Whisperwise: Emergent Communication in Multi-Agent Reinforcement Learning for Complex Tasks

Part II Computer Science Project Proposal

October 10, 2025

Project Supervisor:	**Ian Lewis**
Director of Studies:	**Ramsey Faragher**

1 Introduction

Since Google DeepMind released AlphaGo in 2016, there has been a steady increase in interest in the field of reinforcement learning. This has led to breakthroughs in areas of finance, trading, robotics and even video games. A clear next step for reinforcement learning is multi-agent reinforcement learning, where multiple agents are placed into the same environment and have to learn to interact with one another efficiently. This area of research has the potential to improve surgery robots, traffic optimisations and search and rescue operations. However, there exists a range of challenges which make multi-agent reinforcement learning more difficult in practice, such as: non-stationary problem, credit assignment issues, and the one I will focus on in this project, communication problems. Unlike single-agent reinforcement learning, multi-agent reinforcement learning can communicate to improve the agent's knowledge of the environment and/or teach them the optimal next step. A widespread issue with communication is the "Joint Exploration Problem", which is a struggle for agents to communicate effectively in novel environments.

I propose to extend AI Mother Tongue[1] to a realistic situation for a more rigorous evaluation of the paper's proposed method. I will simulate an assembly line for multiple agents to evalutae the communication protocols. I will train the agents using a Value Decomposition Network to work together to explore the area. I will have a continuous communication protocol which broadcasts raw information from agent to agent and use this as a baseline to compare to my emergent communication protocol. This other protocol will use a vector quantised variational autoencoder (VQ-VAE) to generate discrete symbols to broadcast from agent to agent.

This new method, using a VQ-VAE, allows communication protocols to emerge naturally rather than relying on manually designed rules or incentives. The limited alphabet forces agents to communicate with more compact, informative, and interpretable messages that prioritise essential information sharing to achieve coordination. Hence, we expect an improvement over the baseline communication protocol in the time taken to explore a given environment.

2 Structure of the Project

There are five main components to this project:

- Simulation
- Value Decomposition Network
- Communication Protocols
- Training
- Evaluation: refer to the Evaluation section

Simulation

The simulation is required as it will be how the reinforcement learning algorithm is both trained and evaluated. It will be responsible for the state management and local observations of each agent. Interactions will be handled through discrete timesteps, where agents produce actions informed by their policy and their communication inputs. As training multi-agent systems will require a lot of runs, the simulation must be optimised highly. It will be written in Rust for this reason and mainly function as a headless program (only providing a simple graphical UI when demoing single runs).

The simulation will involve agents working on an assembly line and communicating to pass items along the line.

Value Decomposition Network

The reinforcement learning will optimise the agent's policy (the agent's mapping from state space to action space). I have chosen to use a Value Decomposition Network to facilitate decentralised policy learning, as well as simultaneously optimising for a joint reward. Each agent will maintain its own Q function.

We want the agents to learn coordinated behaviour but maintain their own local observations and local policy. We also want to use a global reward function (successful items assembled). Due to these factors, I have chosen to use a Value Decomposition Network (VDN). In VDN, the value function of the entire team, Q_{tot}, is factorised as the sum of the individual agents' value functions:

$$Q_{\text{tot}}(\tau, a) \;=\; \sum_{i} Q_i(\tau_i, a_i)$$

where Q_i is the action-value function for agent i, based only on its own action a_i and local action-observation history τ_i.

The VDN framework is particularly suitable for my project's cooperative multi-agent tasks:

- It naturally encourages cooperation by tying all agents' learning to the same joint reward.

- It is lightweight and efficient compared to more complex mixing architectures (like QMIX), allowing me more time to focus on other parts of the project.
- The decomposition aids in the interpretability of learned policies, as each agent's value estimates can be analysed independently.

Planned Implementation:

- Each agent will be equipped with its own Q-network, parameterised by a deep neural network, receiving as input both its local observations and any messages broadcast by other agents.
- The value decomposition will be enforced in the training loss, aligning each agent's updates to the global objective.
- This setup will also allow for seamless integration and fair comparison of different communication protocols, as the underlying learning dynamics for coordination are held constant.

Communication Protocols

This is the core of the project. I will primarily produce two distinct communication protocols. A simple protocol which has the goal of providing a baseline for the next protocol. It will emit raw continuous vectors. Our other protocol will emit symbols. These will be produced from a VQ-VAE (Vector Quantised - Variational Autoencoders), providing a discrete alphabet for agents to communicate via. Both of the protocols will initially broadcast every message to every other agent.

Training

The training will manage the model's optimisation process. It will use the simulation to collect data and update both the agents' Q functions and, when required, the VQ-VAE. I will add a simple logging and checkpointing framework to prevent loss of results.

3 Evaluation

Quantitive

I will gather a small subset of simulations not yet seen by the training, and run each model with the different communication protocols on each. I will then compare the average time for the agents to assemble all the items as well as a count of successfully assembled items.

Qualitative

I will generate graphs of the distribution of the messages passed between agents. I will also observe the evaluation runs and note down strategies I see the agents come up with. I will then compare the two models' strategies.

4 Starting Point

I have limited experience in using reinforcement learning. However, I have previously built an autoencoder when analysing the stock market, which will help me develop the variational autoencoder. Furthermore, I have a small amount of experience using Rust from a three month internship I completed, but I have strong fluency in Python. This will help my development of a high-performance backend (Rust) and the ML (Python).

5 Success Criteria

1. Design and implement a simulation to train and demo multi-agent interactions. The simulation should be able to have single runs visualised. The simulation should control the state of every agent and collect data.
2. Implement a functional VDN-based multi-agent reinforcement learning (MARL) system capable of learning, given a simulation program with set methods.
3. Develop a communication protocol that allows agents to transmit raw data between each other.
4. Similarly, develop another communication protocol that uses a VQ-VAE to transmit vectors between agents.
5. The training and evaluation pipeline should be able to train and evaluate models automatically and allow interchangeably using both communication protocols.

6 Extensions

- Implement a third communication protocol using a hierarchically quantised variational autoencoder (HQ-VAE). This method was suggested by the AI Mother Tongue paper as a possible future work.
- Add hyperparameter tuning for my discrete communication protocol. This can involve the language size, quantisation levels, and message frequency to optimise agent coordination and learning efficiency.
- Add a new task where agents must solve for objectives other than exploration (e.g., coordinated navigation, resource gathering, or adversarial avoidance), enabling analysis of communication protocol effectiveness in more diverse MARL settings.

7 Timetable and Milestones

Weeks 1–2 (20 Oct 2025 – 2 Nov 2025)

- Set up Rust-based simulation environment using PyO3 to integrate with PyTorch.
- Prepare PyTorch environment for VDN.

Weeks 3–4 (3 Nov 2025 – 16 Nov 2025)

- Implement simulation for warehouse assembly line.
- Develop a wrapper for the simulation in Rust with py03 using an abstract class (to help potential later extension of using other tasks).

Weeks 5–6 (17 Nov 2025 – 30 Nov 2025): Core VDN Build

- Implement basic value decomposition networks (VDN).
- Develop basic training loop and agent coordination.
- *Milestone:* VDN architecture operational and agents learning basic tasks.

Weeks 7–8 (1 Dec 2025 – 14 Dec 2025)

- Integrate a continuous communication channel into the agent architecture using a modular design.
- Determine which data each agent should communicate.
- *Milestone:* Continuous communication protocol up and running.

Weeks 9–10 (15 Dec 2025 – 28 Dec 2025)

- Develop a variational autoencoder which quantises the latent space (VQ-VAE)
- Integrate the VQ-VAE into a discrete communication protocol for the agents.
- *Milestone:* Discrete communication protocol up and running.

Weeks 11–12 (29 Dec 2025 – 11 Jan 2026)

- Build a training pipeline supporting both communication protocols.
- Implement regular model checkpoints and experiment tracking.
- Automate logging and result collection for training runs.
- *Milestone:* Training pipeline working.

Weeks 13–14 (12 Jan 2026 – 25 Jan 2026)

- Run training on agents using both discrete and continuous communication protocols.
- Collate performance logs and monitor checkpointed models.
- *Milestone:* Complete initial fully trained models for both communication types.

Weeks 15–16 (26 Jan 2026 – 8 Feb 2026)

- Implement scripts to calculate key metrics: learning efficiency, coordination, bandwidth use.
- Automate model evaluation and result logging.

Weeks 17–18 (9 Feb 2026 – 22 Feb 2026)

- Start on extensions, focusing initially on implementing the alternative discrete communication protocol with an HQ-VAE.

Weeks 19–20 (23 Feb 2026 – 8 Mar 2026)

- Catch up on any uncompleted work or drafts.
- Any remaining time can be used to work on extensions.

Weeks 21–22 (9 Mar 2026 – 22 Mar 2026)

- Evaluate all models (continuous, discrete) and tested extensions on chosen environments.
- Collect and analyse results: success rate, sample efficiency, agent coordination, communication cost.

Weeks 23–24 (23 Mar 2026 – 5 Apr 2026)

- Catch up on any uncompleted work or drafts.
- Any remaining time can be used to work on extensions.

Weeks 25–26 (6 Apr 2026 – 19 Apr 2026)

- Write detailed chapters on implementation and evaluation.
- Create figures, tables, and result summaries.

Weeks 27–28 (20 Apr 2026 – 3 May 2026)

- Revise dissertation draft and submit for supervisor feedback.
- Edit dissertation based on feedback.

Week 29 (4 May 2026 – 17 May 2026)

- Final proofreading and formatting.
- Submit dissertation.
- Reserve leftover time for contingency: extra tests, additional evaluation, or further analysis if needed.
- *Milestone:* Submit dissertation and project code.

8 Resource Declaration

I will be using my personal desktop computer (AMD Ryzen 7 3700X, 32GB RAM, NVIDIA RTX 5060 Ti) as my primary machine for software development. As a backup, I will use my personal laptop and resources provided by SRCF (student-run computing facility). I will continuously back up my code and dissertation with Git version control on GitHub.

References

[1] Hung Ming Liu. “AI Mother Tongue: Self-Emergent Communication in MARL via Endogenous Symbol Systems”. In: *arXiv preprint arXiv:2507.10566* (2025). URL: `https://arxiv.org/abs/2507.10566`.