcal{M}}, \{\mathcal{C}_b\}_{b \in \mathcal{B}}, \{R_c^{\text{cs}}\}_{c \in \cup_b \mathcal{C}_b}, L\Big). \tag{15}$$

This state supplies both the raw task input and the quantities required by the wireless tools. Under simultaneous transmit-and-receive MLO, the action for STA n is given by

$$a_n = \{c_{n,b} : b \in \mathcal{B}\}, \quad c_{n,b} \in \mathcal{C}_b \cup \{\varnothing\}, \quad n \in \mathcal{N}. \tag{16}$$

Here, $c_{n,b} = \varnothing$ means that STA n has no active link in band b. Notice that each band can be only occupied by one link. The remaining feasibility constraints are

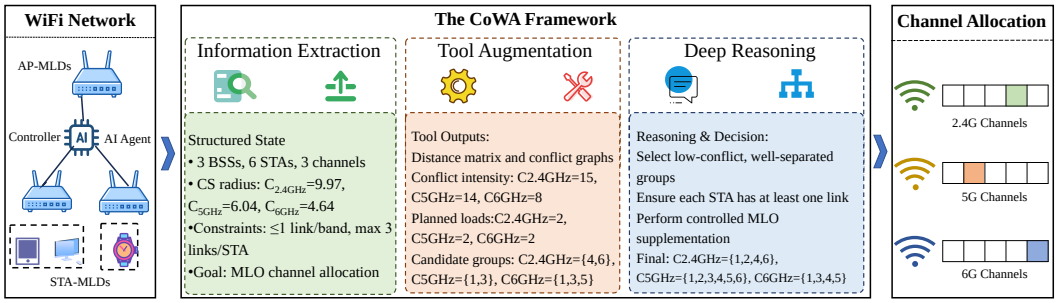

Figure 3: Illustration of the proposed compact LLM agent for Wi-Fi 7 MLO channel allocation.

$$c_{n,b} \in \mathcal{C}_b \cup \{\varnothing\}, \quad \forall b \in \mathcal{B}, \qquad 1 \le \sum_{b \in \mathcal{B}} \mathbf{1}[c_{n,b} \neq \varnothing] \le L. \tag{17}$$

Then, the complete network action is given by

$$a_t = \{a_n\}_{n \in \mathcal{N}} \in \mathcal{A}_{\mathrm{MLO}}(s_t), \tag{18}$$

where $\mathcal{A}_{\mathrm{MLO}}(s_t)$ contains the allocations satisfying these constraints, denoted by Ψ_{MLO}.

Feasibility alone does not determine allocation quality. STAs within R_c^{cs} contend when assigned to the same channel, and each STA's rate therefore depends on the complete allocation. The physical utility in Sec. 2 becomes

$$U_{\mathrm{MLO}}(s_t, a_t) = T(s_t, a_t) = \sum_{n \in \mathcal{N}} T_n(s_t, a_t), \tag{19}$$

where $T_n(s_t, a_t)$ denotes the throughput of STA n under allocation a_t. The resulting constrained control objective is

$$a_t^\star(s_t) \in \arg \max_{a \in \mathcal{A}_{\mathrm{MLO}}(s_t)} U_{\mathrm{MLO}}(s_t, a). \tag{20}$$

The policy objective uses the MLO state distribution $\mathcal{D}_{\mathrm{MLO}}$:

$$\max_{\pi \in \Pi_{\mathrm{feas}}^{\mathrm{MLO}}} \mathbb{E}_{s \sim \mathcal{D}_{\mathrm{MLO}}}[U_{\mathrm{MLO}}(s, \pi(s))]. \tag{21}$$

The number of feasible per-STA channel choices yields

$$|\mathcal{A}_{\mathrm{MLO}}(s)| = \prod_{n \in \mathcal{N}} \left(\sum_{k=1}^{L} \sum_{\substack{\mathcal{B}' \subseteq \mathcal{B} \\ |\mathcal{B}'|=k}} \prod_{b \in \mathcal{B}'} |\mathcal{C}_b| \right). \tag{22}$$

This space grows exponentially with $|\mathcal{N}|$. Since evaluating throughput also requires modeling contention and collisions, exhaustive optimization is unsuitable for repeated decisions. This motivates the workflow-conditioned policy defined next. Appendix A details the instantiated task interface, controller-executed workflow, tool serialization, and output schema.

5 Experiments

The experiments address three questions: (RQ1) whether the complete CoWA pipeline produces competitive system utility; (RQ2) whether the tool context, GRPO refinement, and validity rewards contribute to the observed behavior; (RQ3) how the learned policy behaves beyond the STA scales and what end-to-end runtime is observed relative to the prompted frontier-model baseline.

5.1 Experimental Setup

Task and Data. We study the Wi-Fi 7 MLO allocation problem defined in Sec. 2. The cold-start dataset contains 750 tool-augmented trajectories: 300 examples with 6 STAs and 150 examples each with 12, 18, and 24 STAs. Every trajectory contains the structured network instance, a controller-executed tool prefix, and a target final allocation. The main experiment evaluates these four STA scales, while the cross-scale experiment additionally includes larger STA counts absent from training. Appendix B.1 describes instance generation, trajectory construction, and the separation between hidden reference allocations and model-visible inputs.

Model and Training. An executable PHY/MAC simulator evaluates link quality, contention, and collisions under each generated allocation. The same simulator supplies the throughput component of the GRPO reward. For every reported STA scale, results are averaged over 10 random seeds; Fig. 4 also reports 95% confidence intervals. These experiments evaluate simulated networks and do not include a live Wi-Fi deployment. In addition, CoWA adapts Llama-3.1-8B Grattafiori et al. (2024) as base model. QLoRA-based SFT first learns final-action generation from the tool-conditioned demonstrations. LoRA-based GRPO then refines the SFT checkpoint using normalized throughput, output-format validity, and action feasibility. Both stages use the controller-compatible interface and fixed workflow. More details are given in Appendix B.2. This answers RQ1.

Baselines and Metrics. *Random* randomly selects a channel-allocation candidate and provides a non-optimized reference. *Prompted GPT-5.2* is a frontier-model system that receives the same task definition, topology, channels, carrier-sensing radii, and allocation constraints as CoWA. It does not receive CoWA's controller-executed tool history; its prompt is reproduced in Appendix C.2. *BAI-MCTS* is the search-based optimizer Lian et al. (2025). The metric *Throughput* (↑, Mbps) is the aggregate throughput and instantiates U_{MLO}. *Omission* (↓) is the average number of STAs without any allocated link and directly measures violations of the coverage constraint. *Conflict* (↓) is the average number of same-channel STA pairs within carrier-sensing range. It diagnoses contention in an allocation but is not itself a hard feasibility constraint. *Runtime* (↓) is the end-to-end wall-clock time from task arrival to the final output.

5.2 Main Results (RQ1)

Fig. 4 compares aggregate throughput of four methods at 6, 12, 18, and 24 STAs. Random allocation has the lowest mean at every scale, showing that the task rewards structured allocation rather than arbitrary channel selection. Among the three informed methods, their relative order varies with scale. BAI-MCTS has the highest mean at 6 and 12 STAs, while the three methods are close at 18 STAs. At 24 STAs, CoWA has the highest mean, followed closely by GPT-5.2. CoWA's mean also increases across the four tested scales. In addition, averaged over the four settings, CoWA obtains 293.5 Mbps, compared with 291.8 Mbps for prompted GPT-5.2 and 297.2 Mbps for BAI-MCTS, as indicated by the horizontal dashed lines. Thus, CoWA is 1.7 Mbps above the prompt-only frontier-model system and 3.7 Mbps (about 1.2%) below the search-based optimizer. The BAI-MCTS performs thousands of interactions for every topology, whereas CoWA executes a fixed tool workflow and then generates one final action. These results demonstrate that the complete CoWA pipeline can produce competitive system utility.

5.3 Ablation Study (RQ2)

We study four variants of the CoWA. **w/o GRPO** directly evaluates the cold-start SFT policy and therefore removes reward-based objective alignment. **w/o TA** removes the tool-derived observations, leaving the model to generate the final action from the task description. **w/o RFA** retains GRPO but removes the format-validity and action-feasibility terms, so the reward contains throughput only. **w/ CL** retains the complete method and adds training from smaller to larger networks.

Table 1 first shows the role of objective alignment. Without GRPO, average throughput decreases from 293.5 to 262.8 Mbps, a loss of 30.7 Mbps, while omission increases from 0.050 to 1.125 and conflict increases from 26.917 to 31.000. The SFT demonstrations teach the output task, but their finite coverage does not produce the same utility or coverage behavior as the reward-refined policy. This result supports using SFT as an initialization rather than as the final policy. In addition, the tool context produces a similarly large effect. Without tool augmentation, throughput falls by 29.5 Mbps

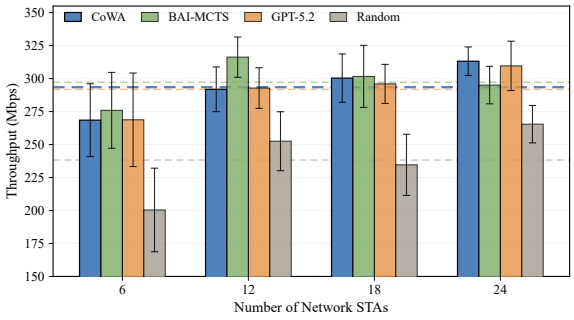

Figure 4: Throughput over the four main STA scales. CoWA generates one final action after its fixed tool workflow, GPT-5.2 uses prompt-only final-action inference, and BAI-MCTS performs per-instance search. Bars show means over 10 random seeds; error bars denote 95% confidence intervals.

Table 1: Ablation results. Numbers in parentheses indicate the absolute change relative to CoWA. Bold values denote the best performance, and underlined values denote the second-best performance.

Method	Avg. Throughput ↑	Avg. Omission ↓	Avg. Conflict ↓
CoWA	<u>293.5</u>	<u>0.050</u>	**26.917**
w/o GRPO	262.8 (-30.7)	1.125 (+1.075)	31.000 (+4.083)
w/o TA	264.0 (-29.5)	0.950 (+0.900)	55.608 (+28.691)
w/o RFA	**293.8** (+0.3)	0.125 (+0.075)	<u>28.159</u> (+1.242)
w/ CL	291.3 (-2.2)	**0.025** (-0.025)	31.292 (+4.375)

Table 2: End-to-end runtime (s) of the CoWA and Prompted GPT-5.2 methods.

Method	Runtime (s)				Average (s)
	6	12	18	24	
CoWA	26.83(2.54×)	33.35(3.07×)	39.58(2.87×)	46.22(2.35×)	**36.50**(2.69×)
GPT-5.2	68.13	102.25	113.43	108.80	98.15

to 264.0 Mbps and omission rises to 0.950. Conflict more than doubles, from 26.917 to 55.608. This comparison indicates that the controller performs deterministic domain calculations, and the policy uses their outputs to form the final decision. Moreover, the validity rewards affect a different part of the behavior. Removing them slightly increases throughput from 293.5 to 293.8 Mbps, but omission rises from 0.050 to 0.125 and conflict rises from 26.917 to 28.159. Throughput alone can therefore prefer allocations with less desirable constraint-related or contention-related properties. Curriculum learning also yields a trade-off: it reduces omission to 0.025, but throughput falls to 291.3 Mbps and conflict rises to 31.292. Overall, the ablations support complementary roles for tool context, reward-based refinement, and validity-aware rewards.

5.4 Generalization and Efficiency (RQ3)

Fig. 5 illustrates the throughput of the CoWA under different network scenarios and training settings. We see that, when the number of STA is less than 24 STAs, CoWA follows a smooth, monotonic increase from 12 to 24 STAs. GPT-5.2 shows a larger local fluctuation between 15 and 18 STAs. At the unseen interpolation points, CoWA remains close to GPT-5.2 and follows the trend formed by its neighboring training scales. Beyond 24 STAs, CoWA is nearly identical to GPT-5.2 at 27 STAs and remains close at 30 STAs. Thus, CoWA preserves competitive throughput when evaluated at network sizes not included in its training trajectories. Moreover, CoWA has a lower standard deviation (24.2 vs. 25.5 Mbps), a lower coefficient of variation (7.83% vs. 8.23%), and a lower standard deviation across setting-level means (14.2 vs. 15.1 Mbps). This evidence indicates that CoWA is stable under interpolation and moderate scale extrapolation within the studied MLO task.

Table 2 compares CoWA with prompted GPT-5.2 with respect to runtime. We see that CoWA averages 36.50 s, compared with 98.15 s for GPT-5.2, giving a runtime ratio of $98.15/36.50 = 2.69$.

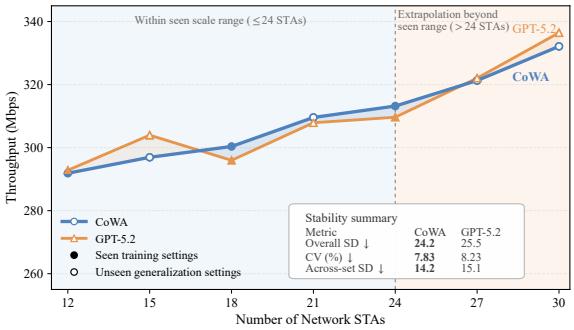

Figure 5: The throughput of the CoWA under different number of network STAs. Filled markers denote training-represented settings (12, 18, and 24 STAs); hollow markers denote unseen interpolation settings (15 and 21 STAs) and extrapolation settings (27 and 30 STAs). The dashed line marks the largest training scale.

CoWA is faster at every reported scale, with ratios ranging from $2.35\times$ to $3.07\times$. Its runtime increases monotonically from 26.83 s at 6 STAs to 46.22 s at 24 STAs, which is consistent with the larger serialized state and tool history processed at larger scales.

6 Related Work

Tool Workflows and Agent Training. ReAct and Toolformer place tool use inside model generation Yao et al. (2023); Schick et al. (2023); FireAct and AgentTuning train on interaction trajectories Chen et al. (2023); Zeng et al. (2023); and ATLAS supervises selected critical steps **?**. CoWA addresses a different boundary: a controller executes every intermediate tool call, observations become policy inputs, and only the final action is learned. It therefore studies decisions conditioned on an external fixed workflow rather than learning which interaction steps to perform.

Model Adaptation for Constrained Decisions. LoRA, QLoRA, and GRPO provide efficient adaptation and reward optimization Hu et al. (2022); Dettmers et al. (2023); Shao et al. (2024). NetLLM further uses networking encoders and task-specific output heads to produce valid task answers Wu et al. (2024). CoWA instead retains language-form action generation and conditions it on serialized tool observations. It combines tool-conditioned supervision with utility, format, and feasibility rewards; these rewards encourage, but do not guarantee, feasible outputs.

Wireless Allocation and Workflow Search BAI-MCTS repeatedly evaluates candidate allocations, while LLM-BAI-MCTS uses LLM guidance to reduce this per-instance exploration Lian et al. (2025). CoWA moves policy learning offline and generates an action after a fixed tool rollout, retaining tool cost but removing iterative utility search. WirelessAgent++ instead searches for workflow programs Tong et al. (2026). CoWA assumes a specified workflow and trains the compact policy that consumes its outputs, locating its contribution between fixed domain computation and learned final decisions.

7 Conclusion

This paper introduced CoWA, a framework that separated fixed domain computation from learned final-action generation for constrained system control. CoWA aligned a controller-compatible task interface, a fixed controller-executed tool workflow, tool-conditioned cold-start supervision, and GRPO-based objective alignment with utility, format-validity, and action-feasibility rewards. In the Wi-Fi 7 MLO case study, a CoWA-trained Llama-3.1-8B model learned from 750 trajectories and achieved 293.5 Mbps average throughput, compared with 291.8 Mbps for prompted GPT-5.2 and 297.2 Mbps for BAI-MCTS, while avoiding the latter's iterative per-instance search. The ablations verified the necessities of tool context, GRPO refinement, and validity rewards. CoWA also experienced a better generation and low end-to-end runtime compared with prompted GPT-5.2. These results showed that a compact policy could use controller-produced workflow observations to make effective constrained decisions without iterative per-instance utility search.

AI USE STATEMENT

In this work, we used generative AI tools for manuscript grammatical polishing, academic sentence refinement, and minor experiment code syntax assistance. We have not used generative AI tools for research idea design, core algorithm design, experimental design, or result reasoning. Generative AI models that serve as part of the proposed algorithm and experimental baselines are research artifacts described in the main paper and do not fall under this disclosure statement.

All AI-assisted text and auxiliary code has been fully reviewed, revised, and verified by the authors to ensure technical correctness. The authors take full and final responsibility for all content, claims, and artifacts presented in this paper.

REPRODUCIBILITY STATEMENT

The main paper specifies the system model, CoWA training objectives, Wi-Fi 7 MLO action space, controller-executed tool workflow, baselines, evaluation metrics, search budgets, and statistical reporting protocol. The appendix further documents instance and trajectory construction, the separation between hidden reference allocations and model-visible inputs, SFT and GRPO hyperparameters, reward weights, the task interface, tool-call serialization, output schema, and the prompt used for the GPT-5.2 baseline. Training uses 750 tool-augmented trajectories and a single NVIDIA H100 80GB GPU, and evaluation results are averaged over 10 random seeds. These details define the procedure used for the reported simulation study. Exact reproduction additionally requires the same PHY/MAC simulator implementation, software environment, model checkpoints, and service configuration; runtime results may vary across hardware and hosted-model conditions. The simulation codes are given in the supplementary material.

ETHICS STATEMENT

This study evaluates channel-allocation policies using generated Wi-Fi network instances and a PHY/MAC simulator. It does not recruit human participants, use human-subject data, or process personally identifying information. The reported results concern simulated network utility and should not be interpreted as evidence of safety or reliability in a live network. A deployment would process topology and operational telemetry that may be sensitive, and should therefore apply data minimization, access control, secure storage, and appropriate retention policies. Because optimizing aggregate throughput can produce unequal service across stations even when every station receives a valid link, practical systems should also monitor per-station quality, fairness, and failure cases. Finally, generated actions should be validated by the controller and tested in a sandbox or staged network before activation; the compact LLM should not be the sole authority for safety-critical or service-affecting control decisions.

REFERENCES

Marc Carrascosa-Zamacois, Giovanni Geraci, Lorenzo Galati-Giordano, Anders Jonsson, and Boris Bellalta. Understanding multi-link operation in wi-fi 7: Performance, anomalies, and solutions. In *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1–6. IEEE, 2023.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Cheng Chen, Xiaogang Chen, Dibakar Das, Dmitry Akhmetov, and Carlos Cordeiro. Overview and performance evaluation of wi-fi 7. *IEEE Communications Standards Magazine*, 6(2):12–18, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Shumin Lian, Jingwen Tong, Jun Zhang, and Liqun Fu. Intelligent channel allocation for ieee 802.11 be multi-link operation: When mab meets llm. *IEEE Journal on Selected Areas in Communications*, 2025.

Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. Mobile edge intelligence for large language models: A contemporary survey. *IEEE Communications Surveys & Tutorials*, 27(6):3820–3860, 2025.

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.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Jingwen Tong, Zijian Li, Fang Liu, Wei Guo, and Jun Zhang. Wirelessagent++: Automated agentic workflow design and benchmarking for wireless networks. *arXiv preprint arXiv:2603.00501*, 2026.

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.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. Netllm: Adapting large language models for networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 661–678, 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57 (8):1–35, 2025.

A Implementation of the CoWA Workflow

Appendix A expands the method-level interface and controller execution used in Sec. 3; Appendix B documents supervision construction and post-training; and Appendix C specifies the evaluation protocol and frontier-model prompt.

The implementation now specializes the three CoWA stages. First, task and interface alignment constructs

$$x_i = g(s_i; \Psi_{\mathrm{MLO}}, \mathcal{F}_{\mathrm{MLO}}, \Sigma_{\mathrm{MLO}}), \tag{23}$$

where Σ_{MLO} is the channel-to-STA output schema. Second, the controller uses the tool set

$$\mathcal{F}_{\mathrm{MLO}} = \{f_{\mathrm{dist}}, f_{\mathrm{cg}}, f_{\mathrm{int}}, f_{\mathrm{load}}, f_{\mathrm{assign}}, f_{\mathrm{unassigned}}\}, \tag{24}$$

where f_{dist} computes pairwise distances, f_{cg} builds channel-specific conflict graphs using R_c^{cs}, f_{int} summarizes conflict intensity, f_{load} plans channel loads, f_{assign} constructs initial groups, and $f_{\mathrm{unassigned}}$ identifies zero-link STAs. These operations are ordered as

$$f_{\mathrm{dist}} \to f_{\mathrm{cg}} \to f_{\mathrm{int}} \to f_{\mathrm{load}} \to f_{\mathrm{assign}} \to f_{\mathrm{unassigned}}. \tag{25}$$

The resulting history is $\tau_i = \mathrm{Rollout}(x_i; \mathcal{F}_{\mathrm{MLO}})$. It exposes distances, conflict summaries, planned loads, an initial assignment, and uncovered STAs to the policy. The model then generates the complete allocation from $\pi_\theta(\cdot \mid x_i, \tau_i)$; it neither chooses the tool order nor performs their calculations.

Third, objective alignment maps the generic reward to this task. The utility term $\bar{U}_{\mathrm{MLO}}$ is the normalized simulator throughput, V_{fmt} checks Σ_{MLO}, and V_{act} checks membership in $\mathcal{A}_{\mathrm{MLO}}(s_i)$. This mapping connects the method to the evaluation: throughput measures utility; omission measures violations of the coverage requirement; conflict counts diagnose contention in the generated allocation; the w/o-TA, w/o-GRPO, and w/o-RFA variants test the workflow context, objective refinement, and validity rewards; and runtime measures the end-to-end cost of the controller-plus-policy pipeline.

A.1 Task Interface, Tool Serialization, and Output Schema

For readability, this appendix retains the generic symbols Ψ, $\mathcal{F}$, and Σ. In the main-text instantiation, Ψ and Σ correspond to Ψ_{MLO} and Σ_{MLO}, while $\mathcal{F}_{\mathrm{MLO}}$ denotes the MLO-focused tool rollout subset of the implementation-level registry $\mathcal{F}$; the separate validation tool is described below. For each wireless state s_i, CoWA constructs a structured, tool-aware instruction instance

$$x_i = g(s_i; \Psi, \mathcal{F}, \Sigma), \tag{26}$$

where x_i contains the task instruction and serialized state, and τ_i contains the controller-executed tool history. The complete input is represented using a Llama-3-style chat template:

$$\mathrm{Serialize}(x_i, \tau_i) = \mathrm{ChatTemplate}\left(m_{\mathrm{sys}}, m_{\mathrm{user}}(s_i, \Psi), \mathcal{H}_{\mathrm{roll}}(s_i; \mathcal{F})\right). \tag{27}$$

The final assistant turn is left open, so the model generates the executable channel allocation conditioned on the serialized wireless state and the tool-derived observations.

State serialization. The network state is serialized in the user message as

$$\begin{aligned} m_{\mathrm{user}}(s_i, \Psi) = \{&\texttt{bss} : \mathcal{D}_i^{\mathrm{bss}},\ \texttt{channels} : \{\mathcal{C}_b\}_{b\in\mathcal{B}},\\ &\texttt{cs_radius} : \mathcal{R}_i^{\mathrm{cs}},\ \texttt{constraints} : \Psi,\ \texttt{task} : \mathcal{G}\}. \end{aligned} \tag{28}$$

Here, $\mathcal{D}_i^{\mathrm{bss}}$ contains the BSS-level topology, including the coordinates of APs and STAs. The `channels` field records the available channel–band mapping, represented consistently with the main text by $\{\mathcal{C}_b\}_{b\in\mathcal{B}}$. The formatter enumerates all STAs in the instance as `STA1`, `STA2`, ..., and also includes AP nodes when computing the distance matrix.

Constraint module. The task-constraint module is instantiated from the `constraints` field:

$$\Psi = \{\psi_{\mathrm{band}}, \psi_{\mathrm{mlo}}, \psi_{\mathrm{cover}}\}. \tag{29}$$

In the implementation, these correspond to `per_band_max_1_link`, `mlo_max_links_per_sta`, and `at_least_one_link`, respectively. They define the feasible action space for the generated allocation: each STA must obtain at least one valid link, the number of MLO links per STA is upper-bounded, and the allocation must respect the per-band link constraint.

Tool module. The domain tool set $\mathcal{F}$ is instantiated by the following implementation-level APIs:

$$\begin{aligned} f_{\mathrm{dist}} &\leftrightarrow \texttt{calc_distance_matrix},\\ f_{\mathrm{cg}} &\leftrightarrow \texttt{calc_conflict_graph},\\ f_{\mathrm{int}} &\leftrightarrow \texttt{calc_channel_conflict_intensity},\\ f_{\mathrm{load}} &\leftrightarrow \texttt{plan_channel_loads},\\ f_{\mathrm{assign}} &\leftrightarrow \texttt{find_and_assign_group},\\ f_{\mathrm{unassigned}} &\leftrightarrow \texttt{get_unassigned_stas},\\ f_{\mathrm{valid}} &\leftrightarrow \texttt{validate_assignment}. \end{aligned} \tag{30}$$

In the scripted, controller-executed prompt prefix, CoWA uses the first six tools to construct the rollout context and stops before the final decision turn. The validation tool is instead reflected in the feasibility reward and explicit post-generation checking.

Tool-rollout history. The scripted, controller-executed rollout is represented as a sequence of tool calls and tool observations:

$$\mathcal{H}_{\text{roll}} = \left[(f_t^{\text{tool}}, o_t^{\text{tool}})\right]_{t=1}^{T}, \tag{31}$$

where $f_t^{\text{tool}} \in \mathcal{F}$ denotes the tool invoked at step t, and o_t^{tool} denotes the corresponding returned observation. In the chat template, each tool call is serialized as

$$\text{Serialize}(f_t^{\text{tool}}) = \texttt{[TOOL_CALL]}\ \text{name}(f_t^{\text{tool}}) \quad \texttt{::}\ \text{JSONArgs}_t. \tag{32}$$

Here, $\text{name}(f_t^{\text{tool}})$ is the implementation-level API name of the invoked tool, and JSONArgs_t denotes the JSON-formatted arguments passed to that tool. The resulting observations provide distance summaries, channel-specific conflict references, conflict intensities, planned channel loads, partial channel assignments, and the remaining zero-link STAs. The compact LLM then synthesizes the final allocation from this tool-derived context.

Output schema. For instance i, the output schema Σ specifies a parseable channel-to-STA allocation:

$$\Sigma: \quad \hat{a}_i = \{(c, \mathcal{N}_{i,c}) \mid b \in \mathcal{B},\ c \in \mathcal{C}_b,\ \mathcal{N}_{i,c} \subseteq \mathcal{N}_i\}, \tag{33}$$

where $\mathcal{N}_i$ is the STA set in instance i and $\mathcal{N}_{i,c}$ is the set of STAs assigned to channel c. In the actual response format, the final decision is written under the marker `### Final allocation`, followed by one channel line per channel. Since MLO permits multi-link operation, the same STA may appear in multiple channel-specific sets, provided that the allocation satisfies Ψ.

The overall workflow is summarized below.

Workflow abstraction for CoWA channel allocation

System message. WiFi 7 MLO-STR channel allocation assistant with a strict multi-round tool-calling pattern.

User message.

$$\{\texttt{bss}: \mathcal{B}_i,\ \texttt{channels}: \mathcal{C}_i, \texttt{cs_radius}: \mathcal{R}_i^{\text{cs}},\ \texttt{constraints}: \Psi, \texttt{task}: \mathcal{G}\}.$$

Scripted tool prefix.

$$\begin{aligned}
&\texttt{[TOOL_CALL] calc_distance_matrix} \rightarrow o_{\text{dist}},\\
&\texttt{[TOOL_CALL] calc_conflict_graph} \rightarrow o_{\text{cg}},\\
&\texttt{[TOOL_CALL] calc_channel_conflict_intensity} \rightarrow o_{\text{int}},\\
&\texttt{[TOOL_CALL] plan_channel_loads} \rightarrow o_{\text{load}},\\
&\texttt{[TOOL_CALL] find_and_assign_group} \rightarrow o_{\text{assign}},\\
&\texttt{[TOOL_CALL] get_unassigned_stas} \rightarrow o_{\text{unassigned}}.
\end{aligned}$$

Final assistant response. Generate `### Final allocation` in the schema Σ, i.e., a channel-to-STA allocation $\hat{a}$.

B Dataset Construction and Training Details

B.1 Dataset and Trajectory Construction

We construct the cold-start dataset from executable Wi-Fi 7 MLO channel-allocation instances and a controller-executed tool workflow. The construction procedure is designed to provide the model with structured wireless-state information and deterministic tool-derived observations while keeping the reference allocation hidden from the serialized training conversation.

Instance generation. Each instance specifies a dense Wi-Fi 7 MLO uplink scenario with multiple BSSs. For each BSS, the instance contains the AP position and the positions of its associated STAs. The implementation uses three BSSs and three available channels. The channel configuration is `Channel 1` = 2.4 GHz, `Channel 2` = 5 GHz, and `Channel 3` = 6 GHz.

Each channel has a channel-specific carrier-sensing radius. STA identifiers are assigned globally as `STA1`, `STA2`, and so on. The resulting network state is serialized into a JSON-formatted user message containing the BSS topology, channel–band mapping, carrier-sensing radii, allocation constraints, and task description.

Reference allocation and hidden supervision. For each instance, a reference allocation is obtained from the corresponding `arm_pool` representation. The implementation maps the arm indices

to channel-specific STA sets using fixed channel maps. This reference allocation is *not* included in the system message, user message, tool-call arguments, or tool observations. In particular, no `target_preferred` field is exposed to the model. The reference allocation is retained by the offline data-generation controller for controlled completion and rejection filtering only.

Controller-executed tool workflow. For every instance, the controller executes a fixed sequence of domain tools before constructing the final allocation:

$$f_{\text{dist}} \rightarrow f_{\text{cg}} \rightarrow f_{\text{int}} \rightarrow f_{\text{load}} \rightarrow f_{\text{assign}} \rightarrow f_{\text{unassigned}}.$$

The distance tool computes the pairwise Euclidean distance matrix over STA and AP nodes. The conflict-graph tool constructs a separate conflict graph for each channel by connecting two STAs when their Euclidean distance is no greater than the channel-specific carrier-sensing radius. The conflict-intensity tool counts the intra-channel conflict edges and normalizes the resulting channel intensities. The load-planning tool assigns a planned number of STAs to each channel, assigning lower load to channels with higher normalized conflict intensity. The group-assignment tool then selects and assigns an initial STA group to each channel. Finally, the unassigned-STA tool identifies STAs that have not yet received any link.

Initial group selection. The initial channel groups are selected without passing the reference allocation to the group-selection tool. The implementation first attempts to construct a large conflict-free group. If a sufficiently large independent set is found, it is selected directly. Otherwise, the controller constructs a group using a weighted heuristic that favors low node conflict, low intra-channel conflict, larger pairwise spatial separation, and a mild preference for STAs that have not yet received a link. The resulting tool observation records the selected STAs, conflict statistics, average pairwise distance, selection mode, and the current channel assignment.

Controlled completion. After the initial tool workflow, the offline data-generation controller checks for zero-link STAs. The completion procedure has two purposes. First, it ensures that every STA obtains at least one link. When possible, an unassigned STA is placed on a channel contained in the hidden reference allocation; otherwise, the controller selects the channel producing the fewest newly introduced conflict edges. Second, the controller performs controlled channel-level supplementation so that the final number of STAs per channel remains consistent with the reference channel scale. When a channel contains too many users, non-reference users are removed first, prioritizing those with higher conflict. When a channel contains too few users, users from the reference set are added according to their incremental conflict cost. These operations construct offline supervision targets only. They are absent from the model-visible tool history and are not executed with hidden references at inference time.

Serialization. The resulting trajectory is serialized as a sequence of `system`, `user`, `assistant`, and `tool` messages. Tool calls use the implementation-level API names and JSON-formatted arguments. Distance matrices and conflict graphs are stored internally through reference identifiers, while the serialized observations expose compact summaries and the information required by subsequent workflow steps. The final assistant message contains the executable channel-to-STA allocation under the marker `### Final allocation`. The same STA may appear on multiple channels because MLO permits multi-link operation.

Offline rejection filtering. The generated allocation is evaluated before an instance is retained. First, for each channel, the controller computes the fraction of predicted users that are not contained in the reference allocation. An instance is rejected if this extra-user ratio exceeds $1/3$ for any channel. Second, the generated allocation is converted back to the corresponding arm representation and evaluated with the executable throughput simulator using 15 simulation runs. Let U_{pred} denote the generated allocation throughput, U_{ref} the reference throughput, and U_{rand} the random-allocation throughput. When $U_{\text{ref}} > U_{\text{rand}}$, an instance is retained only if

$$U_{\text{pred}} - U_{\text{rand}} \geq 0.75\,(U_{\text{ref}} - U_{\text{rand}})\,.$$

These rejection criteria instantiate the generic normalized-gain test in Sec. 3.3 with $U_i^{\text{base}} = U_{\text{rand}}$ and $\eta = 0.75$, together with the extra-user constraint. They are applied offline and are not included in the model-visible prompt.

Training examples. The retained trajectories form the cold-start dataset

$$\mathcal{D}_{\text{cs}} = \{(x_i, \tau_i, a_i^\star)\}_{i=1}^{N},$$

where x_i is the structured task instance, τ_i is the serialized controller-executed tool-workflow prefix, and $a_i^\star$ is the final executable allocation. During cold-start SFT, the model is trained to synthesize the final allocation conditioned on the task instance and the serialized tool observations. The intermediate tool execution itself is performed by the controller rather than learned as an autonomous tool-selection policy.

B.2 Training and Reward Configuration

We use Llama-3.1-8B as the backbone model and adopt a two-stage training pipeline. The first stage performs cold-start supervised fine-tuning (SFT) with QLoRA, while the second stage further optimizes the SFT checkpoint using GRPO with LoRA. The key hyperparameters used in both stages are summarized in Table 3; below, we describe the main training choices in more detail.

In the SFT stage, the model is loaded with 4-bit quantization and trained with LoRA adapters. The maximum sequence length is set to 8192. We train the model for 6 epochs with an initial learning rate of 2×10^{-4}. The SFT stage uses an 8-bit AdamW optimizer, cosine learning-rate scheduling, a warmup ratio of 0.05, and weight decay 0.02. The per-device batch size is 4 and the gradient accumulation step is 8.

In the GRPO stage, the maximum sequence length remains 8192, with a maximum prompt length of 7168 and a maximum completion length of 1024. We train for 2 epochs with an initial learning rate of 5×10^{-6}, using the `paged_adamw_8bit` optimizer, cosine scheduling, a warmup ratio of 0.1, and weight decay 0.1. The per-device batch size is 2, the gradient accumulation step is 2, and each prompt is sampled with 8 generations. Within an update, the sampled batch is associated with the old policy used in the importance ratio, while the fixed cold-start SFT checkpoint is used as the KL reference, matching Eq. (8) and the GRPO objective in Sec. 3.4.

For GRPO, the reward follows Eq. (8) and combines three components: throughput performance, output-format validity, and protocol feasibility. The format-validity term checks whether the generated response satisfies the required schema, while the protocol-feasibility term checks whether the generated allocation belongs to the feasible action set under the current state. For all experiments on MLO channel allocation, the reward weights are set to $\omega_1 = 5$, $\omega_2 = 0.5$, and $\omega_3 = 1$. All training experiments are conducted on a single NVIDIA H100 80GB HBM3 GPU.

C Evaluation Protocol and Frontier-LLM Baseline

C.1 Evaluation Protocol

All methods are evaluated with the same executable PHY/MAC simulator, which maps a topology and channel allocation to aggregate throughput while accounting for link quality, contention, and collisions. The main comparison uses 6, 12, 18, and 24 STAs. The cross-scale study evaluates 12, 15, 18, 21, 24, 27, and 30 STAs: 12, 18, and 24 are represented in training; 15 and 21 are unseen interpolation scales; and 27 and 30 are unseen extrapolation scales above the largest training setting. Reported values are means over 10 random seeds, and the main throughput figure additionally reports 95% confidence intervals.

For BAI-MCTS, the per-instance interaction budgets are 1000, 3000, 4000, and 5000 at 6, 12, 18, and 24 STAs, respectively. CoWA instead executes the fixed six-tool prefix in Appendix A and generates one final allocation. The comparison therefore measures system-level outcome under the stated procedures rather than equal model inputs or equal computation.

Table 3: Training hyperparameters used in CoWA.

Hyperparameter	Value
Training dataset	750 tool-augmented trajectories (300/150/150/150 for 6/12/18/24 STAs)
Backbone model	Llama-3.1-8B
Cold-start method	QLoRA-based SFT
Post-training method	GRPO with LoRA
Maximum sequence length	8192
SFT epochs	6
SFT learning rate	2×10^{-4}
SFT optimizer	8-bit AdamW
SFT batch size	4
SFT gradient accumulation steps	8
SFT warmup ratio	0.05
SFT weight decay	0.02
GRPO epochs	2
GRPO learning rate	5×10^{-6}
GRPO LoRA rank	32
GRPO optimizer	paged AdamW 8-bit
GRPO batch size	2
GRPO gradient accumulation steps	2
GRPO number of generations	8
GRPO maximum prompt length	7168
GRPO maximum completion length	1024
GRPO warmup ratio	0.1
GRPO weight decay	0.1
GRPO maximum gradient norm	0.1
GPU	NVIDIA H100 80GB HBM3
Reward weight ω_1	5
Reward weight ω_2	0.5
Reward weight ω_3	1

Throughput is the aggregate simulator output in Mbps. Omission is the number of STAs with no allocated link, averaged across test instances. Conflict is the number of same-channel STA pairs whose distance does not exceed that channel's carrier-sensing radius, also averaged across instances. Runtime is end-to-end wall-clock time from task arrival to the final allocation. The runtime comparison includes each method's complete execution path and is not hardware-normalized; it should therefore be interpreted only for the reported setup.

C.2 Prompt Template for the Frontier-LLM Baseline

We evaluate GPT-5.2 as a prompt-only baseline. The baseline receives the same wireless task definition, network state, channel configuration, carrier-sensing radii, and allocation constraints as CoWA, but it does not receive CoWA's controller-executed tool-rollout history. The prompt below is instantiated separately for each evaluation instance by replacing the placeholders with the corresponding network information.

> You are a wireless network engineer. Your task is to allocate the available wireless channels to stations (STAs) in a network so that the total throughput of network is maximized. Every STA must be assigned at least one channel.
>
> The network follows the IEEE 802.11be Wi-Fi 7 protocol and supports simultaneous transmit-and-receive multi-link operation (MLO-STR). The same STA may be assigned to multiple channels because links on different bands can operate independently.
>
> You should analyze the network step by step and then output the final channel allocation.
>
> **Network information.**

The network contains <NUMBER_OF_BSS> basic service sets (BSSs). Each BSS contains one access point (AP) and <NUMBER_OF_STAS_PER_BSS> STAs. The STA indices are assigned globally as STA1, STA2, ..., STA<N>.

BSS information:

- <BSS1: AP coordinate and STA coordinates>
- <BSS2: AP coordinate and STA coordinates>
- <BSS3: AP coordinate and STA coordinates>
- <Additional BSSs if any>

Available channels:

- <Channel 1: band, e.g., 2.4 GHz>
- <Channel 2: band, e.g., 5 GHz>
- <Channel 3: band, e.g., 6 GHz>
- <Additional channels if any>

Carrier-sensing radii:

- <Carrier-sensing radius of Channel 1>
- <Carrier-sensing radius of Channel 2>
- <Carrier-sensing radius of Channel 3>
- <Additional carrier-sensing radii if any>

Constraints:

- Each STA must be assigned at least one channel.
- A STA may be assigned to multiple channels under MLO, with at most one channel assigned per frequency band.
- For each channel, STAs within the carrier-sensing radius may contend with each other.
- The final answer must be an executable channel allocation.

Reasoning steps.

Please strictly follow these steps:

1. Identify the basic network information, including the number of channels, the number of STAs, and the available channel bands.

2. Clarify the objective. Since MLO-STR allows independent links across different bands, the allocation should maximize the total network throughput while ensuring that every STA has at least one assigned channel.

3. Analyze conflict relationships. For each channel, two STAs are considered conflicting if their Euclidean distance is no greater than the carrier-sensing radius of that channel. STAs assigned to the same channel may contend under carrier sense multiple access (CSMA), so the allocation should reduce severe intra-channel conflicts when possible.

4. Perform initial allocation. For each channel, first select STAs or STA groups with low conflict and large spatial separation.

5. Perform further allocation. To better exploit MLO, assign additional channels to STAs when this can improve channel utilization without introducing excessive contention. Ensure that no STA is left without a channel.

Output format.

First, provide a concise step-by-step analysis as above. Then provide the final allocation in exactly the following format:

Final allocation: Channel 1: <STA list assigned to Channel 1>
Channel 2: <STA list assigned to Channel 2>
Channel 3: <STA list assigned to Channel 3>...

Now solve the following instance:

<Insert the target network instance here, including BSS coordinates, channel bands, carrier-sensing radii, and constraints.>