TacticCraft: Natural Language-Driven Tactical Adaptation for StarCraft II

Weiyu Ma [1 2] **Jiwen Jiang** [1 2] **Haobo Fu** [3] **Haifeng Zhang** [1 2 4]

Abstract

We present an adapter-based approach for tactical conditioning of StarCraft II AI agents. Current agents, while powerful, lack the ability to adapt their strategies based on high-level tactical directives. Our method freezes a pre-trained policy network (DI-Star) and attaches lightweight adapter modules to each action head, conditioned on a tactical tensor that encodes strategic preferences. By training these adapters with KL divergence constraints, we ensure the policy maintains core competencies while exhibiting tactical variations. Experimental results show our approach successfully modulates agent behavior across tactical dimensions including aggression, expansion patterns, and technology preferences, while maintaining competitive performance. Our method enables flexible tactical control with minimal computational overhead, offering practical strategy customization for complex real-time strategy games.

1. Introduction

StarCraft II represents one of the most challenging environments for artificial intelligence research, combining imperfect information, long-term planning, and real-time decision-making. The release of PySC2 by DeepMind and Blizzard (Vinyals et al., 2017) catalyzed research in this domain, culminating in AlphaStar (Vinyals et al., 2019)—the first AI system to defeat top professional players. Following AlphaStar's success, industry efforts like Tencent's TStarBot(Han et al., 2020) and SenseTime's open-source DI-Star(star Contributors, 2021) have further advanced StarCraft II AI capabilities.

Despite these technical achievements, current StarCraft II agents optimize primarily for win rate, often neglecting gameplay diversity and human-like behaviors that are valued by players and game developers. Moreover, these agents lack intuitive interfaces for non-technical users to specify strategic preferences or tactical styles.

We address these limitations by enabling natural language conditioning of a state-of-the-art StarCraft II agent. Building upon DI-Star, we introduce tactical conditioning through lightweight adapter modules that allow the agent to execute diverse gameplay styles while maintaining competitive performance. Our approach consists of two key components:

First, we develop a dataset connecting natural language to gameplay by processing high-level player replays and using large language models to classify tactical styles, represented as a tactic tensor—a probability distribution over predefined tactical archetypes.

Second, we implement an adapter-based architecture inspired by ControlNet (Zhang et al.) in computer vision. We freeze the original policy network and insert zero-initialized adapter modules that process the tactic tensor to modify the policy outputs according to the specified tactical style.

Our contributions include: (1) a methodology for labeling gameplay trajectories with tactical descriptors using LLMs, (2) an adapter-based approach for conditioning complex policy models, and (3) a system that enables natural language control over agent behavior while preserving fundamental gameplay competencies. By combining LLMs and reinforcement learning policies, our work represents a step toward more accessible, customizable game AI agents that respond to human strategic preferences.

2. Dataset Collection

2.1. StarCraft II Language Corpus

To establish a comprehensive StarCraft II natural language corpus, we aggregated data from prominent community resources, including Spawningtool[1] and Liquipedia[2]. These platforms contain thousands of game guides, match analyses, and strategic discussions that form a rich linguistic foundation for StarCraft II gameplay concepts.

[1]Institute of Automation, Chinese Academy of Sciences, China [2]School of Artificial Intelligence, University of Chinese Academy of Sciences, China [3]Tencent AI Lab, Shenzhen, China [4]Nanjing Artificial Intelligence Research of IA, China. Correspondence to: Haobo Fu <haobofu@tencent.com>, Haifeng Zhang <haifeng.zhang@ia.ac.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

[1]https://lotv.spawningtool.com/
[2]https://liquipedia.net/starcraft2

Given the absence of standardized nomenclature for tactical approaches in StarCraft II, we employed n-gram analysis to extract tactical elements from our corpus. Subsequently, we collaborated with expert players (Grandmaster level) to formalize 8 prevalent tactical paradigms with corresponding linguistic descriptions, creating a structured taxonomy of StarCraft II strategies.

2.2. Replay Processing and Build Order Extraction

Our replay dataset consists of approximately 50,000 Zerg versus Zerg (ZvZ) replays provided by the DI-Star team. We processed these replays using the pysc2 framework to decode each match into standardized StarCraft II build orders—a sequential representation of in-game actions that players traditionally use to communicate strategic progression. To ensure quality, we filtered out replays with Match Making Rating (MMR) below 4800, retaining only high-level competitive gameplay.

A typical build order follows this format:

Supply	Time	Build Action
13	0:12	Overlord
16	0:48	Hatchery
17	1:09	Spawning Pool
17	1:21	Extractor
19	1:49	Overlord
19	1:58	Queen ×2
24	2:04	Overlord
27	2:27	Roach Warren
28	2:54	Overlord ×2

Table 1. Example of a StarCraft II build order sequence

Build orders have historically been used by players to classify gameplay styles and by researchers as reward signals during reinforcement learning.

2.3. Tactical Classification Using Large Language Models

To establish connections between build orders and tactical archetypes, we leveraged large language models (LLMs)—specifically GPT-4 and DeepSeek-v3 to classify each build order according to our established tactical taxonomy. For each classification, we extracted log probabilities across all tactical categories and normalized them using softmax to create distribution vectors. We also incorporated an "Unclear" category to accommodate build orders that did not align clearly with defined tactics.

The resulting dataset entries follow this structure:

```
Tactic dist: [0.1, 0.8, ..., 0.1]
Replay ID: xxxxxx
```

This approach provides not only discrete tactical labels but also confidence metrics that reflect the probabilistic nature of strategic classification in StarCraft II.

3. Methodology

3.1. Base Policy Network

The foundation of our approach is the DI-Star policy network for StarCraft II, as illustrated in Figure 3. This architecture processes the complex game state through multiple specialized components:

The observation encoder converts raw game features into dense representations, capturing spatial (map features), entity-based (units and buildings), and scalar information (resources, game phase). A core LSTM network then integrates these features over time to maintain temporal consistency in decision-making.

The policy network's output is structured as multiple specialized heads corresponding to different aspects of StarCraft II gameplay: *Action Type*, *Target Unit*, *Location*, *Selected Units*, *Delay*, and *Queued*. This multi-head design enables the network to handle the structured action space of StarCraft II, where each action requires specifying several interdependent parameters.

3.2. Adapter Architecture and Tactical Conditioning

To introduce tactical variability while preserving the base capabilities of the DI-Star policy, we employ an adapter-based architecture as shown in Figure 1. The key innovation is the introduction of tactical conditioning through a low-dimensional tactic tensor $\tau \in \mathbb{R}^d$ where $d = 9$ in our implementation. This tactic tensor encodes the desired tactical style (aggressive, defensive, economic, etc.) and is provided as direct input to the adapter modules, while the original encoder network continues to process standard StarCraft II game state inputs.

For each action head, we introduce a lightweight adapter module consisting of a two-layer MLP ($\tau \to h_1 \to h_2 \to o$, where $h_1 = 64$, $h_2 = 32$) with ReLU activations and zero-initialization. The adapter output is combined with the original policy output using one of several fusion methods, with additive fusion ($f(x, y) = x + y$) providing the best balance of performance and efficiency in our experiments.

The fusion outputs produce the final logits that determine action probabilities. Our empirical results show that the additive fusion method provides an effective balance of performance and computational efficiency.

3.3. Training Methodology

During training, we freeze all parameters of the original policy network and only update the adapter modules. This approach ensures that the core capabilities of the base policy remain intact while allowing the adapters to learn tactical modifications.

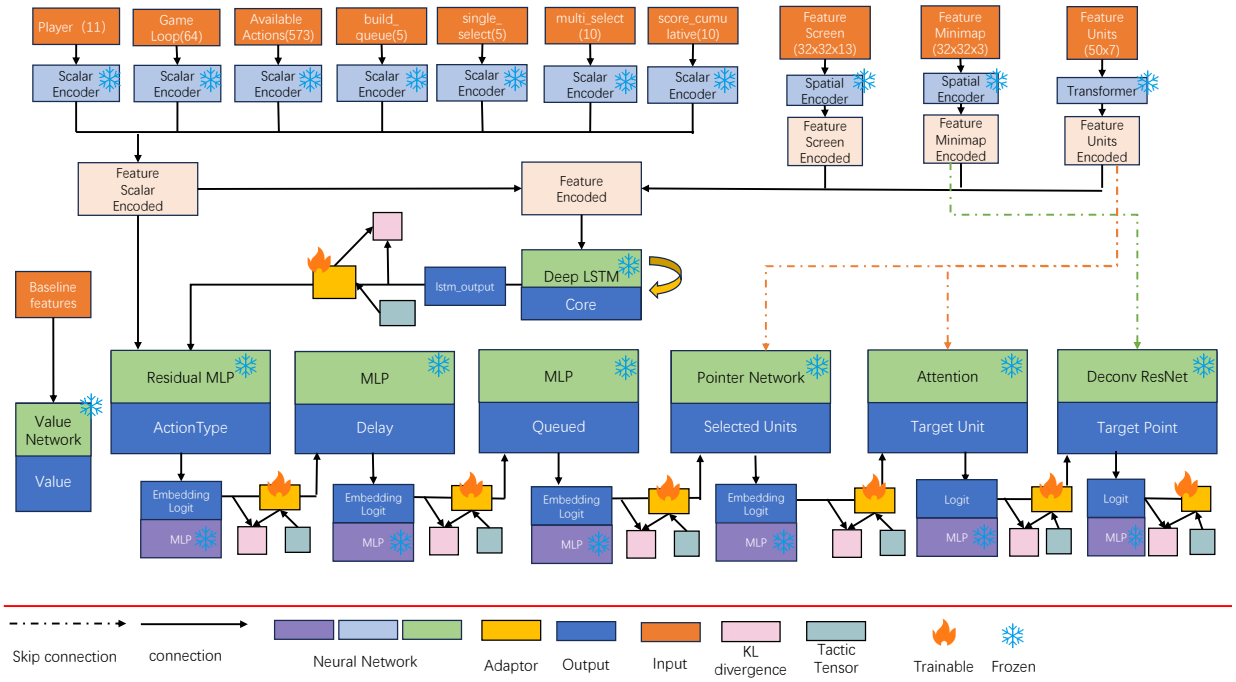

Figure 1. The adapter-based policy architecture. The original policy network (shown in gray) remains frozen, while lightweight adapter modules (shown in blue) modify network outputs according to a tactical conditioning vector. The architecture employs a multi-head design, with separate adapters for different action types required in StarCraft II gameplay.

Our training objective is formulated as a knowledge distillation framework with KL divergence constraints:

$$\mathcal{L}(\phi) = \mathbb{E}_{s\sim D}\left[\sum_{h\in H} \alpha_h \cdot \mathrm{KL}(\pi_\theta^h(a|s)\|\pi_{\theta,\phi}^h(a|s,\tau))\right] \tag{1}$$

where:

- ϕ represents the adapter parameters (the only parameters being optimized)
- θ represents the frozen parameters of the original policy
- H is the set of action heads
- α_h is the weight for the KL divergence term for head h
- π_θ^h and $\pi_{\theta,\phi}^h$ are the output distributions of head h for the original and adapted policies
- D represents the replay dataset

By using different weights α_h for different action heads, we allow more tactical flexibility in certain aspects of gameplay (such as unit targeting and positioning) while maintaining stronger adherence to the original policy in fundamental action selection.

The KL divergence objective serves two crucial purposes: it prevents the adapted policy from deviating too far from the original policy's capabilities, while still allowing for tactical adaptations based on the conditioning vector. This balance ensures that the resulting policy maintains the strong gameplay mechanics of the original DI-Star agent while exhibiting the desired tactical characteristics.

4. Experiments

To evaluate our adapter-based tactical conditioning approach, we conducted extensive experiments using the StarCraft II environment. In this section, we describe our experimental setup, model configurations, and performance results.

4.1. Performance Against Built-in AI

We evaluated our adapter models at different training epochs against the built-in Level 10 AI of StarCraft II, which represents a strong baseline opponent. Table 2 presents the win rates across configurationsD.2 and training epochs. For brevity, we use configurations to refer to our different adapter models, and abbreviate epoch counts (e.g., 1K = 1000).

These results reveal an interesting pattern: all models exhibit high win rates in early training epochs but show declining

Config	Epoch 1K	Epoch 5K	Epoch 7K
A	80.65%	31.25%	14.29%
B	83.00%	36.00%	14.00%
C	87.00%	27.00%	18.00%
D	89.00%	34.00%	–

Table 2. Win rates against built-in Level 10 AI across training epochs

performance as training progresses. This behavior suggests that the adapter models initially maintain the strong core capabilities of the base DI-Star policy but gradually specialize in particular tactical styles at the expense of general performance against the built-in AI. The superior performance of Config B, C, and D at Epoch 1000 (compared to the base DI-Star) indicates that tactical conditioning may initially enhance performance by providing more structured decision-making. The subsequent decline likely reflects the growing influence of tactical specialization. Of particular note is Configuration C, which maintains the highest win rate at Epoch 7000, suggesting that balanced but higher constraints across tactical action heads may promote more effective tactical adaptation with less performance degradation against general opponents.

4.2. Case Studies

To provide a deeper understanding of how our tactical adaptation approach influences gameplay, we present two interesting case studies that emerged during training. These cases demonstrate the model's ability to discover rare but effective tactical variations that are not commonly observed in the base model. One of the most interesting tactical adaptations emerged in Configuration C around Epoch 5000. When conditioned with a tactical tensor emphasizing Categories 5 (+1 Zergling Strategy) and 7 (Lurker Transition Strategy), the model discovered an unusual but highly effective tactical transition, as shown in Figure 2(a). The adapted policy learned to leverage the early economic advantage of the +1 Zergling strategy while simultaneously tech-transitioning toward Lurkers—a hybrid approach rarely seen in standard play due to contradictory investment directions. This tactical adaptation proved highly effective against the built-in Level 10 AI, achieving a 73% win rate in dedicated test matches, demonstrating how our approach can discover novel tactical combinations by blending elements from different strategic archetypes. Another notable case emerged in Configuration B at Epoch 5000, where the model developed an unusual defensive response to early rush tactics, as illustrated in Figure 2(b). When conditioned with tactical tensors emphasizing Category 3 (Economic Three Base Strategy), the adapted policy learned to construct an unusually high number of Spine Crawlers in specific defensive positions. This adaptation is particularly interesting because extensive static defense is generally considered suboptimal in high-level Zerg gameplay, yet the adapted policy discovered a specific positioning pattern that effectively countered rush tactics while minimizing economic impact. These case studies illustrate how our tactical adaptation approach enables the discovery of specialized strategic responses not prominent in the base model's behavioral repertoire. Tactical conditioning allows the model to explore strategic subspaces underrepresented in the original training distribution, while KL divergence constraints provide sufficient flexibility for innovation while maintaining core competencies. Different adapter configurations explore different regions of the tactical space, with Configuration B favoring defensive innovations and Configuration C excelling at tech transitions. These observations highlight adapter-based tactical conditioning's potential for both strategy customization and discovering novel approaches.

(a) +1 Zergling to Lurker Transition

(b) Defensive Spine Crawler Positioning

Figure 2. Emergent tactical adaptations: (a) The model discovered an unusual but effective transition from +1 Zergling to Lurker technology, creating a powerful mid-game timing attack. (b) The model learned to construct multiple Spine Crawlers in strategic defensive positions to counter early rush tactics.

5. Acknowledgments

We extend our heartfelt gratitude to the **StarCraft 2 AI community** (https://discord.com/invite/BH58ZVt), **Spawningtool** (https://lotv.spawningtool.com), and **Liquipedia-sc2** (https://liquipedia.net/starcraft2/Main_Page) for providing rich data sources and granting permission for their use.

We are deeply grateful to **IA** (https://github.com/upia99) and **DI-Star Team** for providing detailed guidance and invaluable assistance. This project would not have been possible without their support. We also thank **Hongsheng Yu** and **the ROA-Star Team** for their guidance and help throughout this endeavor.

6. Impact Statement

This work advances the field of Game AI decision-making through the lens of real-time strategy games. Our framework primarily serves as a research tool for studying AI capabilities in controlled game environments, with minimal risk of direct negative societal impact.

References

Deng, Y., Ma, W., Fan, Y., Zhang, Y., Zhang, H., and Zhao, J. A new approach to solving smac task: Generating decision tree code from large language models, 2024. URL https://arxiv.org/abs/2410.16024. 10

Han, L., Xiong, J., Sun, P., Sun, X., Fang, M., Guo, Q., Chen, Q., Shi, T., Yu, H., Wu, X., et al. Tstarbot-x: An open-sourced and comprehensive study for efficient league training in starcraft ii full game. *arXiv preprint arXiv:2011.13729*, 2020. 1, 10

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019. 10

Huang, R., Wu, X., Yu, H., Fan, Z., Fu, H., FU, Q., and Wei, Y. A robust and opponent-aware league training method for starcraft ii. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 10

Li, Z., Ni, Y., Qi, R., Jiang, L., Lu, C., Xu, X., Liu, X., Li, P., Guo, Y., Ma, Z., et al. Llm-pysc2: Starcraft ii learning environment for large language models. *arXiv preprint arXiv:2411.05348*, 2024. 10

Liu, R.-Z., Guo, H., Ji, X., Yu, Y., Pang, Z.-J., Xiao, Z., Wu, Y., and Lu, T. Efficient reinforcement learning for starcraft by abstract forward models and transfer learning. *IEEE Transactions on Games*, 14(2):294–307, 2021a. 10

Liu, R.-Z., Wang, W., Shen, Y., Li, Z., Yu, Y., and Lu, T. An introduction of mini-alphastar. *arXiv preprint arXiv:2104.06890*, 2021b. 10

Liu, R.-Z., Pang, Z.-J., Meng, Z.-Y., Wang, W., Yu, Y., and Lu, T. On efficient reinforcement learning for full-length game of starcraft ii. *Journal of Artificial Intelligence Research*, 75:213–260, 2022. 10

Ma, W., Xu, D., Lin, S., Zhang, H., and Wang, J. Adaptive command: Real-time policy adjustment via language models in starcraft ii. 10

Ma, W., Mi, Q., Zeng, Y., Yan, X., Wu, Y., Lin, R., Zhang, H., and Wang, J. Large language models play starcraft ii: Benchmarks and a chain of summarization approach, 2024. URL https://arxiv.org/abs/2312.11865. 10

Ma, W., Fu, Y., Zhang, Z., Ghanem, B., and Li, G. Ava: Attentive vlm agent for mastering starcraft ii, 2025. URL https://arxiv.org/abs/2503.05383. 10

Mathieu, M., Ozair, S., Srinivasan, S., Gulcehre, C., Zhang, S., Jiang, R., Le Paine, T., Zolna, K., Powell, R., Schrittwieser, J., et al. Starcraft ii unplugged: Large scale offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021. 10

Peng, P., Wen, Y., Yang, Y., Yuan, Q., Tang, Z., Long, H., and Wang, J. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017. 10

star Contributors, D. Di-star: An open-sourse reinforcement learning framework for starcraftii. https://github.com/opendilab/DI-star, 2021. 1

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017. 1, 10

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019. 1, 10

Wang, X., Song, J., Qi, P., Peng, P., Tang, Z., Zhang, W., Li, W., Pi, X., He, J., Gao, C., et al. Scc: An efficient deep reinforcement learning agent mastering the game of starcraft ii. In *International conference on machine learning*, pp. 10905–10915. PMLR, 2021. 10

Zhang, L., Rao, A., and Agrawala, M. Adding conditional control to text-to-image diffusion models. 1, 10

A. DI-Star original architecture

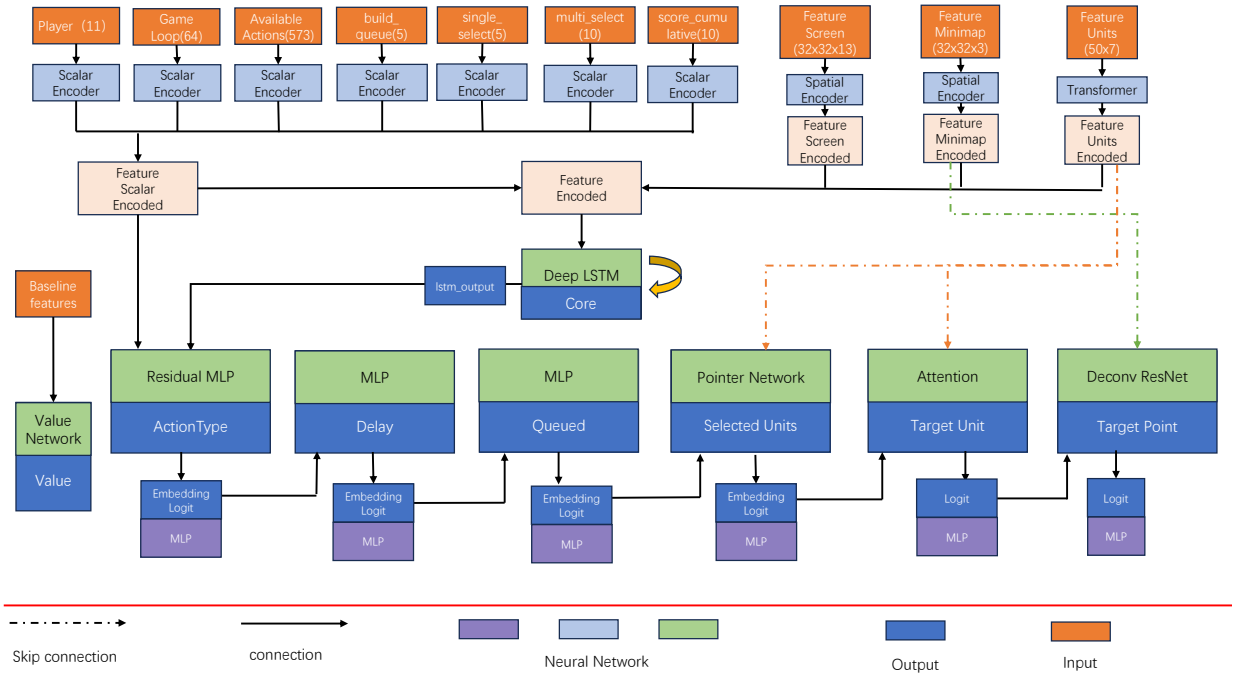

Figure 3. DI-star original architecture.

B. DI-Star Network Architecture Details

The DI-Star architecture consists of three main components: observation encoders, temporal processing (LSTM), and action heads. This appendix provides a detailed overview of these components.

B.1. Observation Encoders

DI-Star employs three specialized encoders to process the complex multi-modal observations from StarCraft II:

B.1.1. Entity Encoder

The Entity Encoder processes information about units, buildings, and other game entities. Its architecture includes:

- Multi-modal feature encoding for entity attributes (unit type, health, shields, etc.)
- A Transformer module to process relationships between entities
- Output projections that produce per-entity embeddings and an aggregated entity representation

The encoder handles variable numbers of entities through masked attention and supports different aggregation methods including attention pooling and entity-count-based averaging.

B.1.2. Scalar Encoder

The Scalar Encoder processes non-spatial numerical features such as:

- Resource counts (minerals, vespene gas)
- Game timing information

- Population statistics (current supply, maximum supply)
- Economic metrics (worker count, collection rates)
- Beginning build order sequences

This encoder employs a mix of one-hot encoding, positional encoding for temporal features, and specialized neural networks like the BeginningBuildOrderEncoder which uses a Transformer to encode the initial build sequence.

B.1.3. Spatial Encoder

The Spatial Encoder processes the 2D grid representation of the StarCraft II map, including:

- Terrain features (height maps, pathing grid)
- Unit positions and densities
- Visibility information (fog of war)

Its architecture includes:

- Initial projection layers
- Downsampling modules (using either convolutional layers or pooling operations)
- Residual blocks for feature extraction
- Skip connections that preserve spatial information for the Location action head

B.2. Temporal Processing (LSTM)

After feature extraction by the encoders, the features are concatenated and processed by a Layer-Normalized LSTM network to capture temporal dependencies:

- Multi-layer design with layer normalization for stable training
- Hidden state dimension of 384
- State persistence across time steps to maintain game state understanding

The LSTM enables the policy to make consistent decisions over time and learn patterns in the opponent's behavior and game state progressions.

B.3. Action Heads

The policy network's output is structured as multiple specialized heads for StarCraft II's complex action space:

B.3.1. Action Type Head

Selects from approximately 50 different high-level action categories (attack, build, research, etc.). Architecturally, it consists of:

- Multiple fully-connected layers with gated linear units
- An attention mechanism that integrates scalar context with LSTM output
- Output logits representing the probability distribution over action types

B.3.2. Target Unit Head

Identifies specific units or buildings as targets. Its structure includes:

- Query-based attention mechanism that computes compatibility between the action embedding and entity embeddings
- Additional processing to handle invalid targets based on the selected action type
- Output logits representing the probability distribution over potential target units

B.3.3. Location Head

Determines spatial coordinates for actions on the game map. Its architecture includes:

- Upsampling layers that progressively increase spatial resolution
- Skip connections from the spatial encoder to maintain spatial fidelity
- Convolutional layers that transform the embedding into a spatial probability map
- Output logits representing a 152×160 grid of potential target locations

B.3.4. Selected Units Head

Controls which friendly units to select for executing commands. It consists of:

- Attention-based selection mechanism between the action embedding and entity embeddings
- Handling of selection constraints based on the action type (e.g., some actions require specific unit types)
- Sequential selection protocol for multi-unit selections
- Output logits representing the probability distribution over selectable units

B.3.5. Auxiliary Heads (Delay and Queued)

These heads handle action timing and sequencing:

- **Delay Head**: Produces a discrete probability distribution over possible delay durations between consecutive actions
- **Queued Head**: Outputs a binary decision on whether the current action should be queued after current unit tasks

B.4. Network Integration

The DI-Star architecture integrates these components in a sequential processing pipeline:

1. Observation encoders process the raw game state into feature embeddings
2. LSTM integrates these features with temporal context
3. Action heads transform the LSTM output into structured action distributions

This modular design allows each component to specialize in its domain while maintaining an integrated decision-making process that captures the complexity of StarCraft II gameplay.

C. Related Work

C.1. StarCraft II Full Game AI

StarCraft AI research, initially focused on StarCraft I with developments like BiCNet (Peng et al., 2017) for multi-agent coordination, has significantly advanced in the StarCraft II era. The release of PySC2 (Vinyals et al., 2017) by DeepMind, coupled with Blizzard's game replays, propelled this research field. A key breakthrough was AlphaStar (Vinyals et al., 2019), which achieved Grandmaster level and defeated top players, demonstrating the potential of RL in complex environments.

Subsequent research expanded upon these foundations. Mini-AlphaStar (Liu et al., 2021b) simplified input variables without compromising learning effectiveness. TG (Liu et al., 2021a) and HierNet-SC2 (Liu et al., 2022) explored efficient RL strategies, with the latter bypassing supervised pre-training. AlphaStar Unplugged (Mathieu et al., 2021) represented a leap in offline RL using human replays. TStarBotsX (Han et al., 2020) and SCC (Wang et al., 2021) furthered federated learning approaches, achieving notable success against master and grandmaster level players.

Recent advancements include DI-star[3], which is accessible for home computer deployment, and ROA-Star (Huang et al., 2023), enhancing AlphaStar's training framework with goal-conditioned exploiters and refined opponent modeling techniques. ROA-Star's practical tests against professional players have shown impressive results, marking significant progress in real-time strategy AI.

C.2. Language Models for StarCraft II

The integration of language models with StarCraft II represents an emerging research direction that seeks to bridge natural language understanding with complex game control. This research broadly falls into two categories: using language as a high-level control interface and leveraging language models for tactical reasoning.

TextStarCraft2 (Ma et al., 2024) pioneered this integration by creating a textual representation of the StarCraft II environment, enabling language models to make high-level strategic decisions through natural language interaction. This approach demonstrated that language representations could capture essential game state information sufficiently for strategic planning.

Several approaches have focused on micromanagement tasks within StarCraft II. LLM-PySC2 (Li et al., 2024) leveraged the PySC2 framework to create a specialized micromanagement environment where language models generate action commands. Similarly, LLM-SMAC and SMAC-R1 (Deng et al., 2024) explored code generation approaches where language models produce decision-making code to address specific combat scenarios in the StarCraft Multi-Agent Challenge environment.

Adaptive Command (Ma et al.) took a different approach by using language models to adjust behavior trees in response to human commands, enabling dynamic adaptation of pre-defined strategies based on natural language input. This work demonstrated the potential for language to serve as an intuitive interface for real-time strategy modification.

More recently, AVA (Ma et al., 2025) extended these efforts into multimodal learning by combining PySC2 with vision-language models to solve micromanagement tasks through visual reasoning and natural language generation. This approach highlights the potential for multimodal understanding in complex game environments.

While these approaches have made significant progress in using language for specific aspects of StarCraft II gameplay, they have primarily focused on either high-level strategic planning or isolated micromanagement scenarios. Our work differs by focusing on tactical conditioning of a full-game policy network, enabling fine-grained control over gameplay style while maintaining competitive performance across all aspects of the game.

C.3. Parameter-Efficient Transfer Learning

Our approach draws inspiration from parameter-efficient transfer learning methods that have shown success in other domains. Adapter-based fine-tuning, first proposed for natural language processing tasks (Houlsby et al., 2019), has become a popular approach for adapting pre-trained models to new tasks without modifying the original parameters.

In computer vision, ControlNet (Zhang et al.) demonstrated how conditional control could be added to diffusion models through lightweight adapter modules, enabling precise control over image generation while preserving the capabilities of the base model. Our approach applies similar principles to reinforcement learning policies, introducing tactical conditioning through adapters while freezing the pre-trained policy network.

[3] https://github.com/opendilab/DI-star

This parameter-efficient approach offers several advantages: it preserves the fundamental capabilities of the original agent, reduces training costs compared to full model fine-tuning, and enables modular control over different aspects of the policy. While adapter-based methods have been extensively studied in language and vision domains, their application to complex reinforcement learning policies—particularly in the context of real-time strategy games—represents a novel contribution of our work.

D. Experimental Setup

We implemented our adapter architecture on top of the DI-Star model, using the pre-trained policy network as our base. All experiments focused on the Zerg vs. Zerg matchup, which offers a rich tactical landscape while maintaining a controlled experimental environment. Our training setup used the following configuration:

```
feature:
  filter_spine: True
  zero_z_value: 1.
  beginning_order_prob: 0.8  # probability of using building order in SL training,
  cumulative_stat_prob: 0.5  # probability of using cumulative statistics in RL training,
  zergling_num: 8
learner:

  job_type: 'train'
  adaptor_tuning: true
  use_cuda: True
  use_distributed: False
  use_value_feature: False
  learning_rate: 0.001
  weight_decay: 0.0001
  load_optimizer: True
  load_last_iter: True
  lr_decay: 1.0
  lr_decay_interval: 10000
  use_warmup: True
  warm_up_steps: 20000
  steps: 100000
  su_mask: False
  use_dapo: False
  grad_clip:
    type: 'momentum_norm'
    threshold: 1.4
  data:
    remote: False
    replay_actor_num_workers: 64
    filter_action: False
    parse_race: ['Z']
    epochs: 2
    batch_size: 128
    trajectory_length: 8
    num_workers: 256
    timeout: 3600
  loss_weight:
    action_type: 30.0
    delay: 9.0
    queued: 1.0
    selected_units: 4.0
    select_unit_num_logits: 8.0
    target_unit: 4.0
    target_location: 8.0
  hook:
    save_ckpt_after_iter:
      ext_args:
        freq: 1000
  adaptor:
```

```
    active_adaptors: ["action_type", "delay", "queued", "selected_units", "target_unit", "location", "ls
    fusion_method: "add"
    tactic_dim: 9
```

The `tactic_dim` parameter of 9 corresponds to the tactical categories described in Section 2, which define our tactical conditioning space. We trained our models for 100,000 steps with warmup for the first 20,000 steps, using Adam optimizer with learning rate 0.001 and weight decay 1e-5.

D.1. Tactical Categories

Our tactical conditioning is based on 9 strategic archetypes collected in Section 2:

1. **Unclassified Strategy**: Build orders that are too short or don't fit other categories. Characterized by insufficient information or unclear strategic direction.
2. **Standard Roach-based Strategy**: Focuses on Roach production with critical upgrades including Roach Warren, Evolution Chamber (for +1 missile attacks), Lair, and Roach Speed. Characterized by upgraded Roaches as main army composition, strong timing attacks, and usually 2-base economy.
3. **Spire-based Strategy**: Transitions into Mutalisk production and air control. Key buildings include Lair, Spire, and multiple gas extractors. Characterized by gas-heavy economy, map control orientation, and tech transition into air dominance.
4. **Economic Three Base Strategy**: Emphasizes fast three base saturation with quick third hatchery and safety structures. Characterized by heavy drone production, defensive early game posture, and maximum economy focus.
5. **Early Pool Aggression**: Features early Zergling pressure with early Spawning Pool (12 pool or earlier) and limited economy buildings. Characterized by sacrificed economy for early pressure and heavy Zergling production.
6. **+1 Zergling Strategy**: Focuses on melee upgrade with mass Zergling production, with early Evolution Chamber and +1 Melee Attacks. Characterized by timing attacks with upgraded Zerglings and moderate economy into strong timing.
7. **Nydus-based Strategy**: Utilizes Nydus Network surprise attacks with Lair, Nydus Network, and Nydus Worms. Characterized by map mobility focus, surprise attacks, and specific army compositions.
8. **Lurker Transition Strategy**: Centers on Lurker-based map control with Lair, Hydralisk Den, and Lurker Den. Characterized by mid-game tech transition, strong map control, and siege capability.
9. **Early/Mid-early Game All-in Strategy**: Features mass production of basic units with 2-3 bases and relevant production structures without Lair tech. Characterized by heavy unit production, committed aggression, and limited tech investment.

These tactical categories were derived from our corpus analysis as described in and validated by Grandmaster-level players to ensure they represent the strategic diversity present in high-level StarCraft II gameplay.

D.2. Model Configurations

We tested several model configurations with different KL divergence weight settings for the action heads to investigate the impact of varying constraints on tactical adaptation. These weights control the strength of the constraint that ties the adapted policy to the original policy, with higher values enforcing stronger adherence to the base policy.

D.2.1. Configuration A

This configuration applies uniform constraints across all action heads:

```
head_weights_dict = {
    'action_type': 1.0,
    'delay': 1.0,
    'queued': 1.0,
```

```
    'selected_units': 1.0,
    'target_unit': 1.0,
    'target_location': 1.0
}
```

D.2.2. CONFIGURATION B

This configuration varies constraints across action heads, with lower constraints on spatial decision-making:

```
head_weights_dict = {
    'action_type': 10.0,
    'delay': 1.0,
    'queued': 10.0,
    'selected_units': 3.0,
    'target_unit': 1.0,
    'target_location': 0.0
}
```

D.2.3. CONFIGURATION C

This configuration applies higher uniform constraints across tactical action heads:

```
head_weights_dict = {
    'action_type': 10.0,
    'delay': 1.0,
    'queued': 10.0,
    'selected_units': 10.0,
    'target_unit': 10.0,
    'target_location': 10.0
}
```

D.2.4. CONFIGURATION D

This configuration represents maximum constraints across all action heads:

```
head_weights_dict = {
    'action_type': 100.0,
    'delay': 100.0,
    'queued': 100.0,
    'selected_units': 100.0,
    'target_unit': 100.0,
    'target_location': 100.0
}
```

E. Discussion

Our experimental results support three key findings:

1. **Tactical Specialization Trade-off**: Higher degrees of tactical adaptation (later epochs) come at the cost of decreased general performance against fixed opponents like the built-in AI. This suggests a natural trade-off between tactical specialization and general capability.

2. **Constraint Balancing**: The choice of KL divergence weights significantly impacts both the degree of tactical adaptation and the preservation of base capabilities. Configuration C's balanced higher constraints appear to offer a favorable trade-off.

3. **Adaptation Progression**: The complete dominance of later epoch models over earlier ones in head-to-head competition confirms that meaningful tactical adaptations continue to develop throughout training.

These findings highlight the importance of carefully balancing tactical flexibility and core competency preservation when implementing adapter-based tactical conditioning for complex game AI systems. The optimal configuration depends on

the specific application goals—whether prioritizing tactical diversity, competitive performance, or adaptation to specific opponent styles.