REVISITING WHITE-BOX COMMUNICATION IN MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Communication is essential for coordinating the behaviors of multiple agents. However, existing black-box communication methods often rely on complex optimization mechanisms and produce communication behaviors that are difficult to interpret. Recent advances in large language models (LLMs) substantially reduce the effort required to construct white-box communication protocols, raising a natural question: can white-box communication become a competitive alternative to learned communication? In this work, we systematically revisit white-box communication by studying its effectiveness, sources of effectiveness, capability boundaries, and potential value for learned communication. We first show that LLM-generated white-box protocols achieve performance competitive with learned communication methods across decisions about what, when, and whom to communicate. Further analysis identifies advances in foundation-model capabilities as a key source of this effectiveness. We then characterize the capability boundaries of white-box communication. While scaling the number of agents alone does not necessarily degrade performance, increasing agent heterogeneity and protocol complexity poses greater challenges; performance also declines as coordination-relevant semantics become less explicit, and feedback-driven self-refinement remains unstable. These findings reveal a natural complementarity between white-box and learned communication: white-box protocols provide explicit communication knowledge but are difficult to refine reliably, whereas learned policies can continue improving through RL. We therefore use white-box protocols as teachers for neural communication policies, improving learning efficiency while retaining the capacity for further optimization. Taken together, our study demonstrates that white-box communication deserves renewed attention as a viable direction for multi-agent communication research.

1 INTRODUCTION

Reinforcement learning (RL) has achieved remarkable success in complex decision-making tasks, including game AI (Osband et al., 2016; Silver et al., 2017; 2018; Vinyals et al., 2019) and robotics (Andrychowicz et al., 2020). However, extending these advances to multi-agent reinforcement learning (MARL) (Lowe et al., 2017; Rashid et al., 2018; Yu et al., 2022; Chen et al., 2022) raises a fundamental challenge: coordinating multiple agents under partial observability. Because each agent has access only to its local observations, it lacks a comprehensive view of the environment. Multi-agent communication helps bridge this information gap by enabling agents to exchange complementary information and refine their local beliefs, thereby supporting a more complete collective understanding of the environment and more effective coordination (Sukhbaatar et al., 2016; Zhang et al., 2019; 2020; Yuan et al., 2022).

Early approaches to multi-agent communication relied on hand-designed protocols that explicitly specified what information agents exchanged and how received messages were used (Stone & Veloso, 1999). Although these protocols made communication behavior interpretable and controllable, designing and revising their rules in response to task feedback required substantial manual effort, as illustrated in Figure 1(a). To reduce this burden, subsequent work introduced neural communication policies that learn what to share (Zhang et al., 2020; Yuan et al., 2022; Chen et al., 2024), when to communicate (Singh et al., 2018; Kim et al., 2019; Hu et al., 2021; Li et al., 2025), and with whom (Ding et al., 2020; Niu et al., 2021; Xue et al., 2022; Sun et al., 2024; Duan et al., 2024).

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.

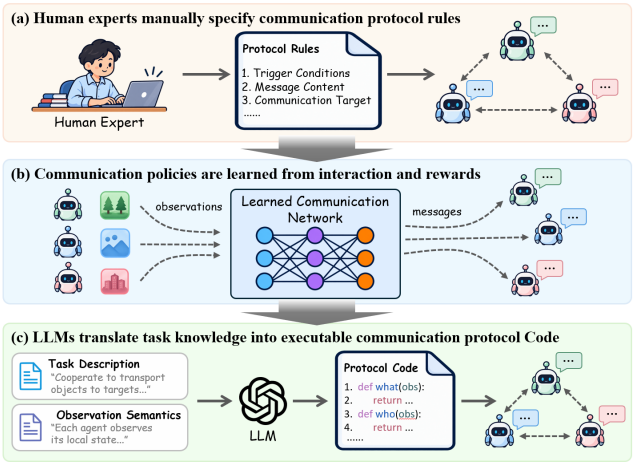

Figure 1: Evolution of Communication Paradigms in MARL.

As shown in Figure 1(b), this shifts communication design from handcrafted rules to policies optimized through interaction with the environment, allowing communication behavior to adapt to task feedback. However, environmental rewards alone may provide only a sparse and indirect signal for optimizing communication policies, while designing communication optimization mechanisms that generalize across diverse tasks remains challenging. Furthermore, learning an additional neural communication policy incurs training overhead, while its black-box nature makes it difficult to explain why particular messages are sent or recipients are selected.

In the era of large language models (LLMs), advances in code generation and iterative refinement offer a new opportunity to revisit white-box communication. LLMs can translate natural-language specifications into executable code (Liang et al., 2022; Singh et al., 2022) and revise communication rules in response to evaluation feedback, as illustrated in Figure 1(c). By automating much of protocol construction and maintenance, this capability can substantially reduce the human effort traditionally required by rule-based approaches. With this practical barrier lowered, we ask: how far can white-box communication go in the LLM era?

In this work, we systematically revisit white-box communication in MARL, asking whether recent advances in LLMs can turn explicit communication protocols from manually engineered heuristics into a competitive and reusable communication paradigm. First, we show that LLMs can automatically generate interpretable and controllable communication protocols at a cost of approximately 0.20–0.30$ per generation. When incorporated throughout policy training, these protocols achieve performance competitive with state-of-the-art learned communication methods across decisions about what to share, when to communicate, and with whom. Second, we show that this capability is closely tied to advances in the underlying LLMs: stronger foundation models consistently produce more effective communication protocols. Third, we characterize the capability boundaries of white-box communication. Scaling the number of agents alone does not necessarily degrade its effectiveness, whereas increasing agent heterogeneity and protocol complexity poses greater challenges. Its performance also declines when coordination-relevant semantics are less explicitly exposed, and feedback-driven self-refinement remains unstable. Finally, we show that the explicit knowledge encoded in white-box protocols can further serve as supervision for learned communication policies, improving learning efficiency while allowing continued optimization through RL.

Taken together, we demonstrate that white-box communication deserves renewed attention as a viable direction for multi-agent communication research. Looking forward, leveraging emerging agentic techniques, such as memory and self-refinement, to construct efficient, stable, and interpretable communication policies may represent a promising direction for future research.

2 Related Work

2.1 Handcrafted Communication Protocols

Early multi-agent communication relied on handcrafted protocols that explicitly specified what information to exchange and how agents should interact. Representative examples include the Contract Net Protocol for task allocation (Smith, 1980) and the locker-room agreement for coordinated robotic soccer (Stone & Veloso, 1999). Similar ideas were later adopted in multi-agent learning, where agents exchanged sensory information, reinforcement signals, policies, or structured advice

according to predefined rules (Tan, 1993; Matarić, 1997; Nunes & Oliveira, 2003). These approaches demonstrate that task knowledge can be directly encoded into communication mechanisms, offering explicit and interpretable interaction structures. However, designing and adapting such protocols typically relies heavily on human expertise, which limits their scalability and flexibility across tasks.

2.2 Learning-Based Multi-Agent Communication

Learning-based multi-agent communication replaces handcrafted protocols with neural communication policies optimized through interaction with the environment. Existing methods mainly study three aspects of communication: what information to transmit, when and with whom to communicate, and how received information should be utilized.

Determining the optimal content of communication (what to communicate). CommNet Sukhbaatar et al. (2016), as a pioneering work in this area, facilitated agents in learning continuous messages. Following CommNet, several methods have been developed to further refine the message learning process. VBC Zhang et al. (2019) aims to filter out noisy parts while retaining valuable content by limiting the variance of messages. TMC Zhang et al. (2020) introduces regularizers to reduce temporally redundant messages. NDQ Wang et al. (2020) employs information-theoretic regularizers to develop expressive and succinct messages. MAIC Yuan et al. (2022) enabled agents to customize communications for specific recipients, advancing tailored message learning.

Deciding appropriate timing and partners for information exchange (when and whom to communicate). To enhance communication efficiency, approaches such as IC3Net Singh et al. (2018) and ATOC Kim et al. (2019) have introduced gating networks to eliminate superfluous communication links. Similarly, SchedNet Kim et al. (2019), IMMAC Sun et al. (2021) and T2MACSun et al. (2024) have modeled the significance of observations, using heuristic mechanisms to gate non-essential communication. Further, methods such as MAGIC Niu et al. (2021), ToM2CWang et al. (2022), I2C Ding et al. (2020) and SMS Xue et al. (2022) have been developed to identify the most suitable recipients. These approaches focus on modeling the contribution of shared information to the decision-making processes of the recipients, aiming to direct communication where it most influences decision-making.

Integrating incoming messages and making decisions (how to utilize received information). Another line of work focuses on how agents utilize received messages to improve their local understanding of the environment and support decision-making. TarMAC Das et al. (2019) studies how agents can attend to and assimilate task-relevant information from abundant raw messages. MASIA Guan et al. (2022) takes a different perspective by employing an auto-encoder and a forward model to integrate communicated information, making an early attempt to introduce self-supervised learning into multi-agent communication. Following this direction, M2I2Sun et al. (2026) further proposes two self-supervised objectives, i.e., masked state modeling and joint-action prediction, to enrich agents' perception of environmental uncertainty and facilitate the anticipation of teammates' intentions.

Although learned communication provides greater adaptability than handcrafted protocols, achieving effective communication often requires carefully designed gating mechanisms, auxiliary objectives, information-theoretic regularizers, or self-supervised losses. Moreover, because communication decisions are typically represented by neural networks, the resulting protocols remain largely black-box, making it difficult to understand why particular information is transmitted, when communication occurs, or why specific recipients are selected.

2.3 LLM-Assisted Multi-Agent Communication

Recent work has explored LLMs as sources of semantic knowledge and reasoning priors for RL, improving policy learning, exploration, and decision making Du et al. (2023); Ma et al. (2024); Xie et al. (2024); Dalal et al. (2024). This idea has also been extended to multi-agent communication. For example, LangGround (Li et al., 2024) uses LLM-generated interaction data to ground learned communication representations in natural language, while the communication policy itself remains a neural network. More recently, LMAC (Bae et al., 2026) uses LLMs to construct an explicit communication-content selection strategy for efficient global-state reconstruction. LMAC focuses on how LLMs can help design a particular interpretable communication mechanism for a specific

objective. In contrast, to the best of our knowledge, we are the first to systematically study LLM-generated white-box communication itself. We evaluate its effectiveness across what, when, and whom to communicate, investigate the sources of this capability, characterize its boundaries under different task conditions, and examine how the resulting white-box knowledge can further guide learned communication policies.

3 PRELIMINARY

3.1 DECENTRALIZED PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

We consider fully cooperative multi-agent tasks modeled as a Dec-POMDP augmented with communication (Oliehoek et al., 2016), $G = (N, S, O, A, \mathbb{O}, \mathcal{P}, R, \gamma, M)$. Here, $N = \{1, \ldots, n\}$ is the agent set; S, O, A, and M denote the state, observation, action, and message spaces, respectively; $\mathbb{O}$ is the observation function; $\mathcal{P}$ is the transition function; R is the shared reward function; and $\gamma \in [0, 1)$ is the discount factor. At time t, agent i receives a local observation o_i^t and communicated information c_i^t, and selects an action according to its policy π_{θ_i}. The joint action $\mathbf{a}_t$ yields a shared reward $r_t = R(s_t, \mathbf{a}_t)$ and a transition $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, \mathbf{a}_t)$. The objective is to maximize the expected return $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$, where π denotes the joint policy.

3.2 WHITE-BOX PROTOCOL CONSTRUCTION

To efficiently construct white-box communication protocols without relying on extensive manual rule engineering, we leverage the semantic reasoning and code-generation capabilities of LLMs to translate task knowledge into explicit and executable communication rules. Specifically, we provide the LLM with a prompt $P = (p_{\text{task}}, p_{\text{obs}})$, where p_{task} describes the task and its coordination requirements, and p_{obs} specifies the semantics of the observation dimensions available to the agents. Based on this information, the LLM first identifies task-relevant communication requirements and expresses them as explicit rules in natural language, and then translates these rules into an executable Python function:

$$P = (p_{\text{task}}, p_{\text{obs}}), \qquad f_P = \text{LLM}(P). \tag{1}$$

The resulting function f_P constitutes the white-box communication protocol: it directly maps the current observations to explicit communication decisions, without introducing additional learnable parameters. Depending on the communication component under study, these decisions specify what information to transmit, when communication should occur, or which agents should communicate. Detailed prompts and implementation procedures are provided in Appendix A.

4 HOW EFFECTIVE IS WHITE-BOX COMMUNICATION?

We begin our study by comprehensively evaluating the communication capabilities of LLM-generated white-box communication protocols. Specifically, we consider three fundamental dimensions of multi-agent communication—what to communicate, when to communicate, and with whom to communicate—and select IMMAC (Sun et al., 2021), SMS (Xue et al., 2022), and M2I2 (Sun et al., 2026) as representative state-of-the-art neural baselines for these three dimensions, respectively. For each baseline, we integrate the corresponding white-box protocol under two schemes:

- **WB-Direct.** We first jointly train the original black-box communication module and the MARL policy for two million environment steps. At evaluation time, we replace the trained communication module with the white-box protocol while keeping all other learned components unchanged.
- **WB-Train.** We replace the black-box communication module with the white-box protocol before training and use the protocol to make communication decisions throughout the training process. The white-box protocol remains fixed, while only the MARL policy is optimized.

We evaluate both schemes on three communication-focused SMAC tasks: `1o_2r_vs_4r`, `1o_10b_vs_1r`, and `5z_vs_1ul`. As shown in Figure 2, directly replacing a trained black-box communication policy with a white-box protocol (**WB-Direct**) leads to a substantial performance

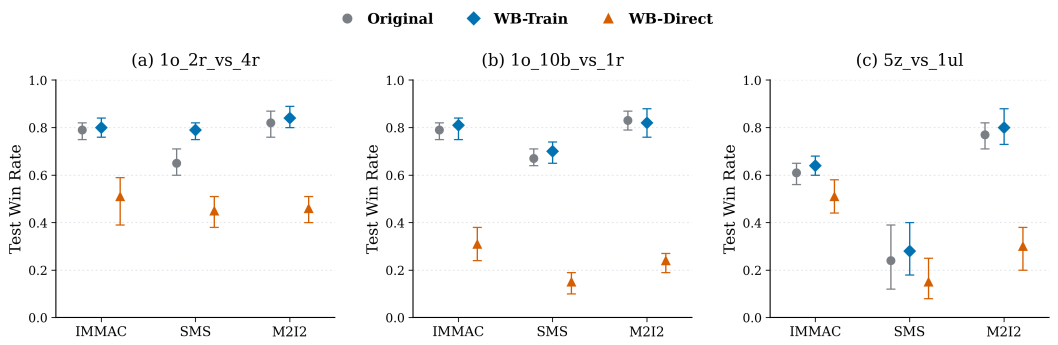

Figure 2: When should white-box communication be introduced? White-box protocols achieve performance comparable to learned communication when used throughout policy training (WB-Train), but direct post-training replacement (WB-Direct) causes substantial degradation. Results are averaged over five random seeds, with error bars showing the minimum and maximum.

drop. Notably, the white-box protocol exhibits a communication rate comparable to that of the trained black-box policy, suggesting that this degradation cannot be simply attributed to a change in communication frequency. The communication policy determines what information is made available to the learned action policy. Replacing it only after training therefore alters the effective information structure and creates a mismatch between the inputs encountered during training and evaluation. Consequently, direct post-training replacement is not a viable way to deploy white-box communication protocols.

In contrast, when the white-box communication protocol is used throughout MARL policy training, **WB-Train** achieves performance comparable to or slightly better than that of its learned black-box counterpart. These results demonstrate that the communication capability of the white-box protocol is not inferior to that of learned black-box communication models. With LLMs substantially reducing the manual effort required to construct and refine explicit protocols, white-box communication therefore emerges as a viable alternative to learned neural communication policies: it avoids the need to design and optimize a separate communication learning mechanism while keeping communication decisions explicit and interpretable.

5 What Makes LLM-Generated White-Box Communication Effective?

In this section, we investigate what makes LLM-generated white-box communication effective. We first compare LLM-generated protocols with manually designed ones to assess the role of protocol quality, and then examine how foundation-model capability affects communication performance. Our results suggest that advances in foundation-model capability are an important factor associated with more effective white-box communication.

5.1 Does LLM Generation Improve White-Box Protocol Design?

We first compare LLM-generated and manually designed protocols for what, when, and whom to communicate, replacing the corresponding modules in IMMAC, SMS, and M2I2 under the **WB-Train** setting. As shown in Figure 3, LLM-generated protocols achieve higher mean win rates with lower variance. This suggests that effective white-box communication depends critically on protocol quality. Compared with manual design, LLMs can leverage pretrained knowledge and reasoning to better identify coordination-relevant information and translate it into explicit communication rules. This naturally raises the question of how the capability of the underlying foundation model affects protocol quality.

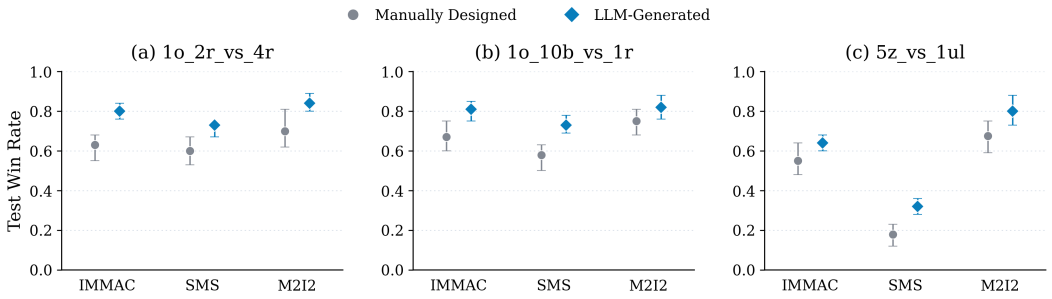

Figure 3: Comparison between manually designed and LLM-generated white-box communication protocols.

5.2 How Does LLM Capability Affect Protocol Effectiveness?

We next investigate how the capability of the underlying LLM affects the quality of the generated communication protocol. To provide a clear view of how advances in model capability relate to communication performance, we compare protocols generated by successive GPT models, including GPT-3.5 Turbo, GPT-4o, GPT-4.1, GPT-5, and GPT-5.6. We conduct this study with M2I2 on `1o_10b_vs_1r`, where the communication module determines what information should be communicated. For each LLM, we replace the corresponding communication module in M2I2 with the generated white-box protocol and evaluate it under the **WB-Train** setting.

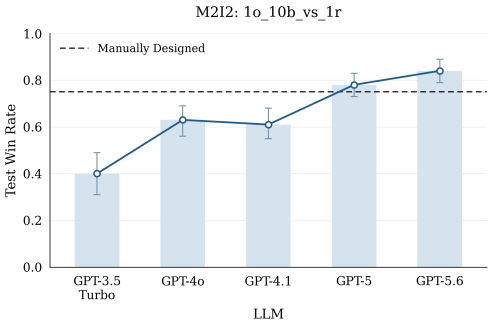

Figure 4: Effect of LLM capability on white-box protocol effectiveness.

As shown in Figure 4, compared with the manually designed protocol, indicated by the dashed line, protocols generated by GPT-3.5 Turbo, GPT-4o, and GPT-4.1 achieve lower mean performance, whereas those generated by GPT-5 and GPT-5.6 surpass the manual baseline. The overall trend suggests that the effectiveness of white-box communication improves as the underlying LLM becomes more capable. Stronger models are better able to interpret task semantics, identify coordination-relevant information, and translate these requirements into coherent communication rules. Together with the comparison against manually designed protocols, these results indicate that advances in foundation-model capabilities are a key factor enabling effective LLM-generated white-box communication.

6 Exploring the Capability Boundaries of White-Box Communication

In this section, we investigate the capability boundaries of white-box communication. Specifically, we examine its performance under two challenging conditions: tasks that require increasingly complex communication rules, and settings where coordination-relevant semantics are less explicitly exposed. Furthermore, we investigate whether white-box communication protocols can effectively leverage environmental feedback to evolve together with the MARL policy in a stable manner.

6.1 How Do Team Size and Heterogeneity Affect White-Box Communication?

We first investigate how white-box communication scales with team size and agent heterogeneity. We compare GACG (Duan et al., 2024) with its white-box variant under the **WB-Train** setting

on SMACv2 Terran, as GACG explicitly learns inter-agent communication relationships and thus provides a natural learned counterpart. We consider two settings: increasing the number of agents while keeping agent types unchanged, and increasing the number of agent types at a fixed team size. The former mainly increases team scale, whereas the latter increases the diversity of communication requirements and thus the complexity of the white-box protocol.

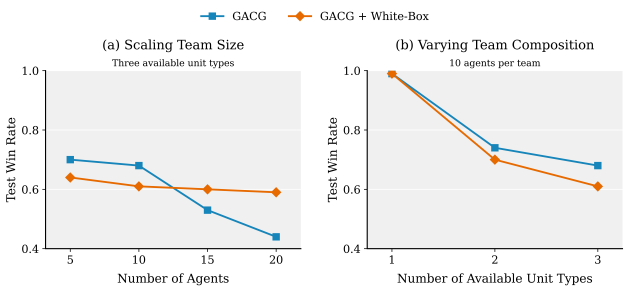

Figure 5: Effects of team size and agent heterogeneity on white-box communication.

Scaling Team Size. Figure 5(a) shows that GACG performs better with 5 and 10 agents, but degrades substantially as the team size increases to 15 and 20, where the white-box variant achieves higher win rates. This is because white-box rules are defined over agent types and can be reused as team size grows, whereas GACG must learn communication relationships over an increasingly large set of agents.

Varying Team Composition. We next fix the team size at 10 and increase the number of agent types. As shown in Figure 5(b), both methods perform well with a single agent type, while increasing heterogeneity leads to lower performance and a larger gap between the white-box variant and GACG. Unlike scaling the number of existing agent types, introducing new types requires additional communication rules for different roles and interactions, increasing protocol complexity. These results suggest that white-box communication is more sensitive to the complexity of communication requirements than to team size itself. As the complexity of communication rules increase, constructing a high-quality white-box protocol becomes increasingly challenging.

6.2 How Does White-Box Communication Perform When Coordination Semantics Are Less Explicit?

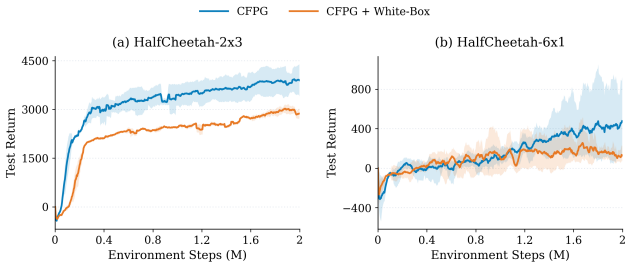

Figure 6: Performance of white-box communication in tasks with less explicit coordination semantics.

We next examine white-box communication in settings where coordination-relevant semantics are less explicit. We compare CFPG (Zhu et al., 2025) with its white-box variant under the **WB-Train** setting on HalfCheetah-2x3 and HalfCheetah-6x1 from Multi-Agent MuJoCo (Peng et al., 2021). Unlike SMAC, observations in Multi-Agent MuJoCo mainly consist of low-level physical quantities such as joint positions, angles, and velocities, whose relevance to inter-agent coordination is less directly expressed. As shown in Figure 6, the white-box variant achieves lower final returns than CFPG on both tasks, with a larger gap on HalfCheetah-6x1. These results suggest that constructing effective white-box protocols becomes more challenging when coordination-relevant semantics are less explicitly exposed in the observation space, highlighting an important limitation of current white-box communication.

6.3 Can White-Box Protocols Evolve Reliably?

We finally investigate whether white-box communication protocols can reliably evolve with MARL policy learning. Specifically, we collect interaction trajectories during train-

ing and provide them as environmental feedback to the LLM, which analyzes potential weaknesses of the current protocol and iteratively rewrites its communication rules. The refined protocol is then used in subsequent policy training, forming a feedback-driven evolution loop between the communication protocol and the MARL policy. As shown in Figure 7, iterative protocol refinement does not consistently improve upon the fixed white-box protocol and can even degrade performance. This suggests that, although current LLMs can construct effective protocols from task descriptions, reliably improving them from environmental feedback remains challenging. This observation highlights an important direction for future white-box communication research. Developing reliable protocol evolution mechanisms, potentially by incorporating emerging agentic techniques such as memoryPark et al. (2023); Shinn et al. (2023); Xu et al. (2026) and recursive self-improvementYin et al. (2025); Zhang et al. (2026), may enable white-box communication to continuously adapt while preserving its explicit and interpretable structure.

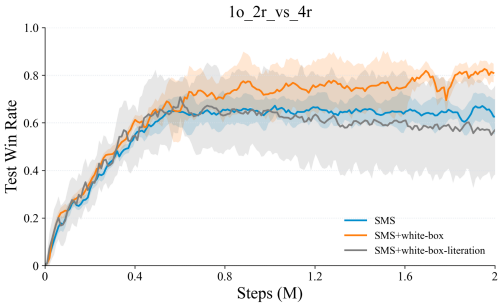

Figure 7: Learning curves of SMS, SMS with a fixed white-box communication protocol, and SMS with iteratively refined white-box protocols on `1o_2r_vs_4r`. Shaded areas indicate the variation across random seeds.

7 Can White-Box Knowledge Guide the Learning of Communication Policies?

White-box protocols provide effective and interpretable communication knowledge, yet their refinement through environmental feedback remains unstable. Learned communication policies, in contrast, can continue to adapt through RL, but must discover effective communication behaviors through optimization. This complementarity suggests a different role for white-box protocols: rather than requiring them to evolve autonomously, we use their explicit communication decisions to guide the learning of neural communication policies, while RL enables further improvement beyond the teacher.

Specifically, we instantiate learned communication with a topology network that determines communication links among agents and a message-selection network that determines what information should be shared. Given observations o^t, the white-box protocol f_P provides supervision for *who*, *when*, and *what* to communicate.

$$\bar{A}^t = f_P^{\text{who}}(o^t) \odot f_P^{\text{when}}(o^t), \qquad \bar{C}^t = f_P^{\text{what}}(o^t), \tag{2}$$

The protocol's *who* and *when* decisions jointly define the supervision target for communication topology, while its *what* decisions define the target for message selection.

Let $\widetilde{A}_\phi^t$ and $\widetilde{C}_\phi^t$ denote outputs of the topology and message-selection networks, respectively, with ϕ collecting their trainable parameters. We supervise the two networks using

$$\mathcal{L}_T(\phi) = \mathbb{E}_{o^t}\left[\ell_T\left(\widetilde{A}_\phi^t, \bar{A}^t\right)\right], \qquad \mathcal{L}_M(\phi) = \mathbb{E}_{o^t}\left[\ell_M\left(\widetilde{C}_\phi^t, \bar{C}^t\right)\right], \tag{3}$$

where ℓ_T and ℓ_M measure disagreement between network outputs and protocol targets, and the expectations are over observations used for training. The overall training objective is

$$\mathcal{L} = \mathcal{L}_{\text{RL}} + \lambda_T \mathcal{L}_T + \lambda_M \mathcal{L}_M, \tag{4}$$

where $\mathcal{L}_{\text{RL}}$ is the RL loss and the nonnegative coefficients λ_T and λ_M weight protocol supervision. The protocol supervision transfers explicit communication knowledge to the learned networks, while the RL objective allows both communication and action policies to further adapt according to task returns rather than simply imitating the teacher. The white-box protocol remains fixed and is used only to provide supervision during training; communication at execution is entirely determined by the learned networks. For network architecture and implementation details, please refer to Appendix C.

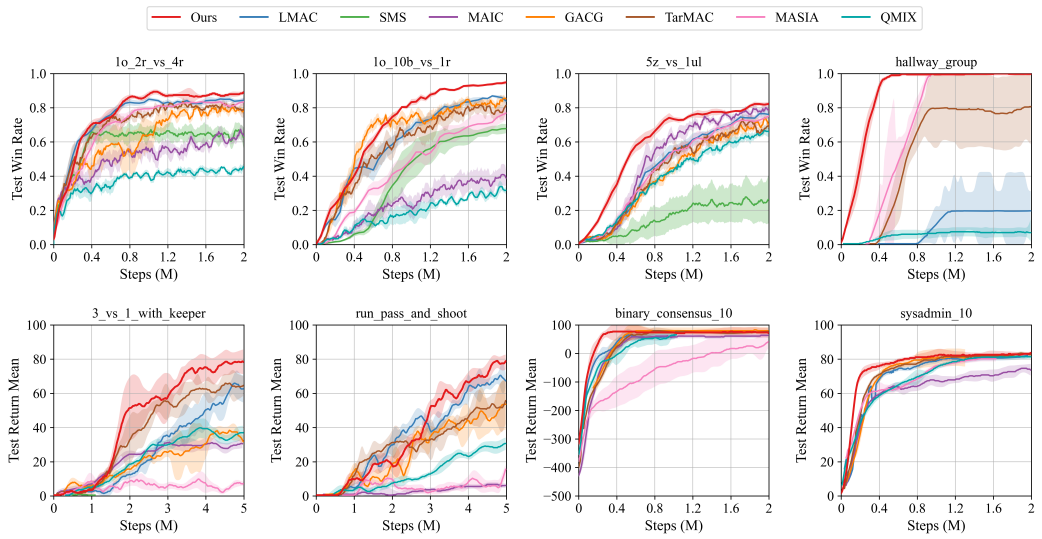

Figure 8: The performance of baseline and communication networks trained with white-box protocol supervision across eight cooperative multi-agent tasks.

We evaluate the framework on eight cooperative tasks spanning SMAC (Samvelyan et al., 2019; Guan et al., 2022), Hallway (Guan et al., 2022), Google Research Football (Kurach et al., 2020), Binary Consensus, and SysAdmin (Sintes & Bušić, 2025), covering diverse coordination requirements from combat and synchronized navigation to consensus formation and network maintenance. We compare against representative communication methods, including SMS (Xue et al., 2022), MAIC (Yuan et al., 2022), GACG (Duan et al., 2024), TarMAC (Das et al., 2019), MASIA (Guan et al., 2022), and LMAC (Bae et al., 2026), together with QMIX (Rashid et al., 2018) as a non-communication baseline. As shown in Figure 8, white-box guidance leads to faster learning and higher final performance on most evaluated tasks. The protocol supervision provides informative communication decisions early in training, reducing the need to explore ineffective communication patterns from scratch. Meanwhile, RL allows the learned communication policies to further refine these decisions according to environmental feedback. These results demonstrate that white-box protocols can serve not only as standalone communication policies, but also as an effective source of knowledge for guiding and improving learned communication policies.

8 Conclusion

In this work, we systematically revisit white-box communication in MARL. Our study shows that LLM-generated white-box protocols can achieve performance competitive with learned communication policies, and further reveals where this capability comes from and where its current boundaries lie. We find that stronger foundation models enable more effective protocol construction, while increasing rule complexity, less explicit coordination semantics, and unstable self-refinement remain important challenges. We further demonstrate that the explicit knowledge encoded in white-box protocols can effectively guide the learning of neural communication policies, combining the interpretability of white-box communication with the adaptability of reinforcement learning. Overall, our study demonstrates that white-box communication deserves renewed attention as a viable direction for multi-agent communication research. Rather than focusing solely on increasingly complex black-box optimization mechanisms, future work may further explore how emerging agentic techniques can enable more efficient, stable, and interpretable white-box communication.

AI USE STATEMENT

We used ChatGPT to assist with language editing, LaTeX formatting, organization and summarization of experimental results, and implementation of repetitive coding tasks. All AI-assisted content was carefully reviewed and revised by the authors.

REFERENCES

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Sangjun Bae, Yisak Park, Sanghyeon Lee, and Seungyul Han. Llm-guided communication for cooperative multi-agent reinforcement learning. *CoRR*, abs/2605.18077, 2026. doi: 10.48550/ARXIV.2605.18077. URL https://doi.org/10.48550/arXiv.2605.18077.

Jingdi Chen, Tian Lan, and Carlee Joe-Wong. Rgmcomm: Return gap minimization via discrete communications in multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17327–17336, 2024.

Yiqun Chen, Hangyu Mao, Jiaxin Mao, Shiguang Wu, Tianle Zhang, Bin Zhang, Wei Yang, and Hongxing Chang. Ptde: Personalized training with distilled execution for multi-agent reinforcement learning. *arXiv preprint arXiv:2210.08872*, 2022.

Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks. In *International Conference on Learning Representations*, volume 2024, pp. 11087–11115, 2024.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pp. 1538–1546, 2019.

Ziluo Ding, Tiejun Huang, and Zongqing Lu. Learning individually inferred communication for multi-agent cooperation. *Advances in Neural Information Processing Systems*, 33:22069–22079, 2020.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*, pp. 8657–8677. PMLR, 2023.

Wei Duan, Jie Lu, and Junyu Xuan. Group-aware coordination graph for multi-agent reinforcement learning. In Kate Larson (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 3926–3934. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/434. URL https://doi.org/10.24963/ijcai.2024/434. Main Track.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. SMACv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1634. URL https://papers.neurips.cc/paper_files/paper/2023/hash/764c18ad230f9e7bf6a77ffc2312c55e-Abstract-Datasets_and_Benchmarks.html.

Cong Guan, Feng Chen, Lei Yuan, Chenghe Wang, Hao Yin, Zongzhang Zhang, and Yang Yu. Efficient multi-agent communication via self-supervised information aggregation. *Advances in Neural Information Processing Systems*, 35:1020–1033, 2022.

Guangzheng Hu, Yuanheng Zhu, Dongbin Zhao, Mengchen Zhao, and Jianye Hao. Event-triggered communication network with limited-bandwidth constraint for multi-agent reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3966–3978, 2021.

Daewoo Kim, Sangwoo Moon, David Hostallero, Wan Ju Kang, Taeyoung Lee, Kyunghwan Son, and Yung Yi. Learning to schedule communication in multi-agent reinforcement learning. *arXiv preprint arXiv:1902.01554*, 2019.

Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4501–4510, 2020. doi: 10.1609/aaai.v34i04.5878.

Dapeng Li, Na Lou, Zhiwei Xu, Bin Zhang, and Guoliang Fan. Efficient communication in multi-agent reinforcement learning with implicit consensus generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23240–23248, 2025.

Huao Li, Hossein Nourkhiz Mahjoub, Behdad Chalaki, Vaishnav Tadiparthi, Kwonjoon Lee, Ehsan Moradi-Pari, Michael Lewis, and Katia Sycara. Language grounded multi-agent reinforcement learning with human-interpretable communication. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.neurips.cc/paper_files/paper/2024/hash/a06e129e01e0d2ef853e9ff67b911360-Abstract-Conference.html.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022. URL https://arxiv.org/abs/2209.07753.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pp. 6379–6390, 2017.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pp. 26516–26560, 2024.

Maja J. Matarić. Using communication to reduce locality in multi-robot learning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997. URL https://cdn.aaai.org/AAAI/1997/AAAI97-100.pdf.

Yaru Niu, Rohan R Paleja, and Matthew C Gombolay. Multi-agent graph-attention communication and teaming. In *AAMAS*, pp. 964–973, 2021.

Luís Nunes and Eugénio Oliveira. On learning by exchanging advice. *AISB Journal*, 1(3), 2003. URL https://paginas.fe.up.pt/~niadr/PUBLICATIONS/2003/aisbj.pdf.

Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

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*, pp. 1–22, 2023.

Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. FACMAC: Factored multi-agent centralised policy gradients. In *Advances in Neural Information Processing Systems*, volume 34, pp. 12208–12221, 2021.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

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.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755*, 2018.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Jules Sintes and Ana Bušić. COGNAC: Cooperative graph-based networked agent challenges for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0200. URL https://papers.neurips.cc/paper_files/paper/2025/hash/089ae994919488e764ed271a132124f1-Abstract-Datasets_and_Benchmarks_Track.html.

Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980. URL https://www.reidgsmith.com/The_Contract_Net_Protocol_Dec-1980.pdf.

Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2), 1999. URL https://www.cs.utexas.edu/~pstone/Papers/99aij/teamwork.html.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2252–2260, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Chuxiong Sun, Bo Wu, Rui Wang, Xiaohui Hu, Xiaoya Yang, and Cong Cong. Intrinsic motivated multi-agent communication. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, pp. 1668–1670, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450383073.

Chuxiong Sun, Zehua Zang, Jiabao Li, Jiangmeng Li, Xiao Xu, Rui Wang, and Changwen Zheng. T2mac: Targeted and trusted multi-agent communication through selective engagement and evidence-driven integration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 15154–15163, 2024.

Chuxiong Sun, Peng He, Qirui Ji, Zehua Zang, Jiangmeng Li, Rui Wang, and Wei Wang. M2I2: learning efficient multi-agent communication via masked state modeling and intention inference. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 25672–25681. AAAI Press, 2026. doi: 10.1609/AAAI.V40I30.39764. URL https://doi.org/10.1609/aaai.v40i30.39764.

M. Tan. Multi-agent reinforcement learning:independent vs.cooperative agent. In *The 10th International Conference on Machine Learning (ICML-93)*, 1993.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. Learning nearly decomposable value functions via communication minimization. In *ICLR 2020 : Eighth International Conference on Learning Representations*, 2020.

Yuanfei Wang, Fangwei Zhong, Jing Xu, and Yizhou Wang. Tom2c: Target-oriented multi-agent communication and cooperation with theory of mind. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=M3tw78MH1Bk.

Tianbao Xie, Siheng Zhao, Chen Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 35663–35699, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Di Xue, Lei Yuan, Zongzhang Zhang, and Yang Yu. Efficient multi-agent communication via shapley message value. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 578–584. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/82. URL https://doi.org/10.24963/ijcai.2022/82. Main Track.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

Lei Yuan, Jianhao Wang, Fuxiang Zhang, Chenghe Wang, Zongzhang Zhang, Yang Yu, and Chongjie Zhang. Multi-agent incentive communication via decentralized teammate modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 9466–9474, 2022.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026.

Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient communication in multi-agent reinforcement learning via variance based control. In *Advances in Neural Information Processing Systems*, pp. 3235–3244, 2019.

Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Succinct and robust multi-agent communication with temporal message control. *Advances in Neural Information Processing Systems*, 33:17271–17282, 2020.

Changxi Zhu, Mehdi Dastani, and Shihan Wang. Communication with factorized policy gradients in multi-agent deep reinforcement learning. *Neural Computing and Applications*, 37:18933–18956, 2025. doi: 10.1007/s00521-025-11272-9.

A Details of White-Box Protocol Construction

We construct white-box communication protocols through a two-stage process involving two LLMs, referred to as the Planner and the Coder, as illustrated in Figure 9. The Planner takes the task description p_{task} and the descriptions of individual observation dimensions p_{obs} as input and produces a communication protocol in natural language. Based on the coordination requirements and the information available to individual agents, the Planner specifies the conditions for communication, the intended senders and recipients, and the messages to transmit. The Coder then translates this description into an executable Python function f_P. Depending on the role of the communication component in the original method, f_P outputs decisions about when to communicate, whom to communicate with, or what messages to transmit. Only the functionality of the component being replaced is implemented. For example, when the original component selects communication recipients, the generated protocol implements `WHO_Strategy` and outputs the corresponding recipient selection decisions.

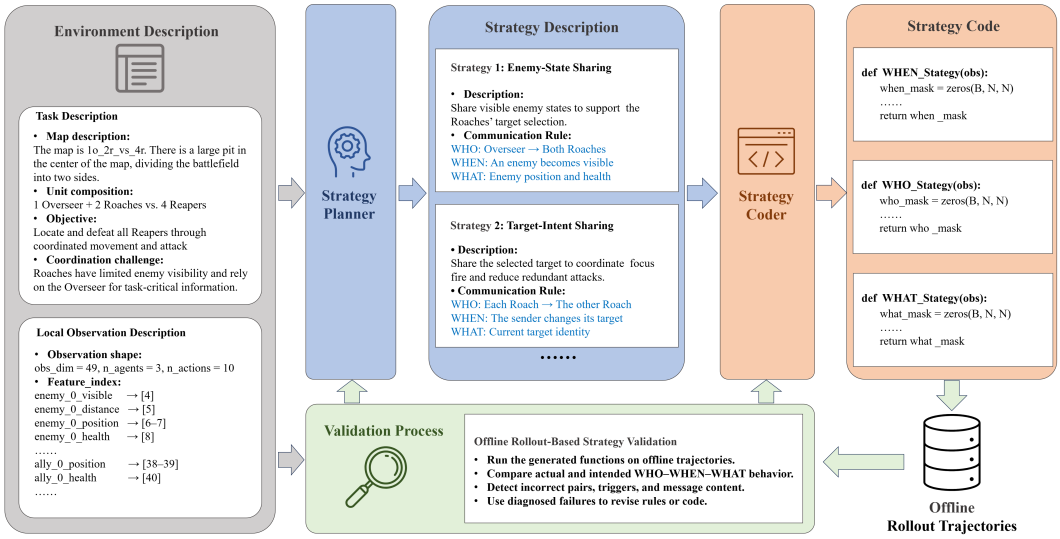

Figure 9: The process of white-box communication protocol construction.

A protocol generated in a single pass may contain inappropriate sender–recipient assignments, incorrect communication triggers, or message content that does not match the intended behavior. We therefore collect offline rollout data and execute the generated function on the recorded observations to examine its communication decisions. These decisions are compared with the natural-language protocol to identify discrepancies in communication timing, recipient selection, and message content. When problems are identified, the relevant observations, generated outputs, and descriptions of the problems are provided to the Planner as feedback. The Planner revises the protocol accordingly, and the Coder generates an updated implementation, which is then tested again on the offline data. This process alternates between protocol revision, code generation, and offline validation. The resulting protocol f_P is then frozen and used in place of the original communication component during subsequent RL training, while the action policies continue to learn.

B Case Study: Prompt Flow for White-Box Protocol Generation

We use `1o_2r_vs_4r` to show how task and observation descriptions are passed between prompts to construct a White-box Protocol. Boxes 1–4 follow a saved initial-generation record, tracing requirement IR1 to rule R1 and its code. Boxes 5–6 describe the implemented validation and optional offline refinement paths. Prompt text is condensed, and selected JSON fields are shown. Colored placeholders mark inputs supplied to a prompt. The example illustrates construction, not a measured performance gain.

$$P \longrightarrow \text{Requirements } (\texttt{IR1}) \longrightarrow \text{Rules } (\texttt{R1}) \longrightarrow \text{Code} \longrightarrow \text{Validation}$$

1. Shared Context: Task and Observation Excerpts

Input

Task description p_{task} and observation description p_{obs}, forming $P = (p_{\text{task}}, p_{\text{obs}})$.

Task excerpt

One Overseer (agent 0) and two Roaches (agents 1 and 2) face four Reapers. The supplied description states that a central pit restricts ground movement and vision, motivating information exchange.

Observation description (excerpt)

The saved prompt specifies $o : [B, 3, 66]$. Each agent has a 66-dimensional observation in this example; the following indices are zero-based. Other fields are omitted.

Index	Meaning in the saved prompt
4	Enemy 0 availability flag
5	Distance to enemy 0
6–7	Enemy 0 relative x and y
8	Enemy 0 health
9–10	Enemy 0 type indicators

Output / handoff

The same context P is supplied to the Analyzer and Planner. The Coder additionally receives the runtime dimension and feature-index map. The dimension 66 belongs to this saved example.

2. Analyzer: From Context to Information Requirements

Prompt inputs

P from Box 1:
Task: {task_description}; observations: {observation_schema}.

Prompt excerpt

Identify decision-relevant information gaps using only the task and observation schema. Do not design a policy or write code. Return JSON describing task decisions, agent groups, information requirements, and uncertainties.

Output excerpt

IR1: enemy positions, health, and type; possible sender group G1 = Overseer; possible receiver group G2 = Roaches. The Analyzer hypothesizes that these facts support Roach target selection and navigation.

Additional prompt constraints

Explain the task or observation basis for agent roles, candidate senders, and missing information. Record uncertainties instead of silently assuming that an agent, feature, threshold, or communication topology is appropriate.

Selected structured output fields

The following fields are extracted from the saved requirement; longer explanations and the remaining enemy slots are omitted.

```
{
  "requirement_id": "IR1",
  "fact": "Positions, health, and type of up to four Reaper enemies
    .",
  "possible_sender_groups": ["G1"],
  "possible_receiver_groups": ["G2"],
  "task_decision_ids": ["D1"]
}
```

Here, D1 denotes target selection and navigation by the Roaches. The saved analysis also flags sender-relative coordinates and potentially stale locations. These are qualifications of the proposed requirement, not evidence that the protocol already resolves these issues.

Prompt-to-prompt connection

The implementation parses the returned JSON and serializes it into the Planner's input under "Information-requirement analysis." Thus, the next stage receives the explicit requirement IDs and supporting fields, rather than relying on an implicit conversation history. Analyzer and Planner are two prompt stages implemented within the same planner component.

3. Planner: From Information Requirements to Rules

Prompt inputs

P and {requirements_json} from the Analyzer.

Prompt excerpt

Convert the analysis into a deterministic WHO/WHEN/WHAT specification. Bind each rule to its requirement IDs and a stable rule_id. Use aligned feature indices; state the basis of thresholds and label unsupported thresholds as hypotheses. Return JSON, not Python.

Output excerpt

Rule R1 references IR1:
WHO: agent 0 → agents 1 and 2.
WHEN: sender feature enemy_0_available > 0.5.
WHAT: sender features 5–10 (distance, relative position, health, and type). Default: no communication.

Selected policy output (R1)

This is a projection of the saved rule onto the fields needed for the example.

```
{
  "rule_id": "R1",
  "requirement_ids": ["IR1"],
  "who": {
    "sender_group": "G1",
    "receiver_selector": "G2"
  },
  "when": {
    "feature_indices": [4],
    "operator": "gt",
    "threshold": 0.5
  },
  "what": {"feature_indices": [5, 6, 7, 8, 9, 10]}
}
```

The saved rule labels the threshold basis as documented_semantics; this is the planner's annotation, not independent semantic verification.

Connection to code generation

R1 retains the link to IR1 while replacing a general information need with a concrete sender, receiver group, trigger, and feature selection. Rules R2–R4 follow the same pattern for the other enemy slots. The Coder receives the policy JSON, including rule IDs, default behavior, and uncertainties, together with runtime dimensions and feature indices.

4. Coder: From Rules to Executable Masks

Prompt inputs

{policy_spec_json} from Box 3 and {runtime_schema}, including agent count, observation dimension, and the feature-index map.

Prompt excerpt

Implement the specified rules using vectorized PyTorch. Mark each rule with its ID; do not add unlisted tactics or thresholds. Return communication_who, communication_when, and communication_what, plus a textual policy description. WHAT must preserve observation indices.

Output

WHO and WHEN: $[B, 3, 3]$ with receiver rows and sender columns. WHAT: $[B, 3, 66]$, a content mask with the same shape as o. WHO allows $0 \to 1$ and $0 \to 2$; WHEN activates these edges if any enemy-specific rule fires. R1 activates sender 0's WHAT indices 5–10. The archived code excerpt and current interface requirements are shown separately below.

Generated code excerpt (R1)

The helper and mask initialization are omitted.

```
# RULE R1: enemy_0 features (indices 5..10)
e0_avail = o[:, 0, 4] > 0.5
assign_block(0, 5, 11, e0_avail)
```

Handoff

The Python module is submitted to programmatic validation. The runtime combines the outputs as $A = \mathrm{clamp}(\mathrm{WHO} \odot \mathrm{WHEN}, 0, 1)$ and $\widetilde{o} = o \odot \mathrm{WHAT}$.

Required interface in the current Coder prompt

```
message_design_instruction()  # non-empty policy description
communication_who(o)          # [B, receiver, sender]
communication_when(o)         # [B, receiver, sender]
communication_what(o)         # same shape as o: [B, N, D]
```

WHO and WHEN must have zero self-communication diagonals; all masks lie in $[0, 1]$. Selected WHAT features stay at their original observation indices, and unselected entries are zero.

Implementation constraints in the prompt

Use only the supplied observation feature map and vectorized PyTorch operations. Do not introduce hidden state, future observations, randomness, trainable parameters, or additional tactical rules. Mark each implemented rule with its stable ID. If a rule cannot be implemented from the runtime observation, leave it inactive and explain why in the corresponding comment.

5. Validator: Interface Checks and Conditional Repair

Input

Generated code and a synthetic observation batch.

What is checked

The current implementation runs the generated functions on a synthetic batch of two observations. It checks that the required callables exist, the description is non-empty, WHO/WHEN have the expected matrix shape, WHAT matches the observation shape, values stay within the required range, and self-communication diagonals are zero. An AST check rejects loops and comprehensions in the communication functions.

Failure-to-repair template

```
[BUG] Generated teacher failed validation:
{exception}

Traceback:
{traceback}
```

Fix only the reported interface/runtime problem.

What the next Coder call receives

The structured policy and runtime schema are included again, with the accumulated validation feedback appended. If no Python code block can be extracted, the feedback instead requests a complete block containing the required communication functions. The attempt limit is ten in the default code-generation call.

Outcome

A passing candidate is saved as `comm_init.py`. If the attempt budget is exhausted, generation returns failure. These checks test interface compatibility; a synthetic input batch does not establish that a tactical interpretation is correct or that return improves. This box describes the implemented repair path, not a recorded failure of R1.

6. Optional Offline Refinement: From Evidence to Revision

Prompt inputs

{candidate_code}, {rollout_summary}, task/schema information, interface checks, and observed communication behavior.

Judge prompt excerpt

Evaluate WHO/WHEN/WHAT using the supplied rollout and code evidence. Return structured feedback, including blocking failures and rule checks. Do not use downstream RL win rate or hidden state.

Judge output and revision input

The structured feedback includes acceptance and scoring fields, checks on individual rules, and blocking failures. The revision stage receives this feedback together with the candidate code and rollout context.

Revision template (condensed).
Candidate: {candidate_code}
Rollout evidence: {rollout_summary}
Judge feedback: {judge_feedback}

Revise the candidate communication policy using the supplied evidence, while preserving the required WHO/WHEN/WHAT interface.

Connection back to validation

The revised module is validated and evaluated again before another judgment or revision step. This changes a candidate protocol using offline evidence; it is distinct from repairing a tensor-shape error in Box 5. The interface explicitly excludes downstream RL win rate and hidden state from the Judge's evidence. This optional mechanism is described from the implementation and is not presented as a recorded revision episode for R1.

C Implementation Details of White-Box-Guided Communication Learning

To describe the implementation of white-box-guided communication learning, we follow the data flow in Figure 10, from protocol targets and network predictions to message aggregation and joint optimization. Algorithm 1 summarizes the training procedure.

Protocol targets and network predictions. Before training, we construct f_P using the task and observation descriptions, following Section 3.2. For the observations used in a training update, we evaluate both f_P and the communication networks. The protocol produces $\bar{A}^t$ and $\bar{C}^t$ according to Equation 2: a topology target is active only when the corresponding *who* and *when* conditions are both satisfied, while the *what* output marks the observation components to share. The TargetMatrix and MessageSelect networks produce $\widetilde{A}^t_\phi$ and $\widetilde{C}^t_\phi$, respectively. These predictions determine the messages used by the action policies; the protocol outputs serve as supervision targets rather than replacing the predictions during training.

Message selection and aggregation. The message-selection output is applied element-wise to the observation input to select the information each sender shares. The topology output then determines how this information is combined for each receiving agent. The operation shown in the figure can

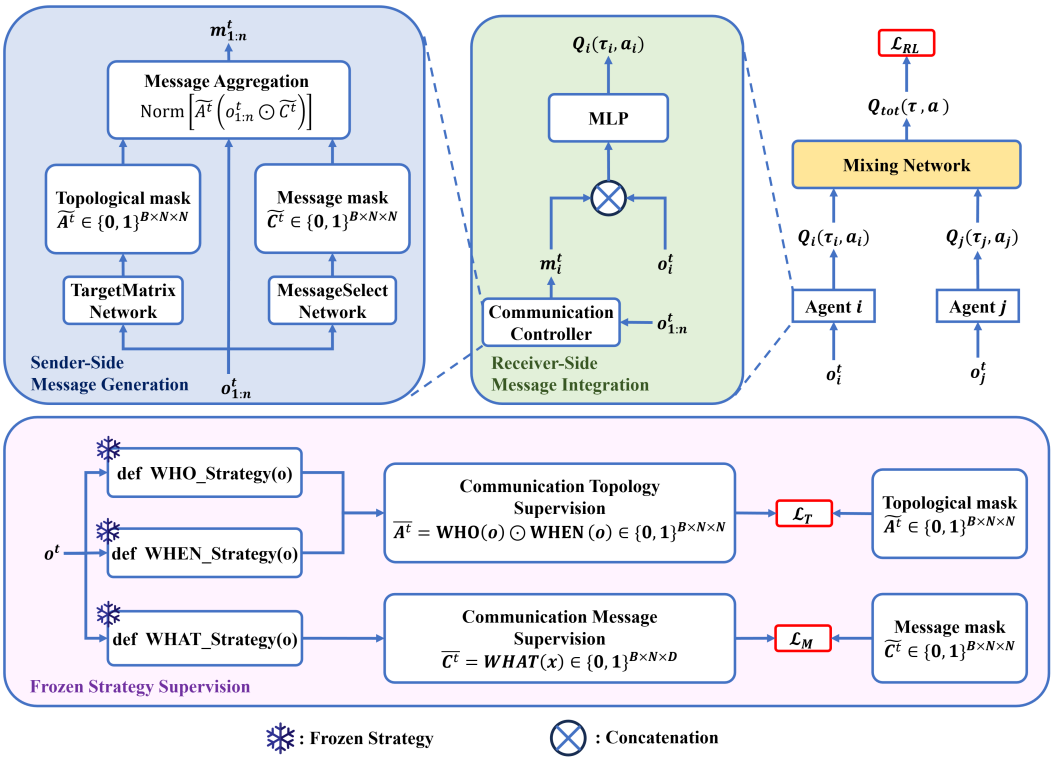

Figure 10: The framework for distilling white-box protocol knowledge into communication networks through joint supervision and reinforcement learning.

be written as

$$c^t = \mathrm{Norm}\left[\widetilde{A}^t_\phi \left(o^t \odot \widetilde{C}^t_\phi\right)\right], \tag{5}$$

where multiplication by $\widetilde{A}^t_\phi$ aggregates across senders and Norm denotes the normalization operation in the figure. In this expression, rows of the topology matrix index receivers and columns index senders; c^t_i is the message received by agent i. The batch dimension is omitted for clarity. This expression describes the processing order without specifying the implementation of normalization or mask discretization.

Action selection and environment interaction. Each agent concatenates c^t_i with its local observation o^t_i and passes the result to its action-value network, shown as an MLP in the figure. The received message is labeled m^t_i in the figure and corresponds to c^t_i in the manuscript. The resulting action values are used to select the agents' actions. Executing the joint action produces a shared reward and the next observations. These interactions provide the training data for the RL objective. During training, the mixing network combines the individual action values into Q_{tot} for computing $\mathcal{L}_{\mathrm{RL}}$.

Joint updates and execution. For each training batch, the topology and message-selection predictions are compared with their corresponding protocol targets to compute $\mathcal{L}_T$ and $\mathcal{L}_M$. We combine these terms with the RL loss using Equation 4. Let θ collect the action-value and mixing-network parameters, while ϕ denotes the communication-network parameters. The joint objective updates θ and ϕ; no updates are applied to f_P. Algorithm 1 separates environment interaction from training updates so that it does not prescribe a particular update frequency or data-sampling scheme. During evaluation, agents use the learned communication networks and action policies without generating protocol targets or calling the LLM.

Algorithm 1 White-Box-Guided Communication Learning

Require: Fixed protocol f_P; supervision weights λ_T, λ_M
Initialize communication parameters ϕ and action-value/mixing parameters θ
while training budget remains **do**
  Collect environment interactions:
  for each interaction step t **do**
    Obtain observations o^t
    Predict $\widetilde{A}^t_\phi$ and $\widetilde{C}^t_\phi$ using the two communication networks
    Compute received messages c^t using Equation 5
    Compute individual action values from local observations and received messages
    Select and execute joint action $\mathbf{a}_t$
    Record reward, next observations, and other quantities required for RL training
  end for
  Perform training updates:
  for each training batch **do**
    Recompute communication predictions and action values for the batch
    Generate $\bar{A}^t$ and $\bar{C}^t$ with f_P using Equation 2
    Compute $\mathcal{L}_T$ and $\mathcal{L}_M$ using Equation 3
    Compute Q_{tot} and the RL loss $\mathcal{L}_{\text{RL}}$
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{RL}} + \lambda_T \mathcal{L}_T + \lambda_M \mathcal{L}_M$
    Update ϕ and θ by minimizing $\mathcal{L}$; keep f_P fixed
  end for
end while
return Learned communication networks and action policies

D Experimental Details

To document the experimental settings, we describe the task configurations, baseline methods, and training and evaluation procedures used in this study. Task compositions are summarized in Tables 1–3.

D.1 Environment Details

SMAC communication scenarios. SMAC (Samvelyan et al., 2019) evaluates cooperative micromanagement under partial observability. We use the three communication scenarios in Table 1. In the two scenarios with an Overseer, agents need to share information to support enemy localization and coordinated attacks (Guan et al., 2022; Wang et al., 2020). These tasks are used for protocol replacement, comparisons of protocols, and communication learning. Performance is measured by test win rate.

Table 1: Team compositions of the SMAC communication scenarios.

Scenario	Agents	Allied composition	Opponent composition
`1o_2r_vs_4r`	3	1 Overseer, 2 Roaches	4 Reapers
`1o_10b_vs_1r`	11	1 Overseer, 10 Banelings	1 Roach
`5z_vs_1ul`	5	5 Zealots	1 Ultralisk

SMACv2 Terran. SMACv2 (Ellis et al., 2023) generates scenarios with randomized team compositions. We vary team size and the number of available agent types separately, as summarized in Table 2. The team-size comparison retains the three-type sampling distribution; the team-composition comparison fixes ten agents per team. Sampling probabilities do not prescribe exact type counts in each episode. We compare GACG and its white-box variant using test win rate.

Hallway Group. The `hallway_group` task evaluates coordination of arrival times under local observations. The grouped Hallway benchmark requires agents within a group to synchronize their arrival, while different groups coordinate distinct arrival times (Guan et al., 2022). We evaluate communication learning using the fraction of successful test episodes.

Table 2: SMACv2 Terran configurations for team-size and team-composition comparisons. The 10-versus-10 three-type setting is shared by both comparisons.

Comparison	Agents per team	Available types	Sampling probabilities
Team size	5	3	$(0.45, 0.45, 0.10)$
Team size	10	3	$(0.45, 0.45, 0.10)$
Team size	15	3	$(0.45, 0.45, 0.10)$
Team size	20	3	$(0.45, 0.45, 0.10)$
Team composition	10	1	(1)
Team composition	10	2	$(0.50, 0.50)$
Team composition	10	3	$(0.45, 0.45, 0.10)$

Google Research Football. We use two cooperative attacking scenarios from Google Research Football (Kurach et al., 2020): `3_vs_1_with_keeper` and `run_pass_and_shoot`. They require players to coordinate movement, passing, and shooting. The scenario compositions are listed in Table 3; these tasks are evaluated using mean test returns.

Binary Consensus and SysAdmin. We use the ten-agent tasks `binary_consensus_10` and `sysadmin_10`. These networked cooperation problems, also described in COGNAC (Sintes & Bušić, 2025), concern agreement among agents with binary states and coordinated maintenance of a computer network, respectively. We report mean test returns.

Multi-Agent MuJoCo. Multi-Agent MuJoCo (Peng et al., 2021) assigns different subsets of a physical system's actuators to different agents. We use the two HalfCheetah partitions in Table 3 to compare CFPG with its white-box variant. Both control the same six actuated joints but differ in how control is distributed among agents. Performance is measured by test return. Protocol construction uses descriptions of the task and observations; these experiments examine communication based on continuous physical measurements.

Table 3: Configurations of additional evaluation tasks. Football compositions describe the scenario participants; controller assignments depend on the environment wrapper.

Task	Composition or control allocation
`3_vs_1_with_keeper`	3 attackers versus 1 defender and 1 goalkeeper
`run_pass_and_shoot`	2 attackers versus 1 defender and 1 goalkeeper
`binary_consensus_10`	10 agents participating in binary consensus
`sysadmin_10`	10 agents maintaining a computer network
`HalfCheetah-2x3`	2 agents, each controlling 3 actuated joints
`HalfCheetah-6x1`	6 agents, each controlling 1 actuated joint

D.2 Baselines

IMMAC (Sun et al., 2021) uses observation-dependent intrinsic motivation to assess the importance of information and guide communication. In our replacement experiments, its communication-timing component is replaced by a white-box protocol, allowing us to examine protocol decisions about when to communicate.

SMS (Xue et al., 2022) evaluates message contributions using Shapley message values to support selective communication. We use SMS to study replacement of communication-partner selection and include its original method in the communication-learning comparison.

M2I2 (Sun et al., 2026) combines masked state modeling and intention inference to improve the use of communicated information. Its dimensional importance mechanism selects information to mask and share. Our replacement experiments focus on this message-content selection function; M2I2 also provides the setting for comparing protocols generated by different LLM versions.

MAIC (Yuan et al., 2022) uses decentralized teammate modeling to support communication tailored to teammates. It is included as a learned communication baseline in Section 7.

GACG (Duan et al., 2024) learns group-aware coordination graphs to represent relationships among agents and guide information exchange. We compare GACG with its white-box variant when varying team size and composition, and include the original method in the communication-learning comparison.

TarMAC (Das et al., 2019) learns targeted communication using an attention mechanism that determines how messages are weighted for different receiving agents. It provides an attention-based communication baseline.

MASIA (Guan et al., 2022) aggregates messages into a shared representation, trained through self-supervised reconstruction and prediction. Individual agents then extract relevant information from this representation for decision-making. It provides a baseline that emphasizes message aggregation and utilization.

LMAC (Bae et al., 2026) uses an LLM to construct executable communication protocols and refines them using feedback on the accuracy and consistency of agents' state reconstruction. It is included as an LLM-based communication baseline, complementing the learned communication methods.

QMIX (Rashid et al., 2018) learns a monotonic decomposition of a joint action-value function into individual action-value functions. Its decentralized policies do not include an explicit communication module. We include it as a value-decomposition baseline in the communication-learning experiments.

CFPG (Zhu et al., 2025) combines learned communication with factorized policy gradients for cooperative multi-agent control. We use it as the baseline for the continuous-control experiments, comparing the original method with a variant in which a white-box protocol replaces its learned communication component.

D.3 Training and Evaluation Settings

The replacement experiments compare **Original**, **WB-Train**, and **WB-Direct** as defined in Section 4. For each baseline and task, **WB-Train** and **WB-Direct** use the same protocol. The comparisons of protocol sources and LLM versions, together with the scaling and continuous-control experiments, follow **WB-Train**. In the communication-learning experiments, we instead use protocol outputs as supervision and train a separate model for each task, following Section 7 and Appendix C. Unless otherwise specified, all MARL experiments are conducted with five random seeds.

D.4 LLM Settings

Except for the GPT-model comparison in Section 5, we use DeepSeek-V3 for all LLM-based protocol-construction stages, including the Planner and Coder; the comparison covers GPT-3.5 Turbo, GPT-4o, GPT-4.1 (`gpt-4.1-2025-04-14`), GPT-5 (`gpt-5-2025-08-07`), and GPT-5.6 (`gpt-5.6`).

Table 4: Hyperparameters of the white-box-guided communication learner on SMAC. The settings correspond to the current training configuration and the default teacher–student launcher; they are not shared settings for all compared baselines. Baseline-specific hyperparameters follow their original implementations unless otherwise stated.

Hyperparameter	Setting
Agent recurrent hidden dimension	64
Communication-selector hidden dimension	64
Latent representation dimension	20
QMIX mixing embedding dimension	32
Hypernetwork layers	2
Hypernetwork hidden dimension	64
Optimizer	Adam
Learning rate	5×10^{-4}
Discount factor γ	0.99
Training batch size (episodes)	32
Replay-buffer capacity (episodes)	5,000
Gradient-norm clipping	10
Target-network update interval (training updates)	200
Double Q-learning	Enabled
Exploration ϵ (initial $\rightarrow$ final)	$1.0 \rightarrow 0.05$
Linear exploration annealing (environment steps)	50,000
Teacher-loss weight (initial $\rightarrow$ final)	$1.0 \rightarrow 0.1$
Linear teacher-weight annealing (environment steps)	500,000
Student communication gating (training/evaluation)	Soft / Soft
Training budget (environment steps)	2,050,000
Evaluation interval (environment steps)	50,000
Evaluation episodes	100
Evaluation action selection	Greedy