POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

An LLM-Driven Agent for Competitive Pokémon Battling with Bayesian Opponent Modeling

Tesi di Laurea Magistrale in
Computer Science and Engineering

Author: **Alessio De Vito**

Student ID: 11006752
Advisor: Prof. Alberto Marchesi
Co-advisors: Gianmarco Genalti, Pierriccardo Olivieri
Academic Year: 2025-26

Abstract

Competitive Pokémon is a demanding testbed for artificial intelligence: it combines the long decision horizons of board games with the hidden information, opponent modeling, and stochasticity of card games, all layered over an encyclopedic set of species, moves, abilities, and items. Recent work has explored two broad strategies for the game — guiding tree search with a large language model (LLM), and learning to play through model-free offline reinforcement learning — yet turn-by-turn LLM agents remain limited by token-heavy prompts, weak opponent modeling, and brittle hand-crafted heuristics.

This thesis presents *PolimiBot*, an LLM-driven agent for Generation 9 OverUsed singles battles on Pokémon Showdown, together with a *Bayesian Predictor* that infers hidden opponent information from statistical models trained on roughly 140,000 historical Generation 9 OU teams. Both are designed as direct responses to concrete weaknesses of PokéChamp, the strongest existing LLM-based agent. PolimiBot encodes the current battle state exhaustively in structured blocks — grounding the LLM in exact turns-to-KO figures from a canonical damage calculator — and decomposes each decision into a three-prompt flow (move selection, switch selection, and a numerically anchored merger) rather than a single conflated call. The Bayesian Predictor fuses per-species replay frequencies with metagame usage statistics, models team composition with a latent-archetype topic model, and conditions its estimates on evidence observed during the battle, supplying the agent with a complete view of the opponent from the first turn.

The two components are evaluated against a heuristic baseline (AbyssalPlayer) and the PokéChamp baseline across five teams of contrasting styles. PolimiBot wins 89% of games against the heuristic baseline and 68% against the LLM baseline, while reducing prompt size by 19% relative to PokéChamp. The predictor's confidence scores are shown to be well calibrated against accuracy, so that a high-confidence prediction holds roughly nine times in ten and can be relied on by the agent. Taken together, the results indicate that careful state encoding, numerical grounding, and calibrated opponent modeling let a scaffolded LLM agent outplay a strong LLM baseline; each gain, and its limitations, is discussed in relation to the specific weakness the design set out to address.

Keywords: competitive Pokémon; large language models; game-playing agents; Bayesian opponent modeling; Pokémon Showdown

Abstract in lingua italiana

Il Pokémon competitivo costituisce un banco di prova impegnativo per l'intelligenza artificiale: combina i lunghi orizzonti decisionali dei giochi da tavolo con l'informazione nascosta, la modellazione dell'avversario e la stocasticità dei giochi di carte, il tutto sovrapposto a un vastissimo repertorio di specie, mosse, abilità e strumenti. Lavori recenti hanno esplorato due strategie principali per affrontare il gioco — guidare la ricerca ad albero tramite un modello linguistico di grandi dimensioni (LLM) e imparare a giocare mediante apprendimento per rinforzo offline e senza modello (*model free*) — ma gli agenti LLM che decidono turno per turno restano limitati da prompt molto onerosi in termini di token, da una debole modellazione dell'avversario e da euristiche fragili costruite manualmente.

Questa tesi presenta *PolimiBot*, un agente guidato da un LLM per le battaglie singole nel formato Generazione 9 OverUsed su Pokémon Showdown, insieme a un *Predittore Bayesiano* che inferisce le informazioni nascoste dell'avversario a partire da modelli statistici addestrati su circa 140.000 squadre storiche di Generazione 9 OU. Entrambi sono progettati come risposte dirette a debolezze concrete di PokéChamp, il più avanzato agente basato su LLM esistente. PolimiBot codifica in modo esaustivo lo stato corrente della battaglia in blocchi strutturati — fornendo all'LLM i valori esatti del numero di turni necessari a mettere fuori combattimento un Pokémon, calcolati da un calcolatore di danni — e scompone ogni decisione in un flusso a tre prompt (selezione della mossa, selezione del cambio e una fusione guidata da valori numerici) anziché in un'unica chiamata indistinta. Il Predittore Bayesiano fonde le frequenze osservate nei replay, specifiche per ciascuna specie, con le statistiche d'uso del metagame, modella la composizione delle squadre tramite un topic model ad archetipi latenti e aggiorna le proprie stime in base alle informazioni osservate durante la battaglia, fornendo all'agente una visione completa dell'avversario fin dal primo turno.

I due componenti sono valutati rispetto a un riferimento euristico (AbyssalPlayer) e a PokéChamp, su cinque squadre dagli stili molto diversi. PolimiBot vince l'89% delle partite contro il riferimento euristico e il 68% contro quello basato su LLM, riducendo al contempo del 19% la dimensione dei prompt rispetto a PokéChamp. Si dimostra inoltre che i punteggi di confidenza del predittore sono ben calibrati rispetto all'accuratezza, cos-

icché una predizione ad alta confidenza risulta corretta circa nove volte su dieci e l'agente può farvi pieno affidamento. Nel complesso, i risultati indicano che una codifica accurata dello stato, solide basi numeriche e una modellazione calibrata dell'avversario consentono a un agente LLM opportunamente strutturato di superare un solido riferimento basato su LLM; ciascun miglioramento, con i suoi limiti, viene discusso in relazione alla specifica debolezza a cui risponde.

Parole chiave: Pokémon competitivo; modelli linguistici di grandi dimensioni; agenti di gioco; modellazione bayesiana dell'avversario; Pokémon Showdown

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

1 Introduction **1**
1.1 Why competitive Pokémon 1
1.2 Contributions 2
1.3 Outline 3

2 Background **5**
2.1 Competitive Pokémon battles 5
2.1.1 Structure of a battle 5
2.1.2 Core mechanics 6
2.1.3 Team building and hidden information 6
2.1.4 Randomness and scale 6
2.2 Pokémon Showdown and competitive formats 7
2.2.1 Generation 9 OverUsed 7
2.3 A game-theoretic formulation 8
2.4 Large language models as game-playing agents 9
2.4.1 A reference architecture 9
2.4.2 Constraining the action space 10
2.5 Evaluation metrics 10

3 Related Work **13**
3.1 Games solved by artificial intelligence 13
3.2 Language agents and the prompting toolbox 13
3.3 LLM agents across strategic games 14

3.3.1 Open-ended and real-time environments . . . 14
3.3.2 Hidden roles and negotiation . . . 15
3.3.3 Search and fine-tuning for strategic competence . . . 16
3.4 Pokémon-playing agents . . . 16
3.4.1 Foul Play: heuristic search . . . 17
3.4.2 PokéChamp: an LLM-guided minimax agent . . . 18
3.4.3 Offline reinforcement learning on archived games . . . 19

4 Problem and Motivation **21**
4.1 Prompt representation and efficiency . . . 21
4.2 Modeling the hidden opponent . . . 22
4.3 Decision structure . . . 22
4.4 Numerical grounding . . . 23

5 Methodology **25**
5.1 Design philosophy . . . 25
5.2 PolimiBot: the battle agent . . . 26
5.2.1 What the agent knows each turn . . . 26
5.2.2 The three-prompt decision flow . . . 27
5.2.3 Majority voting . . . 29
5.2.4 Damage-evidence tracking . . . 29
5.3 The Bayesian Predictor . . . 29
5.3.1 Model hierarchy . . . 30
5.3.2 Per-species frequency tables . . . 30
5.3.3 Smogon usage prior and adaptive fusion . . . 31
5.3.4 Team composition model . . . 34
5.3.5 Evidence conditioning . . . 39
5.3.6 Backward EV inference from damage . . . 39
5.3.7 The per-turn prediction pipeline . . . 40
5.4 Integrating the predictor into the agent . . . 40

6 Experiments and Evaluation **41**
6.1 Experimental setup . . . 41
6.2 Bayesian Predictor accuracy . . . 42
6.2.1 A caveat on benchmark validity . . . 42
6.2.2 Why the redesign is expected to generalize better . . . 43
6.2.3 Species prediction . . . 43
6.2.4 Move prediction . . . 43

6.2.5 Component prediction 44
6.2.6 Confidence calibration 44
6.2.7 Calibration validated on genuine ground truth 46
6.2.8 Evidence conditioning sharpens predictions 48
6.3 Token-consumption efficiency 49
6.4 Battle results against AbyssalPlayer 50
6.5 Battle results against PokéChamp 51
6.6 Discussion 53

7 Conclusions and Future Work **57**
7.1 Summary of contributions 57
7.2 Limitations 57
7.3 Future work 58

Bibliography **61**

A Supporting Material **67**
A.1 PokéChamp weaknesses and PolimiBot improvements 67
A.2 Complete per-matchup battle results 68

B Complexity of Single Battles **71**
B.1 Per-turn branching factor 71
B.2 Size of a player's information set 72
B.3 Number of team configurations 72

List of Figures **75**

List of Tables **79**

1 | Introduction

Reinforcement learning has produced superhuman play in a remarkable range of games. Chess fell first to handcrafted search and later to self-play [4, 26]; Go was mastered by deep networks paired with tree search [25]; and the same broad recipe reached grandmaster level in StarCraft II [28], defeated professionals in Dota 2 [1], and outraced world-class human drivers in Gran Turismo [34]. These systems are extraordinary, but they share a cost: each demands heavy, task-specific training and engineering, and the resulting policy transfers poorly to anything but the game it was trained on.

Large language models (LLMs) sit at the opposite end of this trade-off. They are generalists that carry broad prior knowledge and transfer to new situations without additional training, and a growing body of work equips them with prompting and planning techniques — chain-of-thought [33], self-consistency [32], tree-of-thoughts [38], and reason-and-act loops [39], among others — to elicit more deliberate reasoning. Yet LLMs have well-documented weaknesses in planning: they can lose to simple hand-built heuristics and stumble over basic game mechanics. The open question this thesis engages with is whether the generalist knowledge of an LLM can be harnessed, with the right scaffolding, to play a genuinely hard game well.

1.1. Why competitive Pokémon

Competitive Pokémon is an unusually demanding testbed. A single match is really two linked problems: assembling a team of six Pokémon beforehand, then piloting it in battle. The game brings together properties that are usually studied separately — the long decision horizons of board games, the hidden information and opponent modeling of card games, and stochastic transitions — and layers them over an encyclopedic vocabulary of species, moves, abilities, items, and special-case rules. Battles are partially observable, decisions on each turn are made simultaneously, the state space is astronomically large, and match length varies from a handful of turns to well over a hundred. In modern formats, players must also act under a strict time budget. The game can thus be read as combining chess-like horizons with poker-like hidden information and randomness, which

makes it an attractive and discriminating benchmark for reasoning under uncertainty.

The combinatorial scale of these abstractions is worth making concrete. Even before the first move, the number of legal six-Pokémon configurations a player may bring exceeds 10^{139}; within a single turn the branching factor — the joint choice of both players over moves, switches, and Terastallization — is on the order of 10^5; and the uncertainty a player holds about the unobserved half of the game, the opponent's hidden team, spans an information set of roughly 10^{58} states (the derivations are given in Chapter B). Numbers of this magnitude rule out exhaustive search outright, and they generate a diversity of situations that no fixed table of heuristics can anticipate. An agent must instead reason selectively, focusing on the few lines that matter from a space it can never enumerate.

Hidden information is not incidental to this difficulty but central to it. At the start of a battle each side sees only the species on the opposing team, through the team-preview screen, while every move, item, ability, defensive investment, and Tera type stays concealed until it is used or otherwise revealed in play. Strong human players close this gap by inference: they read an opponent's likely sets from the prevailing metagame, narrow the possibilities as evidence accumulates, and plan against the most probable threats rather than the merely possible ones. Reproducing this faculty — turning sparse public observations into a calibrated belief over the opponent's concealed choices — is one of the two problems this thesis sets out to solve, and it motivates the Bayesian Predictor described below.

Two contrasting strategies frame much of the recent work and serve as natural reference points. The first uses an LLM to guide tree search: the model proposes candidate actions, reasons about the opponent, and evaluates positions, so that prior knowledge narrows the search without any additional model training; the strongest published instance of this approach is PokéChamp [14]. The second dispenses with explicit search and instead trains sequence models with imitation and offline reinforcement learning, relying on long-context memory to infer the opponent's team and tendencies from a large archive of human matches [8]. Experience to date suggests the model-free, long-context route can reach strong human-level play, whereas turn-by-turn LLM agents tend to struggle, particularly over long horizons.

1.2. Contributions

This thesis develops an LLM-driven agent for Generation 9 OverUsed (OU) singles battles on Pokémon Showdown, and makes two contributions:

- **PolimiBot**, an agent that drives turn-by-turn battle decisions. Rather than prepending a long transcript of past turns to every prompt, PolimiBot encodes the *current* battle state exhaustively in structured blocks and grounds the LLM in exact turns-to-KO figures from a canonical damage calculator. It decomposes each decision into a three-prompt flow — move selection, switch selection, and a numerically anchored merger — in place of the single conflated call used by prior LLM agents.
- The **Bayesian Predictor**, which infers hidden opponent information (unrevealed moves, items, abilities, tera types, and bench Pokémon) from statistical models trained on roughly 140,000 Generation 9 OU teams. It fuses replay frequencies with metagame usage statistics, captures team composition with a latent-archetype topic model, and conditions its estimates on evidence observed during the battle. Its ranked predictions are injected into PolimiBot's prompts so the agent has a complete view of the opponent from the first turn.

Both components are designed in direct response to concrete weaknesses identified in PokéChamp, the strongest existing LLM-based agent. They are evaluated against a heuristic baseline (AbyssalPlayer) and against PokéChamp itself, across five teams of contrasting strategic styles, on win rate, prompt-token efficiency, and predictor accuracy.

1.3. Outline

Chapter 2 introduces how competitive Pokémon works, the Pokémon Showdown platform and its formats, a game-theoretic formulation of a battle, and the metrics used to evaluate agents. Chapter 3 surveys how other games have been solved by AI, the prompting and planning toolbox for LLMs, and the existing Pokémon agents most relevant to this work. Chapter 4 distils the concrete weaknesses of PokéChamp that motivate the design. Chapter 5 presents the system design of PolimiBot and the Bayesian Predictor. Chapter 6 reports the experimental evaluation, and Chapter 7 summarizes the findings, limitations, and directions for future work.

2 | Background

This chapter establishes the background needed to follow the rest of the thesis. It describes how competitive Pokémon works as a game, the Pokémon Showdown platform and its formats, a game-theoretic formulation of a battle, and the metrics commonly used to evaluate agents.

2.1. Competitive Pokémon battles

Competitive Pokémon Singles is a two-player game that combines several difficult properties at once: the long decision horizons usually associated with board games such as chess, alongside the hidden information, opponent modeling, and randomness characteristic of card games such as poker. On top of this it carries an unusually large vocabulary of species, moves, abilities, items, and special-case rules. Each match is really two linked problems — assembling a team beforehand, then piloting it in battle — and because outcomes hinge on reasoning under uncertainty across a very large space of possibilities, the game is an attractive testbed for reinforcement learning and game-theoretic methods.

2.1.1. Structure of a battle

Each player brings a team of up to six Pokémon, and the two sides take turns. In the Singles format only one Pokémon per side is on the field at any moment. The goal is to make every one of the opponent's Pokémon faint — reduce its hit points (HP) to zero — and the side that loses all six loses the match. On a given turn a player either uses one of the active Pokémon's moves, switches in a different team member, or uses an item. Although turns alternate, both players commit to their action for a turn at the same time, without seeing the other's choice, so each turn behaves more like a simultaneous-move game than strict alternation. Match length varies widely, from roughly a dozen turns to well over a hundred, which is part of why brute-force search over the full game tree is infeasible.

2.1.2. Core mechanics

Much of the game's depth comes from the interaction of a few systems. Every Pokémon and every move carries one or two elemental *types*, and an elaborate table of type interactions sets a damage multiplier (super-effective, resisted, or immune) that feeds directly into damage calculation. Six *statistics* — HP, Attack, Defense, Special Attack, Special Defense, and Speed — govern combat performance. *Abilities* are passive traits that grant a variety of in-battle effects, and *held items* are objects carried into battle that confer bonuses or special effects. Broad fundamentals carry across game versions, but expert play turns on a long tail of corner cases that differ from one version to the next.

2.1.3. Team building and hidden information

Before a match, a player assembles a team. For each of the six Pokémon they decide on its moves, its ability, its held item, its nature, and an allocation of Effort Values (EVs) and Individual Values (IVs) that fixes the final statistics. This pre-match design stage is itself strategic: choices are made to anticipate and counter the threats a player expects to meet. Although the raw number of possible teams is astronomically large, the set of teams that are genuinely competitive narrows, in practice, to something on the order of thousands.

Battles are partially observable. A player sees only the opponent's currently active Pokémon, and the precise stats, ability, and moveset of any opposing Pokémon stay hidden until they appear in play. The result is an asymmetry — full knowledge of one's own team, partial knowledge of the opponent's — so a single observation is consistent with many true states, and choices must be made under uncertainty. Reading the hidden part of the opponent's team is where much of the skill lies: regularities in how teams are built — which Pokémon tend to appear together, and which movesets are common or mutually exclusive — let a player narrow down what has not yet been revealed. Because turns are resolved simultaneously, anticipating the opponent's next choice from their revealed team and past habits is decisive. Strong play therefore amounts to maintaining and continually revising a belief about the opponent's team and intentions. This observation is the central motivation for the Bayesian Predictor of Chapter 5.

2.1.4. Randomness and scale

State transitions are stochastic — move accuracy and the spread in damage rolls (typically between 85% and 100% of the nominal value) are two sources — and the reward is binary:

a win or a loss. With more than a thousand species, each carrying its own types, stats, abilities, moves, and items, the number of distinct situations is enormous. For the singles format studied in this thesis, three figures capture the scale, each derived in Chapter B: there are on the order of 10^{139} legal team configurations; on a given turn a single player's information set — the set of true states left indistinguishable by partial observability, owing chiefly to the opponent's hidden statistical investment — spans on the order of 10^{58} states; and the worst-case per-turn branching factor, combining each side's simultaneous choice of move, switch, or Terastallization with the stochastic resolution of a move, is on the order of 10^{5}. This vast space persists alongside partial observability for the whole game. Generation 9 further adds *Terastallization*, which lets a Pokémon switch to a new type once per battle and can upend type matchups mid-game; since the chosen type can be any of the eighteen elemental types or the special Stellar type, this sharply enlarges the state and action spaces and makes the opponent harder to model.

2.2. Pokémon Showdown and competitive formats

Most competitive play happens on Pokémon Showdown, a browser-based simulator that reproduces the official battle mechanics and supports many formats. Because it removes the need to breed or train Pokémon in the retail games, it allows fast experimentation and is widely used both by the competitive community and by researchers studying AI in complex games. It draws thousands of players daily, mirrors the mechanics of each major retail release, and — usefully for data-driven methods — has archived turn-by-turn records of matches for well over a decade.

Showdown supports several formats. *Singles* has one active Pokémon per side; *Doubles* has two. *Random Battles* hand each player a procedurally generated, fully-trained team, which removes the team-design problem and leaves only the in-battle decisions; this format is the usual default for AI experiments. *Tiered formats* sort Pokémon into tiers by usage and strength — for example OverUsed (OU), UnderUsed (UU), and RarelyUsed (RU).

2.2.1. Generation 9 OverUsed

OU is the flagship singles format in each generation: the most played, and consequently the one with the richest pool of recorded games. It bars a handful of dominant Pokémon while leaving a broad range of viable options, which keeps the metagame deep and in constant flux. Each "generation" corresponds to a major retail release whose mechanics Showdown reproduces, and in effect every tier-and-generation combination is its own game, with its own viable roster and metagame. A pivotal rule difference is *team preview*:

from the fifth generation onward, the full team of each player is shown at the start of the match, which reduces how much is hidden but leaves the precise sets — moves, items, abilities, spreads — still concealed. This thesis targets Generation 9 OU, the current flagship format, where Terastallization is in play and the metagame is well represented in recorded games.

Competitive matches run under a clock. A common arrangement gives each player a fixed budget for the whole game (for example, 150 seconds) plus a small increment per turn (for example, 15 seconds); exceeding either limit forfeits the match. Such limits reward quick decisions and rule out exhaustive search — a constraint that bears directly on the token-efficiency concerns of Chapter 4.

2.3. A game-theoretic formulation

Pokémon battles can be cast as a *partially observable Markov game* (POMG), given by a tuple $(\mathcal{S}, \mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{B}, H, P, r)$:

- $\mathcal{S}$ — the latent (true) state space.
- $\mathcal{X}, \mathcal{Y}$ — the observation spaces (information sets) of the two players; each contains only what that player can see.
- $\mathcal{A}, \mathcal{B}$ — their respective action spaces.
- H — the horizon.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{B} \to \Delta(\mathcal{S})$ — the transition kernel, with $P(s' \mid s, a, b)$ the probability of reaching s' when the players play a and b.
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{B} \to [0, 1]$ — the reward.

By convention one designates the learner the *max-player* (maximizing reward) and the opponent the *min-player*. A true state carries the full history together with complete information about both teams (HP, stats, items, status, and so on), whereas an observation carries the history plus full information about one's own team but only partial information about the opponent's — hence one observation maps to many states. The game has a tree structure (every state at step h is reached by a unique history) and perfect recall.

Writing the players' policies as $\mu : \mathcal{X} \to \Delta(\mathcal{A})$ and $\nu : \mathcal{Y} \to \Delta(\mathcal{B})$, the value of a policy pair is

$$V^{\mu,\nu} = \mathbb{E}_{\mu,\nu}\left[\sum_{h=1}^{H} r(s_h, a_h, b_h)\right]. \tag{2.1}$$

A *Nash equilibrium* is a pair (μ^*, ν^*) from which neither side gains by deviating unilaterally; the minimax theorem applies, so the equilibrium attains the minimax value,

$$\sup_{\mu} \inf_{\nu} V = V(\mu^*, \nu^*) = \inf_{\nu} \sup_{\mu} V. \tag{2.2}$$

In other words, equilibrium play is optimal against a worst-case opponent. This framing makes precise the challenge of acting near-optimally under partial observability and a tight time budget, and it motivates methods — such as the minimax search of PokéChamp [14] — that approximate minimax or Nash play.

2.4. Large language models as game-playing agents

The POMG of Section 2.3 describes *what* a Pokémon agent must solve; this section sketches *how* a large language model can be turned into such an agent, since that paradigm underlies PolimiBot. Historically, game agents were *control-based*: decision-making was coupled to the environment through hand-written state–action rules or a policy learned by classical reinforcement learning. That recipe produced superhuman play in perfect-information, discrete-state domains, but it ties each agent to a single task and a fixed representation. Large language models offer a different substrate. Their parametric prior knowledge, emergent reasoning, and native handling of natural language make text a *universal interface* in which states, goals, and multi-agent interactions can all be expressed, so that the semantic reasoning of the agent is decoupled from the low-level execution of game commands [12]. An agent built this way is commonly called an LLM-based game agent.

2.4.1. A reference architecture

Recent surveys converge on a modular reference architecture for such agents [12], and it is useful to name its parts because PolimiBot (Chapter 5) is an instance of it. A *perception* module acts as a deterministic state translator: because the engine emits numeric, spatial, or visual state while the model consumes tokens, an observation-to-text adapter renders the current situation as a structured summary — a JSON schema or a natural-language description. An *action* module performs the inverse map, turning the model's textual directives back into executable commands through a fixed API. A *memory* system plays the role that a replay buffer plays in reinforcement learning: short-term working memory holds the current context window, while long-term memory is typically offloaded to a vector store and retrieved by embedding similarity, the pattern

known as retrieval-augmented generation. A *reasoning and planning* module is the central controller; rather than reacting in a single step, it uses techniques such as chain-of-thought and self-reflection to decompose a long-horizon objective into sub-tasks before committing to an action. Finally, in multi-agent settings the architecture adds communication and *belief modeling*, so that an agent can maintain and revise an estimate of the hidden state of its peers or opponents. This last component is exactly where the Bayesian Predictor of Chapter 5 sits: it supplies PolimiBot with a calibrated belief about the opponent's concealed team in place of an implicit, learned one.

2.4.2. Constraining the action space

A practical obstacle separates a stochastic language model from a deterministic game engine: ordinary autoregressive decoding produces free-form text and can easily emit a malformed command, an invalid move, or a schema the parser cannot read, breaking the decision loop. The general remedy is *structured decoding*, which enforces the set of legal outputs during inference rather than hoping for it through prompt wording or repairing it afterwards. An automaton is run in parallel with token generation, and before each token is sampled the engine masks every token that would violate the grammar by driving its logit to $-\infty$, confining generation to valid strings [5]. The expressiveness of the automaton sets what can be constrained: a finite-state machine covers regular patterns but cannot capture arbitrarily nested structure, whereas a pushdown automaton enforcing a context-free grammar tracks nesting depth and so admits recursive schemas. Constraining the output in this way converts the model from a probabilistic text generator into a policy that is guaranteed to return an action the engine will accept — the same guarantee PolimiBot needs each turn, where its choice must resolve to one of the legal moves or switches of the POMG's action space. PolimiBot obtains this guarantee at the orchestration level rather than at the decoder, by validating and re-prompting, but the underlying requirement is identical.

2.5. Evaluation metrics

Agents are usually compared on a few measures. The most basic is *win rate* against a given opponent or pool. *Game length* (the average number of turns) is sometimes reported as an indicator of efficiency or style. For ranked online play, rating systems are more informative. *Elo* is the familiar relative-skill rating; on Showdown it is deliberately noisy and is not comparable across formats, and matchmaking tends to pair players of similar Elo. *Glicko-1* is an Elo-like rating that takes a player's whole match history into

account, giving a better estimate of true skill. *GXE* (Glicko X-Act Estimate) corrects for matchmaking bias to estimate a player's probability of beating a randomly chosen opponent; among the very best human players it sits around 74–90%.

Because the game carries substantial inherent variance — much as high-level poker does, where minimizing risk is a skill but some losses are unavoidable — single games are noisy. Ratings aggregated over many games are therefore preferable to one-off win/loss outcomes, and it is good practice to report variability rather than a single point estimate. This caveat is revisited when the experimental results are interpreted in Chapter 6.

3 | Related Work

This chapter situates the thesis within three bodies of work: the games that artificial intelligence has already solved and the methods that solved them; the prompting and planning toolbox that has grown up around large language models; and the specific line of Pokémon-playing agents that this work builds on and compares against.

3.1. Games solved by artificial intelligence

A long line of work has reached, and surpassed, human-level play in competitive games, almost always through extensive imitation learning and self-play paired with deep networks and search. Chess was conquered first by Deep Blue's specialized hardware and search [4] and later, more generally, by AlphaZero's self-play [26]. Go fell to deep networks combined with Monte Carlo Tree Search [3, 25]. The same broad recipe scaled to games with far larger action spaces and partial observability: AlphaStar reached grandmaster level in StarCraft II [28], OpenAI Five defeated professional teams in Dota 2 [1], and Gran Turismo Sophy outraced champion drivers [34]. The recurring lesson is twofold: these systems generally reach superhuman play through reinforcement learning and self-play, frequently in the AlphaGo/AlphaZero lineage of deep networks plus tree search; and the recurring drawback is the heavy, task-specific training and engineering they require.

3.2. Language agents and the prompting toolbox

Large language models offer a different bargain: broad prior knowledge and transfer without task-specific training, at the cost of unreliable planning. A range of prompting and planning techniques aim to close this gap. Chain-of-Thought prompting elicits step-by-step reasoning through worked examples [33]; Self-Consistency samples multiple reasoning paths and aggregates them [32]; Tree-of-Thoughts explores and evaluates multiple reasoning branches [38]; and ReAct interleaves reasoning traces with actions [39]. Further methods treat the language model as a world model for planning [10], add verbal self-reflection with episodic memory [24], guide decoding with AlphaZero-like tree search

[29], or train models to self-correct through reinforcement learning [16].

Despite this toolbox, language agents still struggle with basic planning, which is precisely why a hard game such as Pokémon is a discriminating benchmark for them. The architectural choices in this thesis — decomposing the decision into specialized sub-agents, grounding the model in exact numerical quantities, and supplying it with calibrated probabilistic beliefs about hidden state — can be read as targeted applications of this toolbox to the demands of competitive Pokémon.

3.3. LLM agents across strategic games

Before turning to Pokémon specifically, it is worth surveying how the toolbox of Section 3.2 has been assembled into full agents across other strategic games, since the recurring design problems — and their solutions — recur in this work too. Each genre stresses a different competence: open-ended exploration, real-time control, hidden-role deception, or multi-party negotiation. The systems below illustrate how the modular architecture of Section 2.4 is specialized to each, and where this thesis sits relative to them.

3.3.1. Open-ended and real-time environments

Open-ended sandboxes such as Minecraft remove predefined objectives entirely, demanding *lifelong learning* — the continual acquisition and composition of skills without catastrophic forgetting. Voyager [30] is the seminal design: instead of updating model weights or emitting low-level motor commands, it queries a frozen model and uses *executable code* as its action space, which natively expresses temporally extended, composable behaviors. Three modules close a learning loop — an automatic curriculum that proposes progressively harder goals from the agent's current state, an iterative prompting mechanism that synthesizes code and refines it from interpreter errors and a self-verification critic, and an ever-growing skill library in which verified programs are stored under an embedding of their description and later retrieved by similarity as few-shot examples. This retrieval-augmented code generation lets the agent compound functional skills and reach deep milestones far faster than text-only baselines.

Real-time strategy games such as StarCraft II add the opposite pressure: strict time limits, partial observability through fog-of-war, and an action rate that far outpaces model inference. Agents therefore translate the numeric state into text through an observation-to-text adapter and compile high-level intents back into API calls, while decoupling slow deliberation from fast execution by delegating unit-level micromanagement to pre-compiled

scripts [18]. To keep the rapid telemetry from overflowing the context window, a *Chain of Summarization* hierarchically compresses single frames and then sequences of frames into actionable intelligence before the model commits to a macroscopic intent. A complementary line trades the frontier model for a fine-tuned small one: SC-Phi2 [15] couples a compact language model with a vision encoder and parameter-efficient adaptation to predict build orders at sub-second latency, trading breadth of prior knowledge for the throughput that continuous play demands.

3.3.2. Hidden roles and negotiation

Social-deduction games such as Werewolf and Avalon turn the hidden-information problem of Section 2.1 into the explicit objective: success requires a *theory of mind* — modeling the beliefs and intentions of other players — together with the capacity for deception. Here a zero-shot language model is a weak player, because alignment training biases it toward predictable, cooperative, templated behavior that adversaries readily exploit [36]. The fix is to hybridize semantic generation with game-theoretic optimization, which requires bridging free-form text and the discrete action spaces that solvers assume. One route constrains the model to propose a finite set of candidate utterances and lets a reinforcement-learning policy, trained by self-play, select among them [36]. A more thorough route abstracts the language space into a discrete *latent* strategy space, solves for an equilibrium-approaching policy there with a regret-minimization solver, and then aligns the model to that policy by direct preference optimization so that its natural text realizes the optimized strategy [37]; agents trained this way develop emergent misdirection that fixed prompting does not.

The board game Diplomacy compounds this with seven-way simultaneous play and an enormous joint action space. Cicero [6] first reached human-level play by pairing a dialogue model with a planning engine that searches for equilibrium strategies. DipLLM [35] shows that much of this can be folded into the model itself through *autoregressive factorization*: rather than predicting a whole joint move at once, it decomposes the assignment into a sequence of unit-level decisions, which maps cleanly onto next-token prediction and is fine-tuned toward an equilibrium objective — reaching state-of-the-art play with a small fraction of the data a search-based predecessor required. Richelieu [9], by contrast, evolves its negotiation strategy through self-play and a reflective vector memory, inferring hidden intentions and revising its priors from the logged outcomes of past alliances and betrayals without human demonstrations.

3.3.3. Search and fine-tuning for strategic competence

Two cross-cutting techniques recur across these systems. The first integrates the model with explicit search, since autoregressive generation alone plans shallowly: a model can serve simultaneously as the policy that proposes candidate actions and as the value function that scores intermediate states inside a Monte Carlo Tree Search, letting the agent simulate, backtrack, and select for long-term reward. Variants adapt the search to the model's quirks — replacing costly rollouts with the model's own confidence-weighted self-evaluation when no environmental reward is available, pruning trajectories under the assumption that the model's value estimates carry systematic bias, or attaching machine-checkable constraints to each generated action so that illegal branches are pruned symbolically and the branching factor collapses. The second technique is post-training. Supervised fine-tuning establishes basic competence but tends to memorize demonstrated lines and forget how to generalize, so elite agents are tuned with reinforcement learning: proximal policy optimization [22] maximizes a reward signal under a divergence penalty that guards against reward hacking, while lighter-weight alternatives — direct preference optimization [20], which folds the reward model into the policy and learns from preference pairs, and group-relative optimization [23], which estimates advantages across a sampled group and so needs no separate value network — cut the cost of aligning a model to a strategic objective.

Against this backdrop, the contribution of this thesis is deliberately positioned at the lighter end. PolimiBot keeps a frozen, general-purpose model and invests its effort in the surrounding architecture — a modular perception-and-reasoning pipeline and an explicit, statistically grounded opponent belief — rather than in weight updates or deep tree search. Where the social-deduction and Diplomacy lines embed opponent modeling inside learned policies or equilibrium solvers, the Bayesian Predictor of Chapter 5 makes that belief an explicit, inspectable probabilistic model fed to the agent's prompt; and where the fine-tuning lines pay in data and compute for unexploitable play, this work asks how far a training-free agent can be carried by careful state representation, numerical grounding, and calibrated beliefs about hidden state. The Pokémon-specific systems of the next section are the immediate baselines for that question.

3.4. Pokémon-playing agents

Existing Pokémon systems fall into a few broad families: rule-based and heuristic agents that follow fixed decision rules (including the AI shipped with the retail games and bots

such as Abyssal, used as a baseline in Chapter 6); search-based agents limited by the very large branching factor and the per-turn time budget; supervised approaches that learn from recorded battles to predict opponent behavior; and reinforcement-learning approaches, which have transformed other games but are difficult here because of the scale and partial observability. Self-play reinforcement learning has been applied directly, both without search [13] and, later, augmented with test-time MCTS [31], reaching competitive ratings in random-battle formats. The game's heavy online presence has also made it a target for LLM-based agents, beginning with PokéLLMon, an early prompting-based player that reached human-parity in random battles [11]. The three systems below are the most directly relevant to this thesis.

3.4.1. Foul Play: heuristic search

Foul Play [19] is among the strongest classical Pokémon battle bots and represents the search-based family at its best. It uses a hybrid, two-language architecture: a Python client manages the Showdown connection, parses the live battle stream, and maintains a local model of the game world, while a high-performance Rust engine performs forward simulation, damage calculation, and search. Because the game is one of imperfect information, Foul Play handles hidden state by *sampling*: it constructs a set of complete, deterministic battle states by filling in the opponent's unrevealed Pokémon, moves, items, and spreads. For standard formats such as OU it draws on data from *Smogon* — the principal competitive Pokémon community, which maintains the unofficial tier system of Section 2.2 (OverUsed and the rest) and publishes monthly per-species usage statistics aggregated from high-ladder play — using these usage frequencies and co-occurrence patterns to estimate the likelihood of an unrevealed teammate T given the set of revealed opponents R as

$$P(T \mid R) = \frac{1}{|R|} \sum_{r \in R} \frac{\text{Co-occurrence}(r, T)}{\text{TotalCount}(r)}, \tag{3.1}$$

sampling from the top candidates to fill empty slots and then sampling abilities, items, and movesets by historical frequency subject to validity filters.

On each of the N sampled states it runs a simultaneous-move variant of Monte Carlo Tree Search in parallel. The search resolves both players' choices at a node by maximizing a UCB1 score independently for each side, branches on the chance events intrinsic to Pokémon (accuracy checks, critical hits, damage rolls, secondary effects), and — rather than rolling out to the end of the game — terminates rollouts early and evaluates leaves with a fast heuristic passed through a sigmoid. The policies from all sampled states are then aggregated, weighted by each state's sample probability, with a small amount

of randomization to avoid predictability. Foul Play is relevant here for two reasons: its teammate-likelihood formula in Equation (3.1) is exactly the kind of independent-co-occurrence estimate that the Bayesian Predictor of Chapter 5 improves upon, and its sampling-plus-search design is the principal alternative to the LLM-driven approach taken in this thesis.

3.4.2. PokéChamp: an LLM-guided minimax agent

PokéChamp [14] is the strongest published LLM-based Pokémon agent and the primary point of comparison for this work. It is built as a player within a modified poke-env [21] environment and integrates three layers: an orchestrator that manages the decision loop and the language-model backend, a lightweight simulator that re-implements Showdown mechanics, and a search-tree node that sandboxes simulated states for look-ahead.

On every turn, PokéChamp initializes the simulator, resolves trivial cases (forced switches and forced moves) cheaply, and otherwise translates the battle state into a natural-language prompt before branching to one of its decision algorithms. The state translation concatenates a historical turn summary, a turn-one macro strategy, a micro-strategy analysis of the active matchup (including a speed-tier check and move-effectiveness estimates), the current status (hazards and per-Pokémon details), and an explanation of generation-specific gimmick mechanics such as Terastallization. Its decision algorithms include an input-output baseline with chain-of-thought, self-consistency voting, tree-of-thoughts, and — its core contribution — a hybrid minimax tree search. The minimax search prunes the enormous action space by generating only a few candidate actions per node (the best damage-calculator move plus a handful of LLM-proposed moves and switches), predicts the opponent's simultaneous action from heuristics and an opponent-perspective LLM query, simulates each combination to a shallow depth, and evaluates leaf states by asking the LLM to score them from 1 to 100. Scores are propagated by the minimax principle: each player action is valued by its worst-case opponent response, and the action maximizing this value is chosen.

PokéChamp reaches expert-level play and is a strong baseline, but its design has a number of concrete weaknesses — in how it represents history, predicts hidden opponent information, conflates move and switch decisions, calculates damage, and evaluates leaf nodes — that directly motivate the design of PolimiBot. Chapter 4 examines these in detail.

3.4.3. Offline reinforcement learning on archived games

A contrasting recent direction dispenses with explicit search and instead trains sequence models with imitation and offline reinforcement learning, relying on long-context memory to infer the opponent's team and tendencies [8]. This approach builds on batch (offline) reinforcement learning [17] and the Transformer architecture [27], and on a key practical idea: *reconstructing trajectory data* by rebuilding spectator-viewpoint replay logs into the partial-observation point of view of an agent, yielding a naturally occurring offline dataset that grows over time. Experience to date suggests this model-free, long-context route can reach strong human-level play and rival the best heuristic search engines, whereas LLM agents that act turn-by-turn without search tend to struggle over the long horizons of the early generations. This thesis draws on the same archived-replay resource — the Metamon Teams dataset [7] — to train the Bayesian Predictor, but applies it to opponent modeling in support of an LLM agent rather than to end-to-end policy learning.

4 | Problem and Motivation

PokéChamp [14] establishes that an LLM-guided agent can play competitive Pokémon at an expert level, and it is the strongest published baseline of its kind. A close reading of its design, however, reveals a set of concrete weaknesses that limit both its decision quality and its efficiency. This chapter catalogs those weaknesses and groups them into four themes, each of which motivates a specific element of the design presented in Chapter 5. The fifteen individual issues are summarized in Table 4.1; the appendix reproduces the full point-by-point table together with the corresponding improvement (Chapter A). Throughout, the aim is not merely to criticize a strong system but to use it as a precise specification of what a better LLM agent must do.

4.1. Prompt representation and efficiency

PokéChamp prepends a raw, sixteen-turn natural-language transcript of battle messages to every prompt. This forces the language model to re-derive the current board state from a long event log rather than reading it directly, and the transcript inflates the prompt to an average of roughly 9,400 tokens per turn. Under the strict time and, in practice, cost budgets of competitive play (Section 2.2), this is wasteful: most of the history is either already reflected in the present state — HP remaining after past damage, active stat changes from past moves — or irrelevant to the current decision. The motivating question is whether the same information can be conveyed by encoding the current state directly, at a fraction of the token cost.

This design choice is grounded in a structural property of Pokémon battles: the game is *near-Markovian*. The current board state — HP values, active status conditions, stat-stage modifiers, field effects, and the portion of the opponent's team that has been revealed — is almost sufficient to determine the optimal action, independently of the sequence of moves that produced it. Past events matter only insofar as they are already reflected in the present configuration; the raw textual log that led there is therefore largely redundant.

A small number of genuine history-dependencies do exist. The most salient is Protect

and its variants (Detect, Spiky Shield, Baneful Bunker): when a Pokémon uses Protect on consecutive turns, the probability of success drops by half with each successive use, so the immediately preceding action is materially relevant to the current decision. Likewise, the power of moves such as *Last Respects* scales with the total number of fainted teammates over the course of the battle, and *Rage Fist* accumulates power with each hit the user has received — neither figure is readable from the static snapshot alone. These exceptions are narrow and manageable: a Protect-streak counter and a small set of cumulative move-power fields can be appended to the state representation as scalar values, preserving the near-Markovian structure without resorting to a full event transcript.

4.2. Modeling the hidden opponent

A cluster of weaknesses concerns how PokéChamp reasons about the parts of the opponent it cannot see — the crux of the game, as argued in Section 2.1. For unknown opponent moves it fills empty slots with the highest-frequency entries from a static dictionary, with no conditioning on what has actually been revealed during the battle and no probabilistic signal attached. Unrevealed opponent Pokémon are simply absent from the prompt, so early in a battle the model knows nothing about most of the opposing team. The opponent's item, ability, and tera type are left empty until directly observed. And the damage the agent deals is never recorded or used to refine an estimate of the opponent's defensive investment, so every turn starts from the same static priors. Collectively these gaps mean the agent reasons about a far more uncertain opponent than the available statistical regularities would permit. This theme motivates the Bayesian Predictor of Section 5.3.

4.3. Decision structure

PokéChamp resolves each turn with a single language-model call whose output must be either a move or a switch. Yet move reasoning (type coverage, STAB, KO thresholds) and switch reasoning (incoming damage, hazard cost, pivoting) are structurally different problems, and there is no mechanism to weigh one against the other with numerical grounding. Related shortcomings compound this: the self-consistency voting mode repeats the same monolithic prompt without giving the votes any reason to be diverse or complementary; pivot moves (U-turn, Volt Switch, Flip Turn) are routed to the same logic as a passive forced switch, with no account of their strategic purpose; and the core minimax search evaluates leaf nodes by asking the LLM for an uncalibrated 1–100 score. Because the minimax decision compares these scores *across different LLM calls*, and because the same

position can plausibly score 62 in one call and 75 in another, the resulting comparison is effectively comparing noise — a problem made worse by a mixed-scale fallback that mixes 1–100 scores with raw HP differences when a call fails. This theme motivates the decomposed three-prompt decision flow of Section 5.2.

4.4. Numerical grounding

The final theme concerns the accuracy of the numbers the agent reasons over. PokéChamp's damage estimates come from an internal Python approximation that misses many item, ability, weather, and multi-hit interactions, and it expresses move power as a dimensionless ratio rather than actual HP, making KO thresholds hard to read. Its speed comparison uses only the base Speed stat adjusted by stat stages, ignoring the full chain of modifiers — Choice Scarf, paralysis, weather and terrain abilities, Tailwind, Booster Energy, and the reversal under Trick Room — and an incorrect speed verdict causes catastrophically wrong play. Entry hazards are described only in generic text, with no per-switch HP cost computed, even though Stealth Rock damage scales sharply with the incoming Pokémon's Rock-type effectiveness. And bench Pokémon are annotated only with a type-effectiveness multiplier, not a KO estimate, so the model cannot directly compare staying with switching. These issues motivate grounding every decision in exact turns-to-KO figures from a canonical damage calculator, with effective post-hazard HP and a complete speed-modifier chain, as described in Chapter 5.

Table 4.1: The fifteen weaknesses identified in PokéChamp, grouped by theme, each addressed by an element of PolimiBot or the Bayesian Predictor. The full point-by-point comparison is reproduced in Chapter A.

#	Theme	Weakness in PokéChamp
1	Prompt efficiency	Sixteen-turn raw history transcript inflates token cost
2	Hidden opponent	Naïve static move prediction, no in-battle conditioning
9	Hidden opponent	Unrevealed opponent Pokémon absent from the prompt
10	Hidden opponent	Opponent item, ability, tera type left empty until observed
11	Hidden opponent	No damage-evidence tracking for EV inference
3	Decision structure	Move and switch conflated in a single call
13	Decision structure	Self-consistency voting is architecturally unstructured
14	Decision structure	No dedicated pivot-move forced-switch flow
15	Decision structure	Uncalibrated, cross-call leaf scores; mixed-scale fallback
4	Numerical grounding	Hard-coded tera heuristic misses most tera plays
5	Numerical grounding	No per-Pokémon authored strategic context
6	Numerical grounding	Inaccurate Python damage calculator; ratio-based power
7	Numerical grounding	Incomplete speed-modifier chain
8	Numerical grounding	Generic hazard text; no per-switch HP cost
12	Numerical grounding	Bench moves show type multiplier only, no KO estimate

5 | Methodology

This chapter presents the design of the two contributions of the thesis. It opens with the design philosophy shared by both (Section 5.1), then describes PolimiBot, the agent that drives turn-by-turn decisions (Section 5.2), and the Bayesian Predictor, which infers hidden opponent information (Section 5.3). A final section explains how the predictor feeds the agent each turn (Section 5.4).

5.1. Design philosophy

The central insight behind PolimiBot is that competitive Pokémon battle decisions are complex enough to resist simple heuristic approaches, yet structured enough to be described in natural language. Rather than hand-crafting decision rules, the agent translates the full battle state — numerical damage calculations, status conditions, terrain, team composition, and predicted opponent information — into a prompt that a capable language model can reason over. Two principles guide every design choice.

Ground the model in exact numbers. Left to its own priors, an LLM reasons about game mechanics from imperfect memory, and small numerical errors (an incorrect KO threshold, a wrong speed verdict) translate into catastrophically wrong play. PolimiBot therefore computes every decision-relevant quantity offline and injects it directly: exact turns-to-KO from the community-standard `@smogon/calc` damage calculator — invoked through Node.js, so the agent inherits the full Generation 9 mechanics (item and ability interactions, multi-hit moves, weather and terrain, entry-hazard scaling) rather than an internal approximation — effective HP after entry hazards, the full speed-modifier chain, and an explicit verdict of who moves first. The model is asked to reason about *strategy*, not to re-derive *arithmetic*.

Decompose the decision, then adjudicate with numbers. A single prompt that must simultaneously choose a move, decide whether to switch, and weigh the two against each other conflates structurally different reasoning problems. PolimiBot instead routes

each turn through specialized sub-agents and resolves any conflict between them with a final judge anchored to a pre-computed KO-race table, so the binary move-or-switch decision rests on objective numbers rather than an uncalibrated score.

The agent is deliberately LLM-agnostic: it can work with any model exposed through a common interface, including those served via OpenRouter, so the design does not depend on a single provider. It targets Generation 9 OU (Section 2.2). The two contributions are complementary: PolimiBot defines *how* the agent reasons each turn, and the Bayesian Predictor defines *what the agent believes* about the hidden opponent state it reasons over.

5.2. PolimiBot: the battle agent

Each turn, PolimiBot constructs a rich natural-language description of the current battle state and queries a language model to decide what action to take. This section describes the information the agent assembles, the decomposed decision flow that consumes it, and two further mechanisms — majority voting and damage-evidence tracking — that improve robustness and feed the opponent model.

5.2.1. What the agent knows each turn

Before querying the model, PolimiBot assembles a set of structured information blocks. Unlike a transcript of past events (Section 4.1), these blocks describe the *current* state exhaustively, so the model never has to reconstruct the board from history.

System prompt and strategic notes. The system prompt establishes the model's role as a competitive Generation 9 OU battler and injects the per-Pokémon strategic notes for the currently active Pokémon. These notes are authored manually and encode team-specific knowledge: the intended role of each Pokémon (for example, "this Pokémon is a wallbreaker; prioritize Nasty Plot setup"), move preferences, and the situations in which Terastallization is most valuable. By injecting only the *active* Pokémon's notes, and at the system level, the agent has focused access to authorial intent before reasoning about the immediate state.

Side conditions. This block lists all active entry hazards on both sides, with their concrete effects: which types are immune, how much HP is lost on switch-in (Stealth Rock damage scales with Rock-type effectiveness), and whether a Pokémon will be poisoned by Toxic Spikes. Hazard costs are pre-computed and also shown alongside each switch candidate, so the model sees the effective HP after entry rather than the current HP.

Active Pokémon. Two blocks describe the two active Pokémon — the agent's own and the opponent's — each giving species, type, HP percentage, ability, and item; status and volatile statuses; stat changes from moves such as Dragon Dance or Intimidate; the move list (the full set for the agent's own Pokémon, only revealed moves for the opponent); the tera type; and a plain-English statement of who moves first. Each move entry carries its type, damage category, base power, a natural-language effect description, and — critically — the turns-to-KO estimate from the damage calculator (for example, "Earthquake: [Ground, Physical, 100] — 2 turns to KO opponent"). For the opponent's active Pokémon, any unknown fields are filled in by the Bayesian Predictor (Section 5.3).

Bench Pokémon. Two analogous blocks describe the non-fainted, non-active Pokémon on each side. For the agent's own team, each bench Pokémon's moves are annotated with a turns-to-KO estimate against the current opponent, giving the model a complete picture of which bench options threaten it. For the opponent's bench, revealed Pokémon and their observed moves are listed, and the predictor fills in likely unrevealed Pokémon and their probable moves.

Terastallization and speed order. A dedicated block explains whether Terastallization is still available for either side, what tera type the agent's active Pokémon would gain, and — from team knowledge or prediction — what tera type the opponent is likely holding, surfaced as a warning when the opponent retains the option, since an unexpected type change can invalidate the damage calculator's predictions. A separate speed block computes the full modifier chain for both Pokémon — stat stages, Choice Scarf ($\times 1.5$), Iron Ball ($\times 0.5$), paralysis ($\times 0.5$), weather and terrain abilities, Tailwind ($\times 2$), and Booster Energy / Protosynthesis / Quark Drive when Speed is the highest stat — inverts the order under Trick Room, and states an explicit verdict ("you move first" or "opponent moves first") together with what it means for damage sequencing.

The action menu. Finally, every legal action for the turn is listed as a structured menu: each move with its type, category, power, and effect, and each switch candidate with its current HP and the pre-computed hazard cost giving an effective post-entry HP.

5.2.2. The three-prompt decision flow

When both moves and switches are available, PolimiBot reaches a decision through three sequential model calls. The decomposition reflects the structured nature of the choice: move selection and switch selection involve different reasoning patterns, while a merger adjudicates between them with numerical grounding (Figure 5.1).

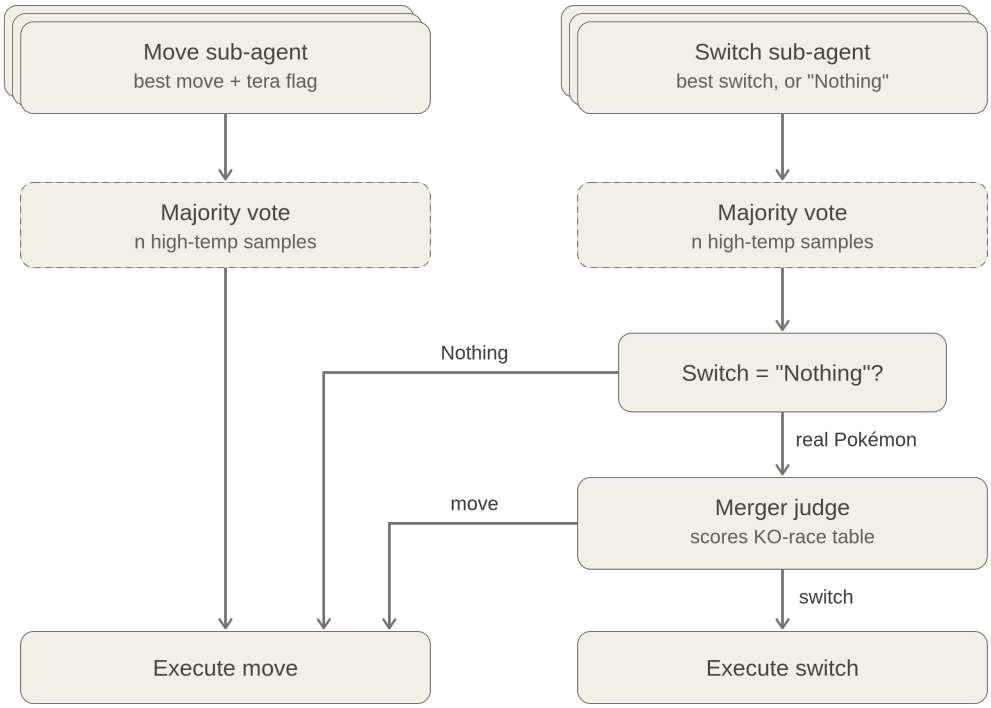

Figure 5.1: PolimiBot's three-prompt decision flow with majority voting. The Switch Sub-Agent can short-circuit the Merger by returning "Nothing".

Move Sub-Agent. Given the full context and the available-moves list (with KO estimates), the model selects the best move and decides whether to terastallize. Delegating the tera decision to the model — informed by the authored strategic notes — replaces PokéChamp's hard-coded heuristic, which approved Terastallization only at full HP and so missed defensive and coverage uses. The sub-agent returns a JSON object with a brief explanation and the chosen move.

Switch Sub-Agent. Given the same context and the available-switches list (with hazard-adjusted HPs), the model evaluates whether switching out is warranted. It returns either a Pokémon name or the special value `"Nothing"` to signal that attacking is better. When `"Nothing"` is returned, the switch stage is short-circuited and the move sub-agent's choice is executed immediately.

Merger Judge. When the switch sub-agent names a real Pokémon, the merger receives a pre-computed KO-race summary table covering all combinations (each of the agent's moves against the opponent, plus each switch candidate's best offense and worst-case incoming threat). The merger applies a conservative switching bias — switching grants

the opponent a free attack, so it must show a clear numerical advantage — and outputs a binary `"move"` or `"switch"`. Crucially, the merger reasons over objective KO-turn numbers from the same calculation, so unlike PokéChamp's minimax leaves (Section 4.3) there is no cross-call score comparison and no mixed-scale fallback.

Special cases. Three situations bypass the full flow. If the active Pokémon has fainted, only the Switch Sub-Agent runs, with a simplified forced-switch prompt. If a pivot move (U-turn, Volt Switch, Flip Turn) was used, the Pokémon must switch out, so only the Switch Sub-Agent runs, but it includes the KO-race summary so the best incoming Pokémon can be chosen with full grounding. If no switches are available, only the Move Sub-Agent runs.

5.2.3. Majority voting

To reduce the variance of any single stochastic response, PolimiBot supports optional majority voting. When the voting parameter $n > 1$, each sub-agent call is parallelized n times at a raised sampling temperature to encourage diverse responses, and the most-voted action string wins before being passed to the merger. Votes are tallied per sub-agent, so move votes and switch votes do not interfere; in particular, "Nothing" votes from the Switch Sub-Agent are tallied separately, and a majority of them short-circuits the merger entirely, saving a call and avoiding a misleading tie. Unlike PokéChamp's self-consistency mode, which repeats one monolithic prompt, the voting here is structured around the specialized sub-agents.

5.2.4. Damage-evidence tracking

Each turn, PolimiBot records the opponent's active HP before and after its own move. When a damage event is detected (an HP drop greater than 0.5%), it stores the attacker's stats, the move used, and the observed damage percentage. This record is passed to the Bayesian Predictor as evidence for backward EV inference (Section 5.3.6): if the observed damage is inconsistent with the opponent running a particular defensive spread, the predictor down-weights that hypothesis. The agent thus does not merely consume predictions — it actively supplies the observations that sharpen them.

5.3. The Bayesian Predictor

In competitive Pokémon, each team member can carry any of hundreds of possible items, abilities, move combinations, and EV spreads, all hidden until revealed in battle (Sec-

tion 2.1). The Bayesian Predictor fills in these unknowns with statistically grounded estimates rather than guesses. It is trained on approximately 140,000 Generation 9 OU teams from the Metamon Teams dataset [7], specifically the `gl_05_26` split, which is derived from ladder games and tournament replays with a preference for replays that reveal more of the ground-truth composition.

5.3.1. Model hierarchy

The predictor is organized as a hierarchy of statistical models, summarized in Figure 5.2. A singleton entry point is called by PolimiBot each turn; beneath it, a team predictor combines four components: per-species frequency tables built from replays, a prior derived from Smogon usage statistics, a latent-archetype model of team composition, and an evidence updater that conditions on what has been observed during the battle (including backward inference from observed damage).

```
PokemonPredictor          ← singleton, called each turn
  ⌞ BayesianTeamPredictor
      ⊢ ComponentModel    ← per-species frequency tables
      ⊢ SmogonPrior       ← normalized Smogon usage stats
      ⊢ ArchetypeModel    ← LDA topic model over teams
      ⌞ EvidenceUpdater   ← conditioning on observations
          ⌞ DamageInference ← backward EV inference
```

Figure 5.2: The hierarchy of statistical models in the Bayesian Predictor.

5.3.2. Per-species frequency tables

The foundation is a set of frequency counters built from the replay corpus. For each species, the model stores how often each move, item, ability, nature, EV spread, and tera type was observed, together with pairwise move co-occurrence counts — how often two moves appear together on the same Pokémon — which later enable evidence conditioning. Before normalization the raw counts are *Dirichlet-smoothed*: a constant pseudo-count $\alpha = 0.5$ is added to every category — every candidate move, item, ability, and so on — so that a combination never observed for a species still receives a small non-zero probability instead of zero mass. Equivalently, the smoothed estimate is the posterior mean of the categorical distribution under a symmetric Dirichlet(α) prior over its categories, from which the technique takes its name; the value $\alpha = 0.5$ is the Jeffreys prior, a half-count that barely perturbs well-populated categories while keeping rare ones reachable.

5.3.3. Smogon usage prior and adaptive fusion

Smogon publishes monthly usage statistics for Generation 9 OU, reporting, for each species, a usage-weighted breakdown of the sets it appears with. A record opens with the species' raw usage count, its average weight, and a viability ceiling, and then lists, as percentages, the distribution over each component: abilities, held items, EV/IV spreads (given as a nature together with the six stat investments), moves, and Tera types. It also reports the most common teammates and the species' main checks and counters. Table 5.1 shows an abridged record for Great Tusk, one of the most-used Pokémon in the format, as an illustration of the format and of how sharply usage concentrates on a few choices per component.

Table 5.1: An abridged Smogon usage record for Great Tusk (raw count 668,457; viability ceiling 92), showing the top entries of each component the prior consumes. Percentages are usage shares; the catch-all "Other" bucket is omitted.

Component	Top entries (usage share)
Ability	Protosynthesis (100.0%)
Item	Booster Energy (29.5%), Heavy-Duty Boots (26.5%), Rocky Helmet (15.7%), Leftovers (15.1%)
Spread	Jolly 0/252/0/0/4/252 (19.7%), Jolly 0/252/4/0/0/252 (19.7%), Jolly 252/4/0/0/0/252 (10.7%)
Moves	Rapid Spin (92.0%), Headlong Rush (78.2%), Ice Spinner (69.2%), Close Combat (38.3%), Stealth Rock (37.7%)
Tera type	Steel (25.7%), Ground (20.0%), Water (13.8%), Ice (9.5%)

These statistics complement the replay frequencies: they cover a broader sample and represent the current metagame, whereas the replay corpus may skew toward certain archetypes or time periods. The prior consumes the per-species distributions over abilities, items, spreads, moves, and Tera types; the teammate and checks-and-counters fields are not used here, since teammate correlations are modeled instead from the replay corpus by the archetype model (Section 5.3.4). The catch-all "Other" bucket in each distribution — the residual mass of items or moves individually below the reporting threshold — is discarded so that no probability is assigned to an uninformative category.

Not every component is equally predictable from usage statistics. Figure 5.3 quantifies this across all 449 species in the Smogon record using two complementary measures, computed per species and component: the usage share captured by the single most common choice, and the Shannon entropy of the full usage distribution. The two panels tell the same story from opposite directions. Abilities are nearly deterministic: the median most-used ability accounts for roughly 88% of a species' usage, a large fraction of species run a

single ability in 100% of their appearances, and the median entropy is about half a bit. Held items, EV spreads, and Tera types are far more dispersed — median top-1 shares near 40% and entropies typically between two and three bits — meaning that for these components even a perfect usage model leaves genuine uncertainty that only in-battle evidence can resolve. This asymmetry is a structural property of the metagame rather than of any particular model, and it anticipates the accuracy gradient observed in the evaluation (Section 6.2), where ability prediction is near-perfect while item and Tera-type prediction remain substantially harder.

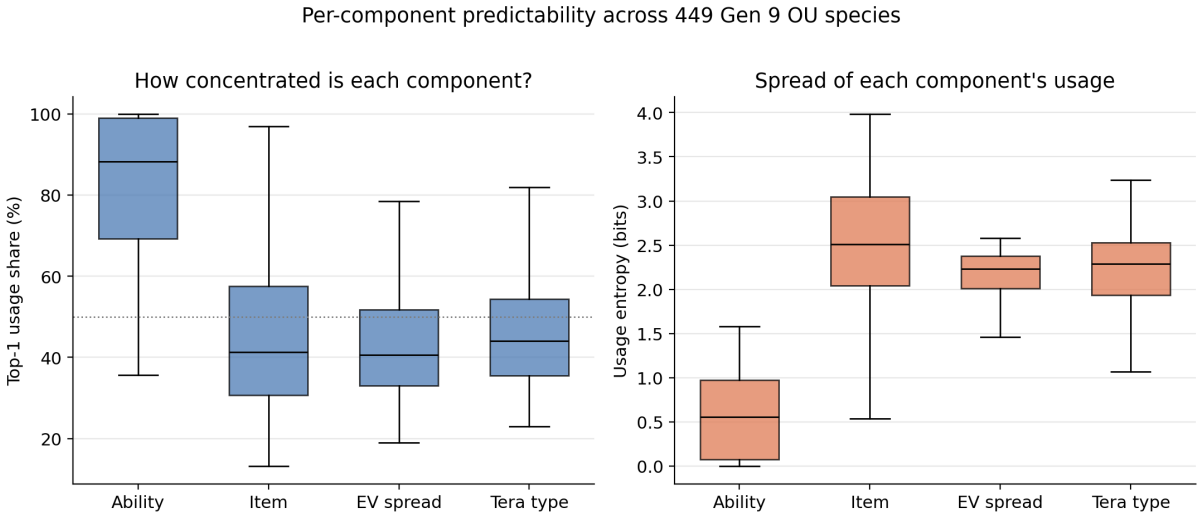

Figure 5.3: Predictability of each set component across the 449 Generation 9 OU species in the Smogon usage record. Left: distribution (across species) of the usage share held by the most common choice of each component. Right: distribution of the Shannon entropy of each component's usage distribution. Boxes span the interquartile range with the median marked; whiskers extend to the most extreme non-outlying species. Abilities are close to deterministic, whereas items, EV spreads, and Tera types are intrinsically dispersed.

The replay distribution and the Smogon prior are blended, per species and component, into a single distribution by a convex combination — a *linear interpolation* of the two — with a weight $\lambda \in [0.1, 1]$ on the prior that adapts to how much replay data is available:

$$\lambda = \max\left(0.1,\ 1 - \sqrt{\frac{c_{\text{replay}}}{c_{\text{replay}} + 0.05\, c_{\text{smogon}}}}\right), \tag{5.1}$$

$$\text{fused}[k] = \lambda\, \text{smogon}[k] + (1 - \lambda)\, \text{replay}[k], \tag{5.2}$$

where c_{replay} and c_{smogon} are the numbers of observations backing the species' replay and Smogon distributions. Equation (5.2) is the mixture itself: λ is the share of mass given

to the Smogon prior and $1 - \lambda$ the share given to the replay estimate, so the weights sum to one and the result is again a valid probability distribution. Equation (5.1) sets that share from the available evidence. When a species has been seen in many replays, the fraction under the root tends to one, the root tends to one, and λ falls to its floor of 0.1, so the fused estimate is dominated by the replay data; when a species is rare in the replays the fraction tends to zero and λ rises towards one, leaning on the prior. The constant 0.05 discounts the Smogon count so that even a modest number of replays is enough to outweigh the prior — a tunable trust parameter governing how quickly replay evidence takes over — while the outer $\max(\cdot, 0.1)$ keeps at least a tenth of the weight on the prior, so a single species' own data can never drive the broader metagame signal to zero. The expression is a count-based confidence weighting (a data-dependent shrinkage towards the prior, in the spirit of interpolated smoothing) rather than a closed-form Bayesian posterior, chosen for its robustness on sparsely observed species. Note that this mixing weight λ is a different quantity from the Dirichlet smoothing constant $\alpha = 0.5$ of Section 5.3.2: that α is a per-category pseudo-count that smooths a *single* distribution from within, whereas λ weights *two* whole distributions against each other; they act at different stages and are otherwise unrelated. Species with many replays thus trust their replay data more, while species rarely seen lean harder on the Smogon prior. This avoids over-fitting to sparse data for niche Pokémon, and is one of the two structural reasons the redesigned predictor is expected to generalize better than a pure replay-count baseline (Section 6.2).

Figure 5.4 makes the hand-off of Equation (5.1) visible by plotting λ as a function of the replay count for three representative Smogon usage counts. With only a handful of replay observations the weight sits close to one and the fused distribution is essentially the prior; as replay evidence accumulates the weight decays smoothly, and it reaches the 0.1 floor once the replay count is roughly a fifth of the Smogon count ($c_{\text{replay}} \approx 0.21\, c_{\text{smogon}}$, the solution of Equation (5.1) at $\lambda = 0.1$). The more popular a species is on the ladder — and hence the larger its Smogon count — the more replay evidence is demanded before the model commits to its own corpus statistics.

Figure 5.5 then shows the fusion operating on real data, for the held-item component of two species at opposite ends of the data-availability spectrum. For Great Tusk, the most frequent species in the replay corpus ($c_{\text{replay}} = 33{,}974$, giving $\lambda = 0.29$), the replay estimate and the Smogon prior largely agree, and the fused distribution tracks the replay data with a mild correction toward the prior. For Pineco, observed only ten times in the corpus ($\lambda = 0.48$), the raw replay estimate is visibly over-concentrated — items that happened to appear in two of ten observations receive close to 20% of the mass — and

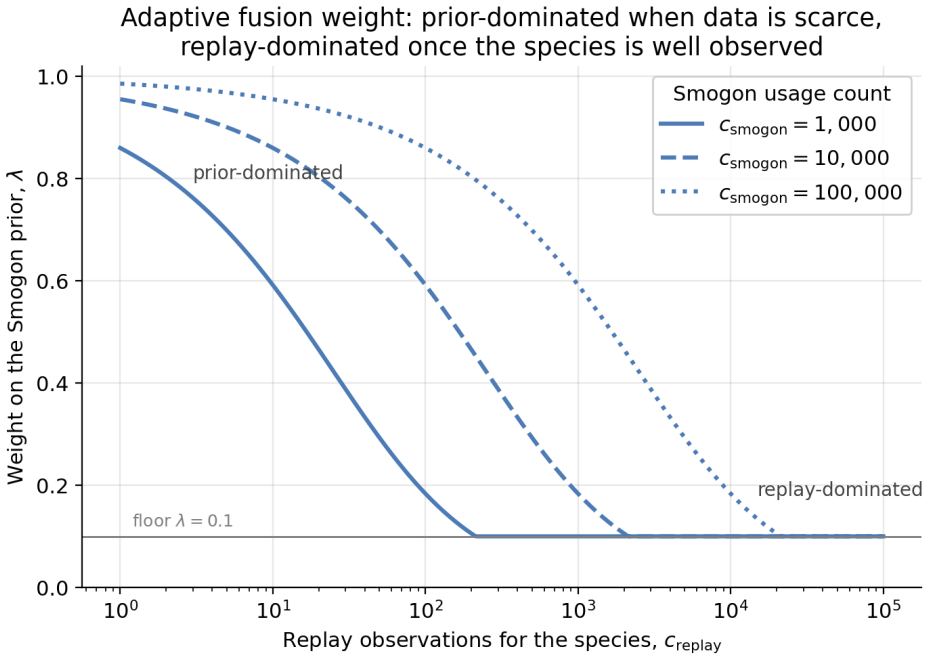

Figure 5.4: The adaptive fusion weight λ of Equation (5.1) as a function of the number of replay observations c_{replay}, for three representative Smogon usage counts (note the logarithmic horizontal axis). Sparsely observed species are prior-dominated (λ near 1); abundantly observed species are replay-dominated, with λ clamped at the floor of 0.1 so that the metagame-wide signal is never discarded entirely.

the fusion both dampens these small-sample artifacts and reinstates items, such as Rocky Helmet, that the prior considers common but that never surfaced in so few replays. This is precisely the shrinkage behavior the adaptive weight was designed to produce.

5.3.4. Team composition model

Pokémon teams are not collections of independently chosen Pokémon — certain species cluster into archetypes (weather teams, hyper-offense cores, defensive pivots, and so on). This structure is plainly visible in the raw usage data: Figure 5.6 draws the teammate co-occurrence graph over the forty most-used species of the format, keeping each species' strongest few links, and partitions it with a standard community-detection algorithm (greedy modularity maximization). The resulting communities correspond to groupings a competitive player would recognize immediately — a weather-offense cluster, a cluster of bulky defensive staples, and a core of ubiquitous balance Pokémon bridging them. Any model that predicts unrevealed teammates must therefore capture which cluster the opponent's team belongs to, not merely which species are individually popular.

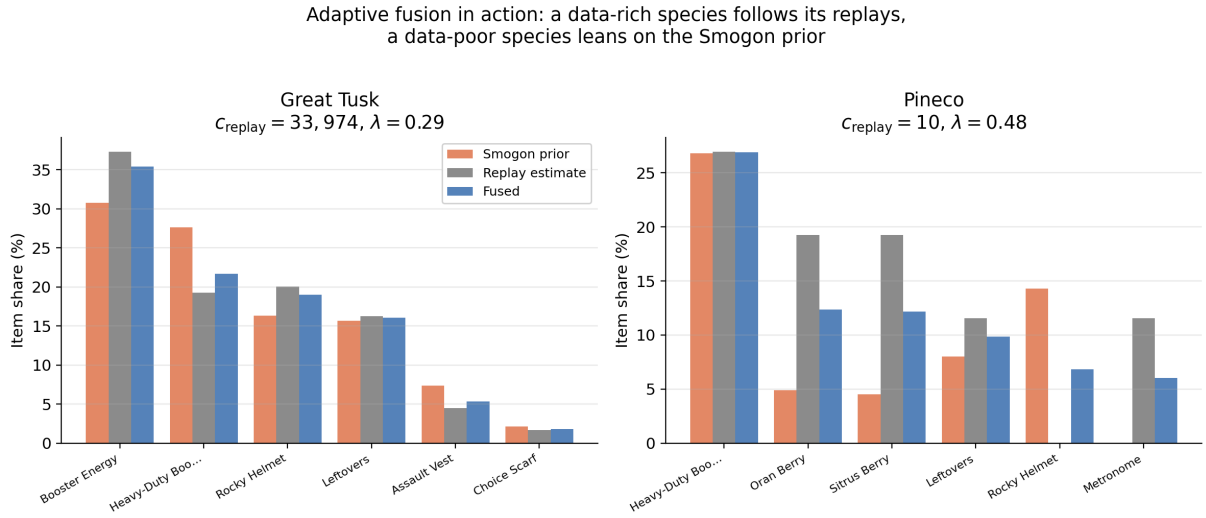

Figure 5.5: Adaptive fusion in action on the held-item distributions of two species (Smogon prior, replay estimate, and fused result, restricted to the top fused items). Left: Great Tusk, a data-rich species whose fused distribution follows its abundant replay evidence. Right: Pineco, a data-poor species (10 replay observations) whose over-concentrated replay estimate is shrunk toward the broader Smogon prior, which also restores plausible items absent from the sparse replay sample.

The archetype model captures these correlations with *Latent Dirichlet Allocation* (LDA) [2]. LDA is a generative probabilistic topic model from natural-language processing: it represents each document as a mixture over a fixed number of latent *topics*, where each topic is itself a distribution over the vocabulary, and fitting the model to a corpus discovers the topics that best explain which words tend to co-occur. Mapping that structure onto team building, the model is trained with 20 topics on the six-species lists of the replay corpus: each team is treated as a six-word document, the species form the vocabulary, and the learned topics play the role of archetypes. After training, each topic defines a distribution over species and each team is explained as a mixture of these latent archetypes.

Figure 5.7 shows the twenty archetypes the model learns from the corpus. Each row is one topic — labeled automatically with its three most probable species and its prevalence in the corpus — and each cell gives $P(\text{species} \mid \text{archetype})$; rows and columns are ordered so that each topic's signature species fall near the diagonal. The matrix exhibits a clear block structure: each archetype concentrates its mass on a small, internally coherent set of species, and the recovered groups map onto established competitive archetypes without any supervision — among them sun offense (Walking Wake, Great Tusk, Raging Bolt), rain (Iron Treads, Raging Bolt, Pelipper), and full stall (Gliscor, Dondozo, Corviknight).

Figure 5.6: Teammate co-occurrence network of the forty most-used Generation 9 OU Pokémon, built from the Smogon teammate statistics. Node size is proportional to usage; edge thickness to co-occurrence weight; colors mark the communities found by greedy modularity maximization. The communities align with the archetypes of competitive parlance, motivating a team-composition model that captures cluster membership explicitly.

That an unsupervised topic model recovers the archetypes named by the community is the central qualitative evidence that the "teams as mixtures of archetypes" view is the right generative description of team building.

At inference time the model weights topics using the already-revealed Pokémon and marginalizes to a distribution over the unrevealed slots, blended 70/30 with the base usage rate to prevent over-concentration on a single archetype:

$$P(\text{unrevealed species}) = 0.7 \times \text{archetype score} + 0.3 \times \text{base usage rate}. \tag{5.3}$$

Figure 5.8 illustrates this inference on a team drawn from the corpus, revealed one Poké-

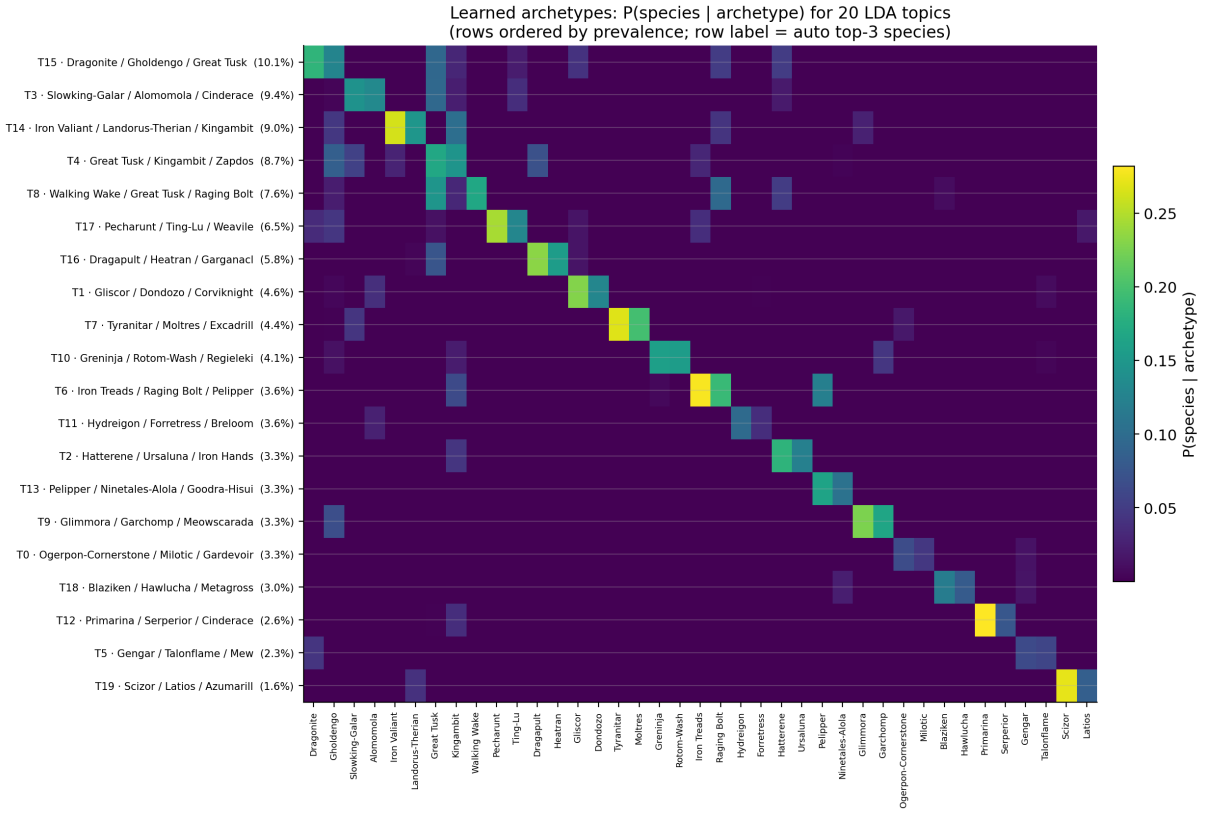

Figure 5.7: The twenty archetypes learned by the LDA model, as the matrix $P(\text{species} \mid \text{archetype})$. Rows are topics, ordered by corpus prevalence (shown in parentheses) and labeled with their three most probable species; columns are the union of each topic's top species, ordered to align signature species with their topic. The block-diagonal structure shows that each archetype concentrates on a small coherent core, recovering recognized team styles (sun, rain, stall, balance) without supervision.

mon at a time. After a single revealed Pokémon (Great Tusk, a near-ubiquitous species compatible with many archetypes), the predicted teammate distribution is dominated by generic format staples and none of the team's actual members appears among the top candidates. Revealing Walking Wake — a strong sun indicator — shifts the inferred archetype mixture decisively: two of the team's true members (Ninetales and Torkoal, the canonical sun core) immediately enter the top eight predictions. Confirming Torkoal then also raises Venusaur, a third true member, into the candidate list. The distribution is never forced to a single archetype — the 30% base-rate component of Equation (5.3) keeps popular species in view — but the team's genuine composition rises steadily through the ranking as the revealed evidence identifies the archetype.

This is a deliberately different model from the independent-co-occurrence estimate used by Foul Play (Equation (3.1)) and the naïve Bayes baseline. Multiplying individual con-

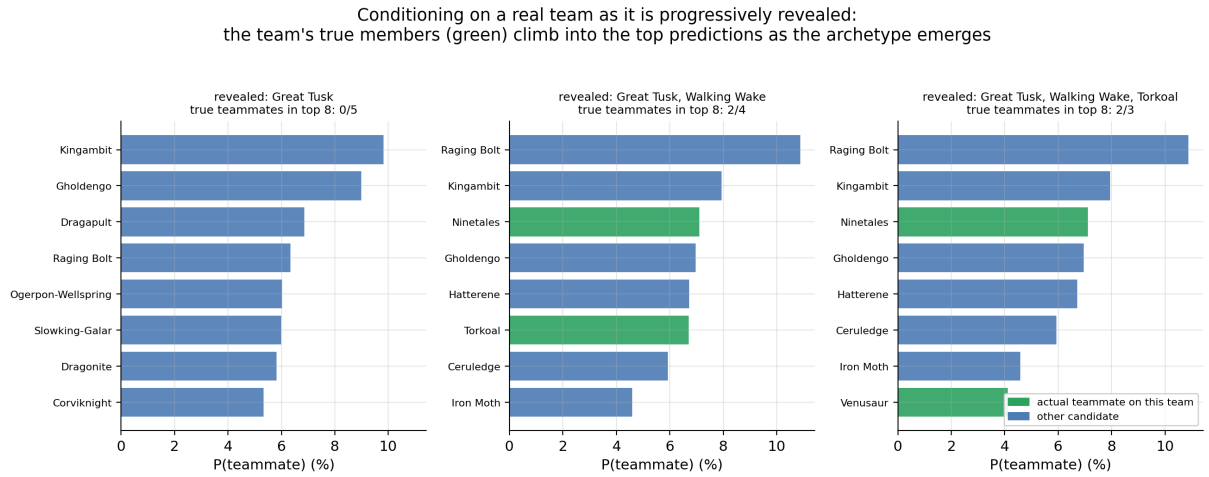

Figure 5.8: Teammate inference conditioned on a real corpus team as it is progressively revealed (left to right: one, two, and three revealed Pokémon). Bars give the predicted probability of the top eight unrevealed candidates; bars in green mark species that are actually on the team in question. With only the archetype-ambiguous Great Tusk revealed, no true member is ranked highly; once Walking Wake identifies the sun archetype, the true sun core (Ninetales, Torkoal, then Venusaur) climbs into the top predictions.

ditionals $P(B \mid A) \times P(C \mid A) \times \cdots$ assumes teammates are chosen independently given the anchor species, which double-counts the correlations within a synergistic core and produces overconfident, overly narrow distributions. The LDA model instead treats teams as mixtures of latent archetypes — the correct generative view — capturing inter-species correlations globally and marginalizing correctly over unrevealed slots. Figure 5.9 demonstrates the difference on a concrete case: conditioning on the core of Great Tusk and Kingambit — two species that co-occur heavily with each other and with the same third partners — the naïve Bayes estimate piles nearly a quarter of the entire probability mass onto a single candidate (Gholdengo), because the evidence from each revealed Pokémon is counted as if it were independent when it is in fact largely the same evidence twice. The archetype model, integrating over its topic mixture, assigns the same candidate a far more realistic share of about five percent and distributes the remainder across the other members of the implied archetype. In a format where, as Figure 5.8 shows, even a well-identified archetype leaves several plausible fillers, the broader distribution is not a weakness but the honest representation of the residual uncertainty — and it is what keeps the confidence labels attached to these predictions calibrated (Section 6.2).

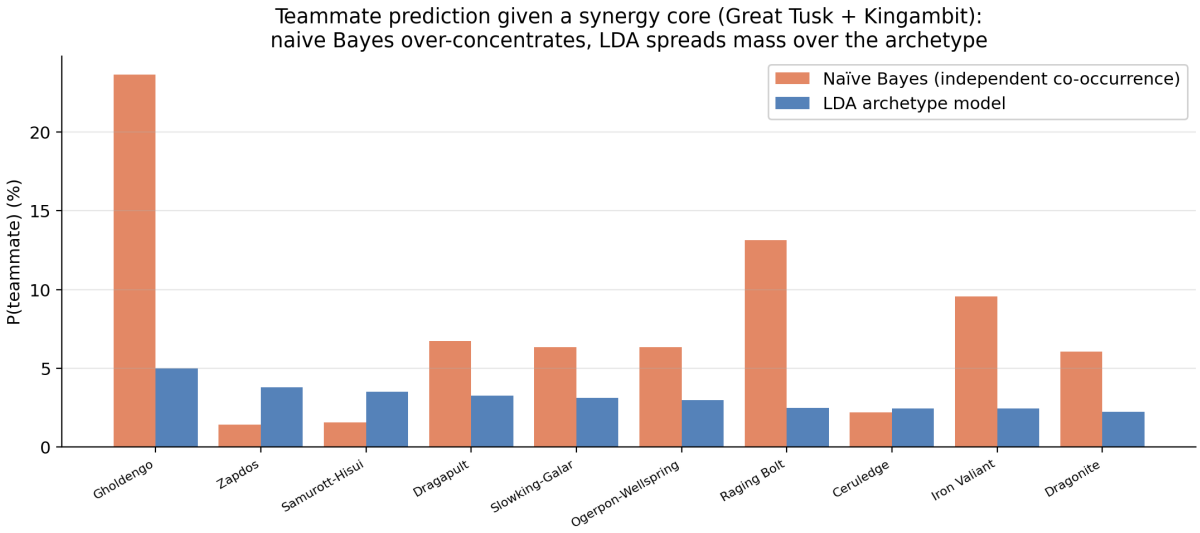

Figure 5.9: Teammate prediction given the revealed core of Great Tusk and Kingambit, under the independent-co-occurrence (naïve Bayes) estimator and the LDA archetype model, for the top ten candidates by archetype score. The naïve estimator double-counts the strongly correlated evidence of the two revealed species and concentrates nearly 24% of the mass on Gholdengo; the archetype model spreads its mass across the plausible members of the implied archetype, yielding a calibrated rather than overconfident distribution.

5.3.5. Evidence conditioning

Once base distributions are obtained, they are updated with information observed during the battle. *Confirmed fields* (item, ability, tera type), once directly observed, collapse the corresponding distribution to a point mass. *Positive move evidence* re-weights the remaining candidate moves by how often each co-occurs with the already-observed moves, computed in log-space to avoid numerical underflow when conditioning on several moves at once. *Negative move evidence* attenuates moves that rarely co-occur with the observed set (capped at an 80% reduction), after which the distribution is renormalized. As a result, predictions become sharper as the battle progresses and more is revealed.

5.3.6. Backward EV inference from damage

When PolimiBot records a concrete damage event — the exact HP percentage dealt by a known move from a known attacker (Section 5.2.4) — this is implicit evidence about the defender's defensive EVs. The damage-inference step re-weights the EV-spread distribution so that candidate spreads which would produce damage outside the observed

range receive lower posterior weight, narrowing the uncertainty about the opponent's investment.

5.3.7. The per-turn prediction pipeline

Each turn, PolimiBot requests the component probabilities for the opponent's active Pokémon, and the inference runs in sequence: (1) *fuse* the replay distribution with the Smogon prior for each component (Equations (5.1) to (5.2)); (2) *condition* on all observed battle evidence (Section 5.3.5); (3) *backward-update* on damage evidence if available (Section 5.3.6); and (4) *return* sorted probability lists for each component. Because the model is trained offline and only fast inference runs at battle time, this adds negligible latency to the per-turn loop.

5.4. Integrating the predictor into the agent

The two contributions meet at the opponent-description blocks of the prompt. Each turn, PolimiBot calls the predictor for the opponent's active Pokémon and its likely bench, and consumes the ranked probability lists to complete the picture the language model reasons over.

Two design choices make these predictions safe to rely on. First, because the predictor's confidence scores are calibrated against accuracy (Section 6.2), a high-confidence prediction is correct most of the time, so the language model is given reliable information rather than an arbitrary guess. Second, the predictions are always presented as the opponent's *likely* state, and the Terastallization warning explicitly flags the residual uncertainty when the opponent can still change type. The net effect is that PolimiBot reasons about a complete, four-move, six-Pokémon opponent from the first turn — directly addressing the cluster of hidden-information weaknesses identified in Section 4.2 — while the agent's own damage observations continually feed back to sharpen the beliefs it depends on.

6 | Experiments and Evaluation

This chapter evaluates the two contributions empirically. It first describes the experimental setup — baselines, teams, and metrics (Section 6.1). It then reports four result tracks: the accuracy and calibration of the Bayesian Predictor (Section 6.2); the token-consumption efficiency of PolimiBot relative to PokéChamp (Section 6.3); and battle results against the heuristic baseline (Section 6.4) and against the PokéChamp baseline (Section 6.5). A closing discussion ties the outcomes back to the specific weaknesses of Chapter 4 and is candid about the limits of what the evidence can establish (Section 6.6).

6.1. Experimental setup

All battles were run in a controlled local environment using Pokémon Showdown, in the Generation 9 OU singles format (Section 2.2). PolimiBot was evaluated against two baselines.

Baselines. *AbyssalPlayer* is a deterministic, hand-crafted heuristic bot that uses no language-model reasoning. Each turn it scores the active matchup from type effectiveness, a speed-tier bonus, and the HP differential, and uses that score to decide whether to switch out to a better matchup, to set or remove entry hazards, or to set up with a stat-boosting move; failing those, it selects the attacking move that maximizes a damage heuristic — base power weighted by STAB, the attacker-to-defender stat ratio, accuracy, the expected number of hits, and type effectiveness. Because every one of these rules is evaluated on the current turn alone, it represents a strong but inflexible lower bound. *PokéChamp* [14] is the LLM-based minimax agent described in Section 3.4.2, and is the principal baseline against which the contributions of this thesis are measured.

Teams and matchups. PolimiBot was evaluated across all five of its reference teams, each designed with a distinct strategic style: Teams 1 and 2 are aggressive (hyper-offense and fast breakers respectively), Team 3 is balanced, Team 4 is control-oriented, and Team 5 is a stall team (Table 6.1). Against each opponent, every one of the five PolimiBot

teams was played against each of the five opposing teams, with four games per team-versus-team matchup. This yields 20 games per PolimiBot team and 100 games per opponent.

Table 6.1: The five reference teams and their strategic styles.

Team	Style
Team 1	Hyper-offense
Team 2	Offense
Team 3	Balanced
Team 4	Control
Team 5	Stall

Metrics. Battle performance is reported as *win rate*, overall and broken down by PolimiBot team and by individual matchup. The Bayesian Predictor is evaluated by top-1 and top-3 *accuracy* on species, move, and component prediction, and by the *calibration* of its confidence scores against observed accuracy. Prompt efficiency is reported as the *mean number of tokens per turn*. As noted in Section 2.5, single games are noisy and win rates over four games per matchup carry substantial variance; the per-matchup figures should therefore be read as indicative, and the aggregates as the more reliable summaries. The number of games is held fixed and equal across all conditions, so the comparisons between teams and between opponents are made on a common footing.

6.2. Bayesian Predictor accuracy

The predictor was evaluated on a held-out set of teams from the same `gl_05_26` corpus, comparing the redesigned predictor of Section 5.3 against the previous naïve Bayes baseline across species, move, and component prediction. Before presenting the numbers, it is essential to be precise about what this benchmark can and cannot show.

6.2.1. A caveat on benchmark validity

The two predictors play different roles. The *old* naïve Bayes predictor was used by the Metamon project to fill in missing fields when building the training corpus from partial replay data; the *redesigned* predictor is the one integrated into PolimiBot. The evaluation teams were generated by the Metamon project from replays captured from a third-person spectator perspective, so only publicly visible information — moves actually used, Pokémon actually sent out — was observed, and the remaining unknown fields were

filled in *by the old predictor*. The ground-truth labels are therefore not truly ground truth: they are the old predictor's own outputs. This introduces a systematic bias in the old predictor's favor and makes a fully controlled comparison impossible. Whenever the two predictors disagree, the new one is automatically scored wrong — even when its prediction is, in reality, the better one. The results below should accordingly be read as "roughly comparable performance on a dataset whose ground truth is partially synthetic", not as a clean ablation, and the small deficits the new predictor shows on some tasks are consistent with this bias rather than evidence of a worse model.

6.2.2. Why the redesign is expected to generalize better

On a dataset with real ground-truth team files — teams authored directly by players rather than inferred from partial replays — the redesigned predictor is expected to outperform the naïve Bayes baseline for two structural reasons. First, the team-composition model: the old predictor multiplies independent co-occurrence conditionals (Equation (3.1)), which assumes teammates are chosen independently given an anchor species and double-counts the correlations within a synergistic core, producing overconfident distributions; the LDA archetype model (Section 5.3.4) is the correct generative view and marginalizes properly over unrevealed slots. Second, Smogon prior fusion: the old predictor is driven purely by replay counts and overfits to corpus-specific artifacts, whereas the adaptive fusion of Equations (5.1) to (5.2) pulls predictions for niche, rarely replayed Pokémon toward the true metagame distribution. Neither advantage can surface on a benchmark whose labels are the old predictor's own outputs.

6.2.3. Species prediction

Table 6.2 reports the accuracy of predicting unrevealed teammates given k revealed Pokémon (see also Figure 6.1). The two predictors are essentially tied on top-1 accuracy; the differences on top-3 are small and mixed in sign, consistent with the benchmark bias of Section 6.2.1.

6.2.4. Move prediction

With k moves already revealed, the task is to predict the remaining moves. Table 6.3 (and Figure 6.2) shows that top-1 accuracy is moderate and decreases as more moves are observed, since the fourth slot is often a situational or uncommon pick. The new predictor slightly underperforms here — again consistent with the benchmark bias, since the corpus move distributions were themselves shaped by the old predictor.

Table 6.2: Species prediction accuracy by number of revealed Pokémon k. "Old" is the naïve Bayes baseline, "New" the redesigned predictor; Δ is New − Old in percentage points (pp).

	Top-1			Top-3		
k	Old	New	Δ	Old	New	Δ
1	0.0%	0.0%	+0.0	10.0%	3.3%	−6.7
2	6.7%	6.7%	+0.0	23.3%	20.0%	−3.3
3	13.3%	13.3%	+0.0	26.7%	16.7%	−10.0
4	13.3%	16.7%	+3.3	16.7%	23.3%	+6.7
5	13.3%	6.7%	−6.7	16.7%	16.7%	+0.0

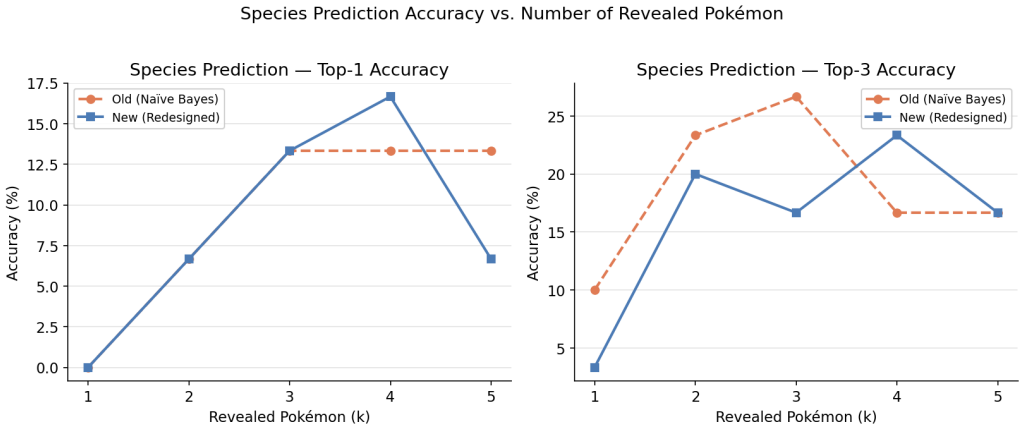

Figure 6.1: Species prediction accuracy as a function of the number of revealed Pokémon, for the old and redesigned predictors.

6.2.5. Component prediction

When all four moves of a Pokémon are already revealed — the most informative scenario — both predictors achieve similar accuracy on the remaining components, as shown in Table 6.4 and Figure 6.3. Ability prediction is near-perfect in both systems because most Pokémon have only one or two viable abilities in Generation 9 OU; item and tera type are harder, owing to their higher variance in practice.

6.2.6. Confidence calibration

A property of the redesigned predictor that the head-to-head accuracy numbers do not capture is that its probability scores are *well calibrated* against actual accuracy.

Each *prediction* here is the predictor's probability distribution over one hidden component

Table 6.3: Move prediction accuracy by number of observed moves k.

	Top-1			Top-3		
k	Old	New	Δ	Old	New	Δ
1	27.8%	28.9%	+1.1	67.2%	65.0%	−2.2
2	32.8%	28.3%	−4.4	62.8%	58.3%	−4.4
3	31.7%	21.7%	−10.0	71.7%	62.8%	−8.9

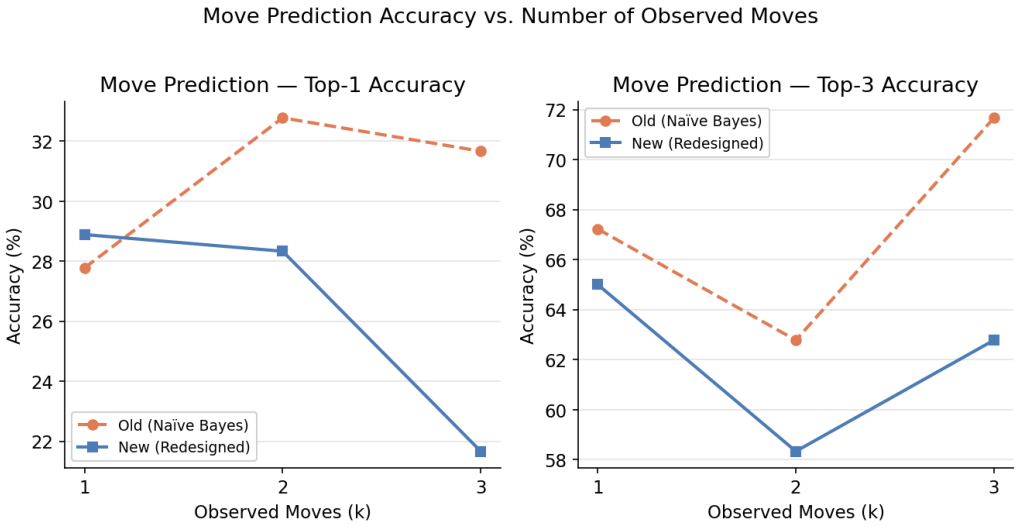

Figure 6.2: Move prediction accuracy as a function of the number of observed moves.

— the item, ability, or Tera type — of a single Pokémon, evaluated with all four of that Pokémon's moves revealed. Before any label is checked against the truth, the predictor classifies its own distribution into one of three *confidence tiers*, based solely on the shape of that distribution: the probability p_1 it assigns to its top-ranked candidate and the gap $p_1 - p_2$ to the second candidate. Writing the thresholds explicitly,

- **Confident** — $p_1 \geq 0.60$ *and* $p_1 - p_2 \geq 0.20$: one candidate is both individually probable and clearly ahead of the field;
- **Likely** — $p_1 \geq 0.35$ *or* $p_1 - p_2 \geq 0.15$ (and not Confident): one candidate stands out, but either not strongly enough or not far enough ahead to be relied on;
- **Uncertain** — neither condition holds: the distribution is flat or its mass is split across several candidates.

The label is thus a statement the model makes about *its own* certainty, set purely from the predicted probabilities and independently of whether the prediction turns out to be correct. The count n reported for each tier is simply the number of these component

Table 6.4: Component prediction accuracy when all four moves are observed.

	Top-1			Top-3		
Component	Old	New	Δ	Old	New	Δ
Ability	93.3%	93.3%	+0.0	100.0%	100.0%	+0.0
Item	54.4%	53.3%	−1.1	85.6%	83.9%	−1.7
Tera Type	41.1%	40.0%	−1.1	77.8%	78.3%	+0.6

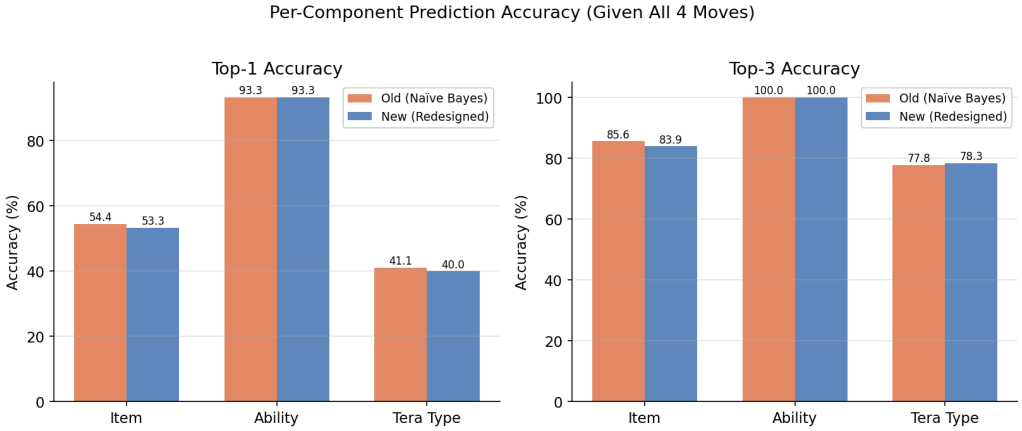

Figure 6.3: Component prediction accuracy (ability, item, tera type) with all four moves observed.

predictions that the predictor placed in that tier; the three counts partition all the component predictions made over the evaluation set, and each tier's accuracy is the fraction of its predictions whose top-ranked candidate matched the true value.

With the tiers defined this way, Table 6.5 and Figure 6.4 show that they line up closely with reality: a "confident" prediction is correct 90.0% of the time, a "likely" one 47.4%, and an "uncertain" one 29.8%. This monotonic relationship is what makes the scores usable by PolimiBot (Section 5.4): a top-1 prediction issued with high confidence — an item or ability, say — can be relied upon roughly nine times in ten, so the language model is given dependable information rather than an unqualified guess. Figure 6.5 summarizes the overall comparison across tasks.

6.2.7. Calibration validated on genuine ground truth

The calibration figures above are computed on the benchmark's partially synthetic labels (Section 6.2.1); it is therefore worth confirming that the same calibration holds *out of sample*, on teams the predictor was never trained on. The predictor was retrained on a 90% split of the `gl_05_26` corpus (125,585 teams, fixed seed) and evaluated on 3,000

Table 6.5: Confidence calibration of the redesigned predictor: accuracy within each confidence tier. The tier is assigned from the predicted probabilities alone (Confident: $p_1 \geq 0.60$ and $p_1 - p_2 \geq 0.20$; Likely: $p_1 \geq 0.35$ or $p_1 - p_2 \geq 0.15$; Uncertain: otherwise). "Count" is the number n of component predictions (item, ability, or Tera type) the predictor assigned to that tier; accuracy is the share of those whose top candidate was correct.

Confidence tier	Accuracy	Count
Confident (high probability)	90.0%	119,990
Likely (moderate probability)	47.4%	85,704
Uncertain (low probability)	29.8%	45,466

teams drawn from the held-out 10%. These are Showdown teambuilder files from the Metamon Teams dataset [7]: real ladder teams whose publicly revealed slots are taken from replays and whose remaining hidden fields are reconstructed by the dataset's usage-statistics-based prediction pipeline. They are therefore not raw player ground truth but they are *disjoint* from the training split, so the experiment measures calibration on unseen data rather than memorization. Predicting the item, ability, and Tera type of every Pokémon given its full moveset yields 53,991 predictions. Figure 6.6 reports the outcome in two forms. The left panel is a *reliability diagram*: predictions are binned by the probability the predictor assigned to its top-1 candidate, and each bin's mean predicted probability is plotted against the accuracy actually observed in that bin. The curve tracks the diagonal of perfect calibration closely over the entire range — predictions issued with, say, 40% confidence are correct close to 40% of the time, and those issued with 95% confidence are correct about 95% of the time. The right panel aggregates the same predictions by the three confidence labels exposed to PolimiBot: *Confident* predictions are correct 88.8% of the time (n = 26,165), *Likely* 46.5% (n = 17,772), and *Uncertain* 30.2% (n = 10,054). These held-out tier accuracies agree closely with those measured on the full benchmark (90.0%, 47.4%, and 29.8%; Table 6.5), indicating that the calibration is stable and generalizes to teams outside the training split rather than being an artifact of fitting. One caveat tempers this check: because the corpus fills hidden fields from usage statistics and the predictor itself fuses a usage-statistics prior (Section 5.3.3), the labels are not fully independent of the model, so the result corroborates the calibration property rather than establishing it against wholly external ground truth — which would require a corpus of hand-built teams with verified hidden fields that the present dataset does not provide.

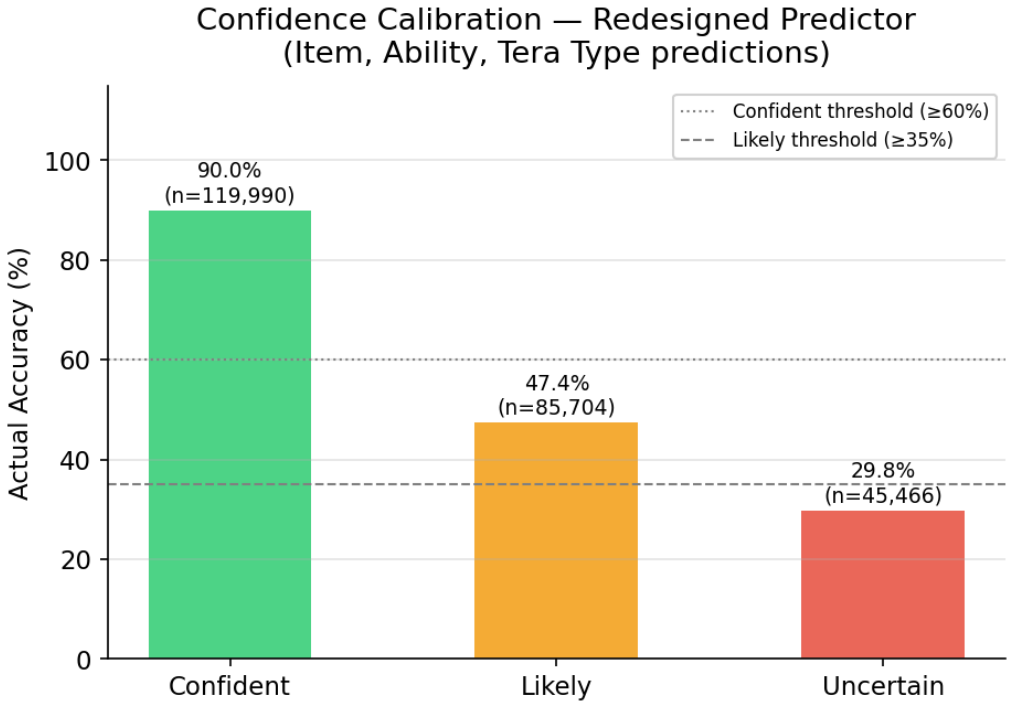

Figure 6.4: Calibration of the redesigned predictor's confidence scores against observed accuracy.

6.2.8. Evidence conditioning sharpens predictions

The same held-out teams allow a direct measurement of the evidence-conditioning mechanism of Section 5.3.5. For each held-out Pokémon, the predictor was queried with $k = 0, 1, 2, 3$ of its moves marked as observed, and the Shannon entropy of the resulting distribution over the remaining moves was recorded ($n = 17{,}997$ Pokémon per condition). Figure 6.7 shows the mean entropy falling monotonically from 3.28 bits with no observed moves to 3.09, 2.73, and finally 2.09 bits as one, two, and three moves are revealed — a reduction of more than a third, equivalent to shrinking the effective set of plausible remaining moves from roughly ten ($2^{3.28}$) to about four ($2^{2.09}$). Two qualitative features are worth noting. First, the largest single drop occurs between the second and third observed move: move slots are strongly correlated through the set's role (an offensive pivot set implies different fillers than a defensive one), so each additional observation disambiguates the underlying set archetype as well as removing one candidate. Second, the growing spread of the per-Pokémon entropies at higher k reflects genuine heterogeneity across species — for many Pokémon three known moves pin the fourth almost completely, while for species with a genuinely free final slot substantial uncertainty rightly remains. This confirms, quantitatively and on held-out teams, the design intent of Section 5.3.5: the predictor's

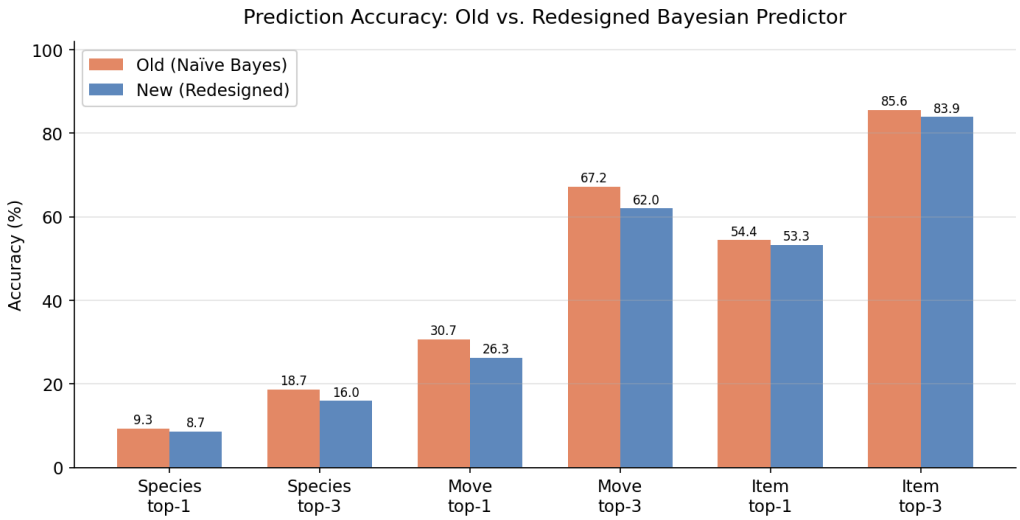

Figure 6.5: Overall comparison of the old and redesigned predictors across the prediction tasks.

output distributions sharpen as the battle progresses and evidence accumulates.

6.3. Token-consumption efficiency

A key architectural difference between PolimiBot and PokéChamp is how the battle history is represented in the prompt (Section 4.1). PokéChamp prepends a natural-language transcript of the last sixteen turns to every prompt; the window is fixed, so its token cost is roughly constant once a battle exceeds sixteen turns, but the transcript still accounts for a significant fraction of the prompt, worst of all in the early-to-mid game and in long stall matches. PolimiBot logs no history at all: it encodes the current state exhaustively in the structured blocks of Section 5.2.1, pre-computing every derived quantity offline, so prompt size is roughly constant across turns regardless of game length.

Table 6.6 reports the mean number of tokens per turn for the two systems. PolimiBot averages 7,616.8 tokens per turn against PokéChamp's 9,399.7, a reduction of 1,782.9 tokens, or 19.0%. Because the saving comes from not carrying a growing event log, the gap widens further in long games. This is a more principled representation: rather than asking the model to re-derive the current board from a long log, PolimiBot computes the relevant derived quantities — effective speed, turns-to-KO, hazard entry cost — and presents them directly, and the reduction translates directly into lower API cost.

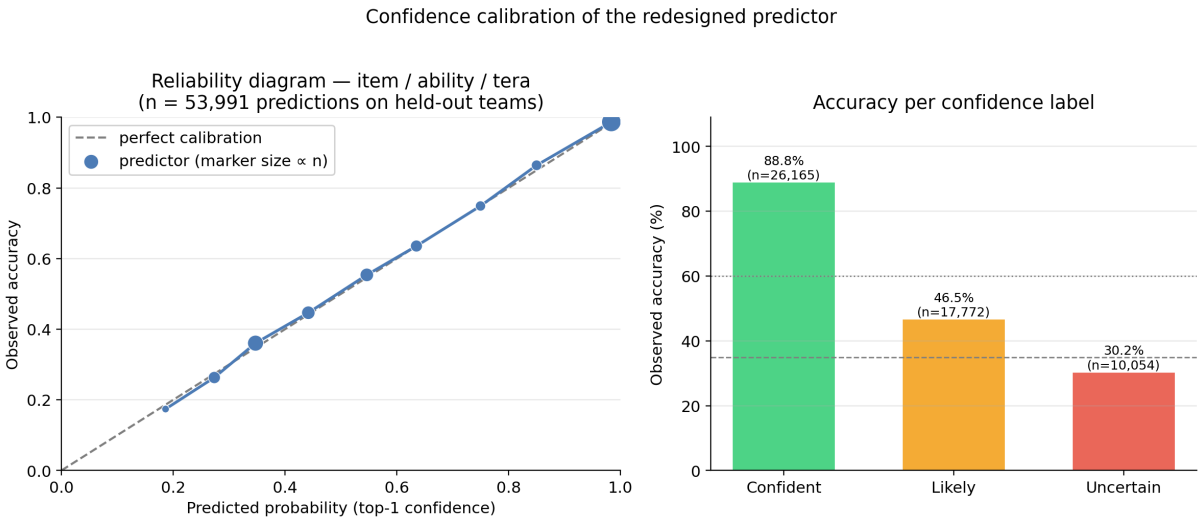

Figure 6.6: Confidence calibration of the redesigned predictor measured on held-out teams: 53,991 item, ability, and Tera-type predictions for 3,000 `gl_05_26` teams disjoint from the training split. Left: reliability diagram of observed accuracy against predicted top-1 probability (marker size proportional to bin population); the dashed diagonal is perfect calibration. Right: observed accuracy within each of the three confidence labels exposed to the agent, with the label thresholds of the summary layer drawn as horizontal reference lines.

Table 6.6: Mean prompt tokens per turn for PolimiBot and PokéChamp.

System	Mean tokens per turn
PokéChamp	9,399.7
PolimiBot	7,616.8
Reduction	−1,782.9 (−19.0%)

6.4. Battle results against AbyssalPlayer

Against the heuristic AbyssalPlayer baseline, PolimiBot won 89 of 100 games, an overall win rate of 89.0%, with no ties. Table 6.7 breaks the result down by PolimiBot team, and Figure 6.8 plots the win rate across all team matchups; the complete per-matchup table is given in Chapter A.

The win rate declines monotonically from aggressive to defensive team styles, from 100.0% for the hyper-offense team down to 75.0% for the stall team. This gradient is consistent with the nature of LLM decision-making: the model excels at identifying immediate tactical opportunities — "I can KO the opponent in one turn with this move" — but has

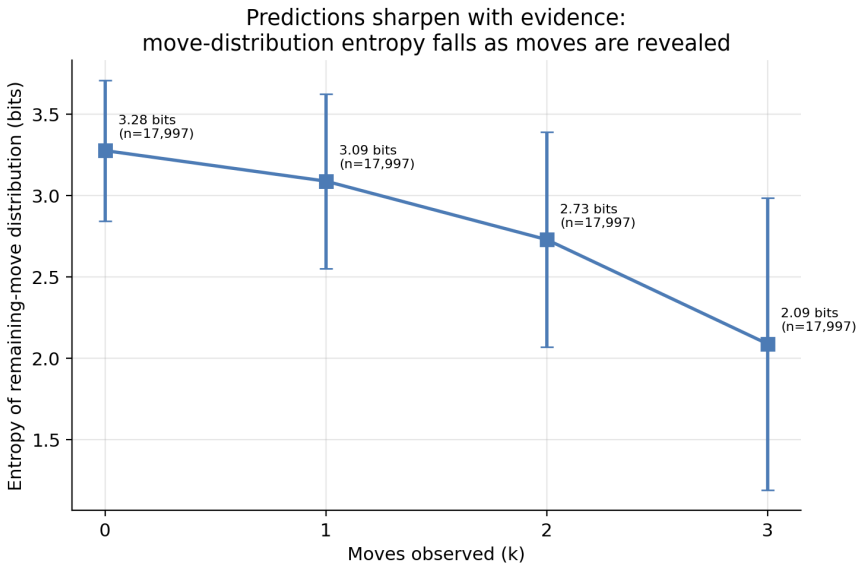

Figure 6.7: Effect of evidence conditioning, measured on held-out teams: mean Shannon entropy of the predicted distribution over a Pokémon's remaining moves as a function of the number of moves already observed (n = 17,997 per condition; error bars give one standard deviation). Entropy decreases monotonically from 3.28 to 2.09 bits, i.e. the effective number of plausible remaining moves shrinks from roughly ten to about four as three moves are revealed.

less leverage on stall teams, where winning requires sustained positional play over many turns, a longer planning horizon that is harder to express within a single-turn prompt. Even at its weakest, however, PolimiBot wins three games in four against a baseline that is far from trivial: AbyssalPlayer backs its damage-maximizing move heuristic with hand-crafted matchup estimation and dedicated switch-out, entry-hazard, and set-up logic (Section 6.1). What it lacks is any model of how the game will unfold — every one of those rules fires on the current turn alone — and it is on exactly that horizon that PolimiBot's per-turn tactical precision compounds into a decisive edge.

6.5. Battle results against PokéChamp

Against the PokéChamp baseline — the strongest published LLM-based agent, and the system whose weaknesses motivated this work (Chapter 4) — PolimiBot won 68 of 100 games, an overall win rate of 68.0%, again with no ties. Table 6.8 gives the breakdown by team and Figure 6.9 plots the win rate across all matchups; the complete per-matchup table appears in Chapter A.

Winning 68% of games against a strong LLM baseline is a meaningful margin and indi-

Table 6.7: PolimiBot win rate against AbyssalPlayer, by team (20 games each).

Team	Style	Wins	Losses	Win rate
Team 1	Hyper-offense	20	0	100.0%
Team 2	Offense	19	1	95.0%
Team 3	Balanced	18	2	90.0%
Team 4	Control	17	3	85.0%
Team 5	Stall	15	5	75.0%
Overall		**89**	**11**	**89.0%**

Table 6.8: PolimiBot win rate against PokéChamp, by team (20 games each).

Team	Style	Wins	Losses	Win rate
Team 1	Hyper-offense	16	4	80.0%
Team 2	Offense	14	6	70.0%
Team 3	Balanced	13	7	65.0%
Team 4	Control	13	7	65.0%
Team 5	Stall	12	8	60.0%
Overall		**68**	**32**	**68.0%**

cates a real gain from the added components. The head-to-head result is corroborated by an indirect comparison through the common heuristic opponent: against AbyssalPlayer, PokéChamp achieves an 84% win rate, below the 89% PolimiBot reaches on the same baseline (Section 6.4), so both the direct and the indirect measurements place PolimiBot ahead. The same team-style gradient seen against AbyssalPlayer is present, though compressed: from 80.0% for the hyper-offense team to 60.0% for the stall team. The aggressive teams benefit most from the per-turn tactical precision that the merger and the ground-truth damage calculator provide, whereas the stall team's advantage is narrower because long-horizon positional decisions are difficult for any single-turn reasoning architecture — a limitation shared by both agents, and one revisited in Section 6.6 and Chapter 7. Because both agents are LLM players subject to the same horizon limitation, the consistent margin in PolimiBot's favor is most naturally attributed to the architectural differences of Chapter 5 rather than to team strength alone, since every PolimiBot team faced every PokéChamp team an equal number of times.

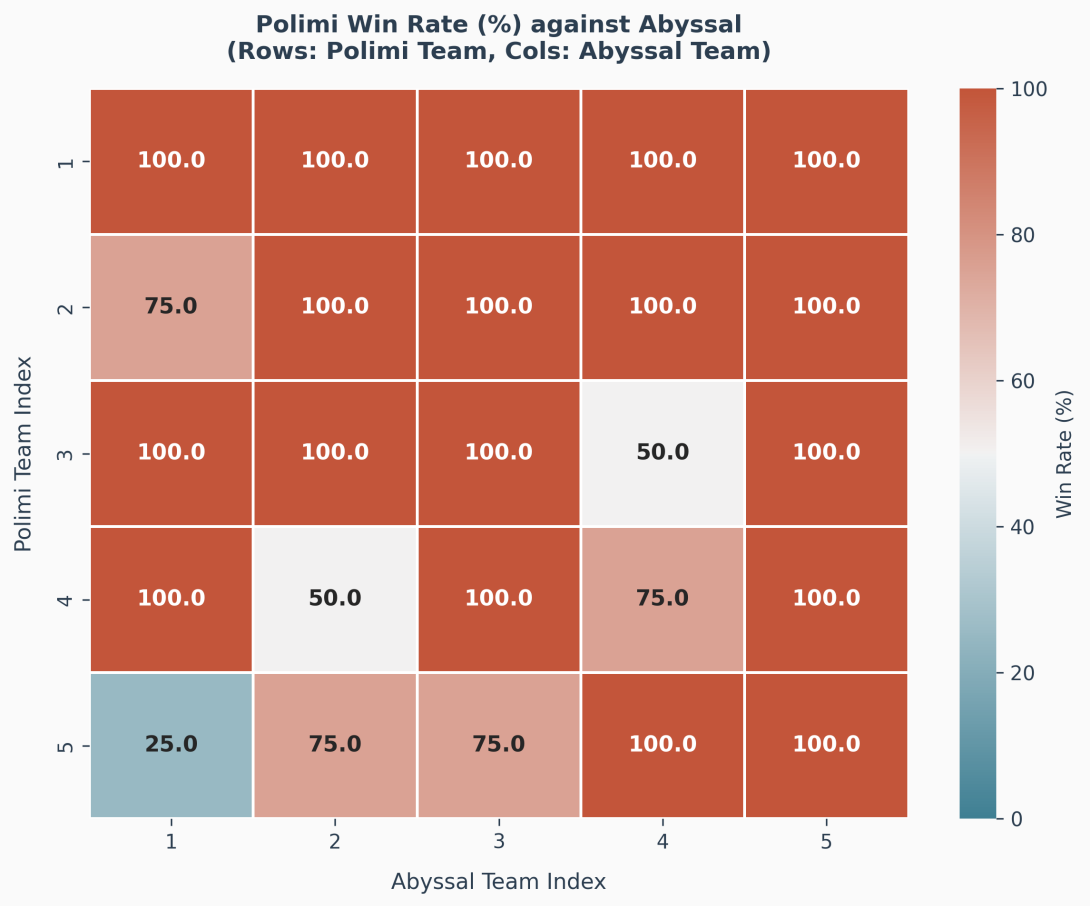

Figure 6.8: PolimiBot win rate against AbyssalPlayer across all team matchups (4 games each, 100 total).

6.6. Discussion

Taken together, the four result tracks support the central claim of the thesis — that a carefully scaffolded LLM agent can outplay a strong LLM baseline — while leaving some of the design's individual claims established more directly than others.

What the results establish. The token-efficiency result (Section 6.3) directly confirms the improvement targeted by weakness 1 of Section 4.1: replacing the sixteen-turn transcript with a stateful current-state encoding cuts mean prompt size by 19.0%, with the gap widening in long games. The predictor evaluation (Section 6.2) speaks to the hidden-information weaknesses of Section 4.2 (numbers 2, 9, 10, and 11): the redesigned predictor matches the baseline's accuracy on a benchmark biased against it, and — more importantly for its use in the agent — its confidence scores are well calibrated, so the predictions injected into the prompt (Section 5.4) are dependable when issued with high

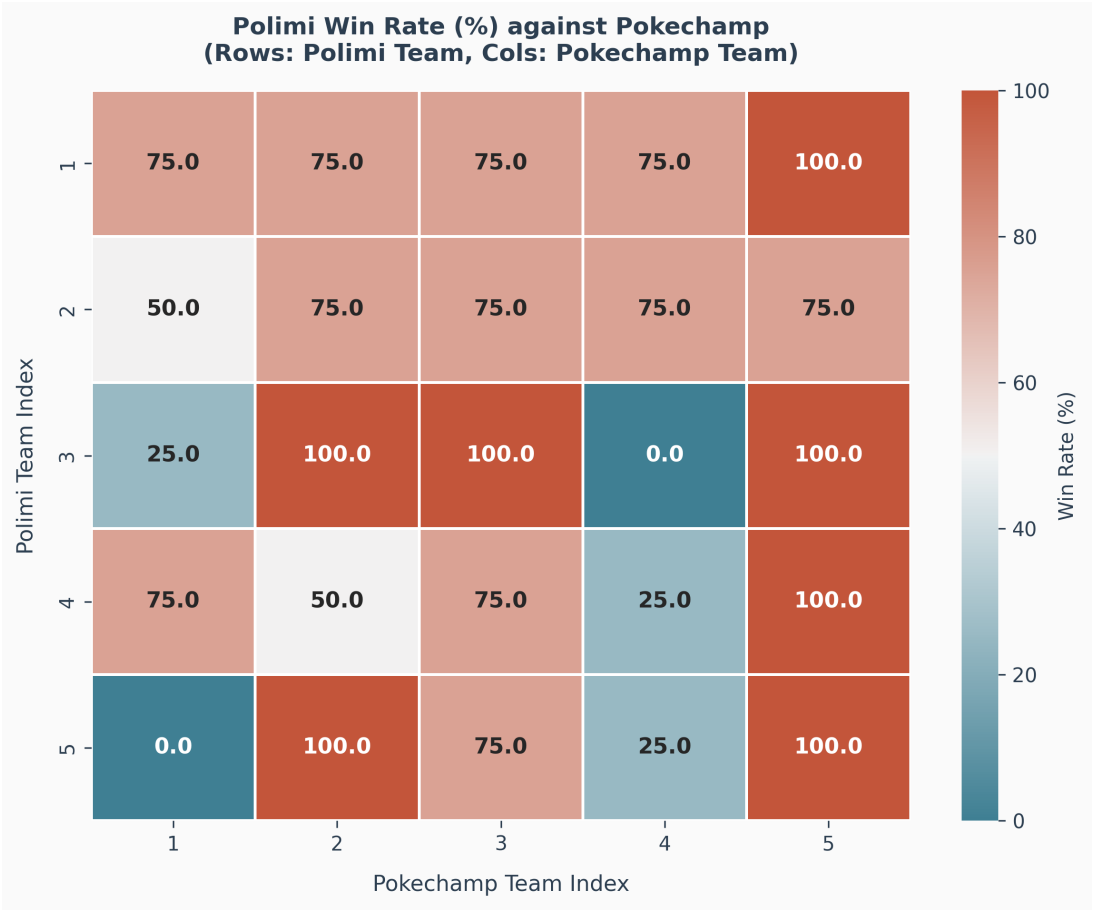

Figure 6.9: PolimiBot win rate against PokéChamp across all team matchups (4 games each, 100 total).

confidence. The head-to-head battle results (Sections 6.4 to 6.5) show that the full system, combining these components with the decomposed decision flow and ground-truth damage calculation (the weaknesses of Sections 4.3 and 4.4), wins 89% of games against the heuristic baseline and 68% against PokéChamp.

What the results do not isolate. The win-rate numbers are *end-to-end*: they measure the combined effect of every component at once. They do not, on their own, attribute the margin to any single improvement — the three-prompt flow, the calibrated predictor, the `@smogon/calc` grounding, the full speed-modifier chain, or the per-switch hazard costs. A per-component ablation, removing one element at a time and re-measuring win rate, would be needed to apportion credit, and was not part of the present evaluation; it is identified as future work in Chapter 7. Two further caveats deserve emphasis. First, the predictor comparison rests on a benchmark whose ground truth is partially the old predictor's own output (Section 6.2.1), so the expected advantage of the LDA and

Smogon-fusion design (Section 6.2.2) remains an argued, not a measured, result. Second, as stressed in Section 2.5 and Section 6.1, four games per matchup carry substantial variance; the per-matchup cells should be read as indicative, and only the aggregates (89% and 68% over 100 games each) treated as reliable.

The team-style gradient. The most consistent qualitative finding is the monotonic decline in win rate from aggressive to defensive teams, present against both baselines. It reflects a genuine limitation of single-turn LLM reasoning rather than a flaw specific to PolimiBot: tactical, KO-driven decisions are easy to express in one prompt, whereas the long-horizon positional play that stall teams demand is not. That the gradient is *compressed* against PokéChamp — a 20-point spread there versus 25 points against AbyssalPlayer — is consistent with both LLM agents sharing this horizon limitation, so that the contest between them turns more on per-turn execution, where PolimiBot's architecture has the advantage.

7 | Conclusions and Future Work

This thesis set out to test whether the generalist knowledge of a large language model can be harnessed, with the right scaffolding, to play competitive Pokémon well. It answered the question by building and evaluating two complementary contributions: PolimiBot, an LLM-driven battle agent, and the Bayesian Predictor, a statistical model of the hidden opponent state.

7.1. Summary of contributions

Both contributions were designed as direct responses to concrete weaknesses in PokéChamp, the strongest existing LLM-based agent (Chapter 4). PolimiBot replaces a token-heavy history transcript with an exhaustive encoding of the current state, grounds the model in exact turns-to-KO figures from a canonical damage calculator, computes the full speed-modifier chain and per-switch hazard costs, and — in place of a single conflated decision call — routes each turn through a three-prompt flow whose merger adjudicates move-versus-switch with objective numbers rather than an uncalibrated score (Section 5.2). The Bayesian Predictor supplies the hidden-state beliefs the agent reasons over, fusing replay frequencies with metagame usage statistics, modeling team composition with a latent-archetype topic model, and conditioning on evidence — including the agent's own observed damage — revealed during play (Section 5.3).

Empirically (Chapter 6), PolimiBot won 89% of games against a heuristic baseline and 68% against PokéChamp, while reducing mean prompt size by 19.0%. The predictor matched the naïve Bayes baseline's accuracy on a benchmark biased against it, and its confidence scores proved well calibrated — a "confident" prediction is correct 90% of the time — which is what makes them usable inside the agent's prompt.

7.2. Limitations

Four limitations bound these conclusions. First, the battle results are end-to-end and do not isolate the contribution of any single design element; the margin over PokéChamp

is attributable to the architecture as a whole rather than to a named component. Second, the predictor benchmark's ground truth is partially the old predictor's own output (Section 6.2.1), so the expected generalization advantage of the LDA and Smogon-fusion design remains argued rather than measured. Third, the most consistent qualitative finding — the monotonic decline in win rate from aggressive to defensive teams — reflects a real limitation of single-turn LLM reasoning on the long horizons that stall play demands. The per-matchup figures also carry substantial variance over four games each. Fourth, and more practically, evaluation is itself expensive: every game drives a language model through dozens of turns, so each prompt and each chain of reasoning carries a real monetary cost in tokens. Measuring whether a design change actually helps requires playing out enough games to overcome the variance just described, which couples statistical power directly to expenditure and is the main reason the win-rate figures here rest on only four games per matchup. This cost is a genuine barrier to iterating on the agent.

7.3. Future work

Several directions follow naturally. The most immediate is a *per-component ablation* that removes one element of PolimiBot at a time — the three-prompt flow, the calibrated predictor, the ground-truth damage calculator, the speed-modifier chain — and re-measures win rate, so that credit can be apportioned among the design choices. A second is to evaluate the predictor on a dataset of *real, player-authored teams*, where its structural advantages over naïve Bayes (Section 6.2.2) could be measured directly rather than argued. A third is to collect per-game data and report *confidence intervals*, in line with the good practice for a high-variance game noted in Section 2.5, and to scale the evaluation to more games and to the live Showdown ladder, where a rating such as GXE would situate the agent against human players. A fourth direction targets the evaluation bottleneck itself: instead of estimating strength from many full — and costly — games, the agent could be probed on a curated set of *tactical puzzles*, in the spirit of chess puzzles. Each puzzle would place the agent in a position narrowed to a tunnel where only one line leads to victory, typically resolving within a couple of turns, so that evaluation concentrates on exactly the critical moments that reveal what a bot does and does not handle — replacing thousands of noisy, expensive games with a small number of decisive, reproducible tests. Grouping the puzzles by the kind of decision they demand — a forced switch, a set-up window, a sacrifice, a tight damage race — would turn the benchmark into a diagnostic that pinpoints a bot's specific weaknesses rather than reporting a single aggregate win rate. A fifth direction concerns the language model itself: every result here uses a frozen, general-purpose model steered by hand-built scaffolding, whereas the strongest agents in

adjacent strategic games invest instead in *post-training* the weights (Section 3.3.3). The simplest version is supervised fine-tuning on the games of expert human players, so that competitive intuition is carried in the weights rather than supplied entirely through the prompt. On its own, however, supervised fine-tuning tends to memorize the demonstrated lines and to forget how to generalize to the out-of-distribution positions that decide hard matchups, which points toward *reinforcement-learning fine-tuning* from the win/loss signal the game already provides. Pokémon is in fact a favorable setting for it: because the base model starts with a working semantic grasp of the rules, it produces enough winning trajectories early on to yield a dense reward signal, sidestepping the sparse-reward exploration that stalls training from scratch. The classical route, proximal policy optimization [22] under a divergence penalty that guards against reward hacking, is costly because it runs several models at once; lighter alternatives — direct preference optimization [20], which learns from won-versus-lost game pairs without a separate reward model, and group-relative optimization [23], which estimates advantages across a sampled group of moves and so drops the value network — fit the token-cost constraints of Section 7.2 better. A natural study would fine-tune the backbone this way while keeping the surrounding architecture and the Bayesian Predictor fixed, isolating how much a learned playing style adds on top of the prompted generalist; the calibrated predictor is in principle complementary, since it could continue to supply the hidden-state belief that the fine-tuned policy reasons over. Finally, the team-style gradient points to the central open problem for turn-by-turn LLM agents: extending the architecture with an explicit notion of a multi-turn plan or win condition, so that the long-horizon positional play required by stall teams becomes expressible. Addressing it would close the gap between the tactical strength demonstrated here and the strategic depth that the hardest matchups demand.

Bibliography

[1] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, Mar. 2003. ISSN 1532-4435.

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

[4] M. Campbell, A. J. Hoane, and F.-h. Hsu. Deep blue. *Artificial Intelligence*, 134 (1–2):57–83, 2002. doi: 10.1016/S0004-3702(01)00129-1.

[5] Y. Dong, C. F. Ruan, Y. Cai, R. Lai, Z. Xu, Y. Zhao, and T. Chen. Xgrammar: Flexible and efficient structured generation engine for large language models, 2025. URL `https://arxiv.org/abs/2411.15100`.

[6] M. F. A. R. D. T. (FAIR)†, A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu, A. P. Jacob, M. Komeili, K. Konath, M. Kwon, A. Lerer, M. Lewis, A. H. Miller, S. Mitts, A. Renduchintala, S. Roller, D. Rowe, W. Shi, J. Spisak, A. Wei, D. Wu, H. Zhang, and M. Zijlstra. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022. doi: 10.1126/science.ade9097. URL `https://www.science.org/doi/abs/10.1126/science.ade9097`.

[7] J. Grigsby. Metamon teams. `https://huggingface.co/datasets/jakegrigsby/metamon-teams`, 2025. Hugging Face dataset; `gl_05_26` split.

[8] J. Grigsby, Y. Xie, J. Sasek, S. Zheng, and Y. Zhu. Human-level competitive pokémon

via scalable offline reinforcement learning with transformers, 2025. URL https://arxiv.org/abs/2504.04395.

[9] Z. Guan, X. Kong, F. Zhong, and Y. Wang. Richelieu: self-evolving llm-based agents for ai diplomacy. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

[10] S. Hao, Y. Gu, H. Ma, J. Hong, Z. Wang, D. Wang, and Z. Hu. Reasoning with language model is planning with world model. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507/.

[11] S. Hu, T. Huang, and L. Liu. Pokellmon: A human-parity agent for pokemon battles with large language models, 2024. URL https://arxiv.org/abs/2402.01118.

[12] S. Hu, T. Huang, G. Liu, R. R. Kompella, F. Ilhan, S. F. Tekin, Y. Xu, Z. Yahn, and L. Liu. A survey on large language model-based game agents, 2026. URL https://arxiv.org/abs/2404.02039.

[13] D. Huang and S. Lee. A self-play policy optimization approach to battling pokémon. In *2019 IEEE Conference on Games (CoG)*, page 1–4. IEEE Press, 2019. doi: 10.1109/CIG.2019.8848014. URL https://doi.org/10.1109/CIG.2019.8848014.

[14] S. Karten, A. L. Nguyen, and C. Jin. Pokéchamp: an expert-level minimax language agent. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

[15] M. J. Khan and G. Sukthankar. Sc-phi2: A fine-tuned small language model for starcraft ii macromanagement tasks, 2024. URL https://arxiv.org/abs/2409.18989.

[16] A. Kumar, V. Zhuang, R. Agarwal, Y. Su, J. D. Co-Reyes, A. Singh, K. Baumli, S. Iqbal, C. Bishop, R. Roelofs, L. M. Zhang, K. McKinney, D. Shrivastava, C. Paduraru, G. Tucker, D. Precup, F. Behbahani, and A. Faust. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

[17] S. Lange, T. Gabel, and M. Riedmiller. *Batch Reinforcement Learning*, pages 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-

27645-3. doi: 10.1007/978-3-642-27645-3_2. URL https://doi.org/10.1007/978-3-642-27645-3_2.

[18] W. Ma, Q. Mi, Y. Zeng, X. Yan, Y. Wu, R. Lin, H. Zhang, and J. Wang. Large language models play starcraft ii: benchmarks and a chain of summarization approach. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

[19] P. Mariglia. Foul play: A pokémon showdown battle bot. https://github.com/pmariglia/foul-play, 2019. Computer software.

[20] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[21] H. Sahovic. Poke-env: pokemon ai in python. https://github.com/hsahovic/poke-env, 2026.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[23] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[24] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

[26] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mas-

tering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

[28] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, D. Silver, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, 2019. doi: 10.1038/s41586-019-1724-z.

[29] Z. Wan, X. Feng, M. Wen, S. M. McAleer, Y. Wen, W. Zhang, and J. Wang. Alphazero-like tree-search can guide large language model decoding and training. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[30] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL `https://arxiv.org/abs/2305.16291`.

[31] J. Wang. Winning at pokémon random battles using reinforcement learning, Feb. 2024. URL `https://hdl.handle.net/1721.1/153888`.

[32] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL `https://arxiv.org/abs/2203.11171`.

[33] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

[34] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, H. Aghabozorgi, L. Barrett, R. Douglas, D. Whitehead, P. Dürr, P. Stone, M. Spranger, and H. Kitano. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022. doi: 10.1038/s41586-021-04357-7.

[35] K. Xu, J. Chai, S. Li, Y. Fu, Y. Zhu, and D. Zhao. Dipllm: Fine-tuning llm for strategic decision-making in diplomacy, 2025. URL https://arxiv.org/abs/2506.09655.

[36] Z. Xu, C. Yu, F. Fang, Y. Wang, and Y. Wu. Language agents with reinforcement learning for strategic play in the werewolf game. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[37] Z. Xu, W. Gu, C. Yu, Y. Wu, and Y. Wang. Learning strategic language agents in the werewolf game with iterative latent space policy optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

[38] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[39] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

A | Supporting Material

This appendix collects supporting material referenced from the main text: the full point-by-point comparison between PokéChamp and PolimiBot (Section A.1), and the complete per-matchup battle results (Section A.2).

A.1. PokéChamp weaknesses and PolimiBot improvements

Table A.1 reproduces, in full, the fifteen weaknesses summarized by theme in Table 4.1, each paired with the corresponding improvement in PolimiBot or the Bayesian Predictor.

Table A.1: Full comparison of PokéChamp weaknesses and PolimiBot improvements.

#	Weakness in PokéChamp	Improvement in PolimiBot
1	Sixteen-turn raw history transcript inflates tokens	Stateful current-state prompt; −19% tokens
2	Naïve static move prediction for opponents	Full Bayesian predictor with evidence conditioning
3	Move and switch decision conflated in one call	Three-prompt decomposed flow with Merger Judge
4	Hard-coded tera heuristic misses most tera plays	Model decides, informed by authored per-Pokémon notes
5	No per-Pokémon authored strategic context	`teamN.json` strategy notes injected in system prompt
6	Inaccurate Python damage calculator; ratio-based power	`@smogon/calc` ground truth; turns-to-KO output
7	Incomplete speed-modifier chain	Full modifier chain with explicit move-order verdict

continued on next page

Table A.1 – continued

#	Weakness in PokéChamp	Improvement in PolimiBot
8	Generic hazard text; no per-switch HP cost	Exact post-entry effective HP for each switch candidate
9	Unrevealed opponent Pokémon absent from prompt	LDA archetype model predicts bench composition
10	Opponent item, ability, tera type unknown / empty	Bayesian predictor fills in top-1 predictions
11	No damage-evidence tracking for EV inference	Backward EV update on each observed hit
12	Bench move context: type multiplier only, no KO estimate	KO-turn estimates for all bench Pokémon
13	Self-consistency voting is architecturally unstructured	Per-sub-agent majority voting with "Nothing" short-circuit
14	No dedicated pivot-move forced-switch flow	Dedicated pivot path with KO-race context
15	Uncalibrated, cross-call minimax leaf scores; mixed-scale fallback	Merger makes a binary choice anchored to objective KO-turn numbers

A.2. Complete per-matchup battle results

Table A.2 and Table A.3 give the win rate for every PolimiBot-team versus opponent-team combination (4 games each), summarized by team in Table 6.7 and Table 6.8 respectively.

Table A.2: PolimiBot vs AbyssalPlayer: games and win rate per team matchup.

Polimi Team	Abyssal Team	Games	Polimi Wins	Opp. Wins	Win rate
Team 1	Team 1	4	4	0	100.0%
Team 1	Team 2	4	4	0	100.0%
Team 1	Team 3	4	4	0	100.0%
Team 1	Team 4	4	4	0	100.0%
Team 1	Team 5	4	4	0	100.0%
Team 2	Team 1	4	3	1	75.0%
Team 2	Team 2	4	4	0	100.0%
Team 2	Team 3	4	4	0	100.0%
Team 2	Team 4	4	4	0	100.0%
Team 2	Team 5	4	4	0	100.0%
Team 3	Team 1	4	4	0	100.0%
Team 3	Team 2	4	4	0	100.0%
Team 3	Team 3	4	4	0	100.0%
Team 3	Team 4	4	2	2	50.0%
Team 3	Team 5	4	4	0	100.0%
Team 4	Team 1	4	4	0	100.0%
Team 4	Team 2	4	2	2	50.0%
Team 4	Team 3	4	4	0	100.0%
Team 4	Team 4	4	3	1	75.0%
Team 4	Team 5	4	4	0	100.0%
Team 5	Team 1	4	1	3	25.0%
Team 5	Team 2	4	3	1	75.0%
Team 5	Team 3	4	3	1	75.0%
Team 5	Team 4	4	4	0	100.0%
Team 5	Team 5	4	4	0	100.0%

Table A.3: PolimiBot vs PokéChamp: games and win rate per team matchup.

Polimi Team	PokéChamp Team	Games	Polimi Wins	Opp. Wins	Win rate
Team 1	Team 1	4	3	1	75.0%
Team 1	Team 2	4	3	1	75.0%
Team 1	Team 3	4	3	1	75.0%
Team 1	Team 4	4	3	1	75.0%
Team 1	Team 5	4	4	0	100.0%
Team 2	Team 1	4	2	2	50.0%
Team 2	Team 2	4	3	1	75.0%
Team 2	Team 3	4	3	1	75.0%
Team 2	Team 4	4	3	1	75.0%
Team 2	Team 5	4	3	1	75.0%
Team 3	Team 1	4	1	3	25.0%
Team 3	Team 2	4	4	0	100.0%
Team 3	Team 3	4	4	0	100.0%
Team 3	Team 4	4	0	4	0.0%
Team 3	Team 5	4	4	0	100.0%
Team 4	Team 1	4	3	1	75.0%
Team 4	Team 2	4	2	2	50.0%
Team 4	Team 3	4	3	1	75.0%
Team 4	Team 4	4	1	3	25.0%
Team 4	Team 5	4	4	0	100.0%
Team 5	Team 1	4	0	4	0.0%
Team 5	Team 2	4	4	0	100.0%
Team 5	Team 3	4	3	1	75.0%
Team 5	Team 4	4	1	3	25.0%
Team 5	Team 5	4	4	0	100.0%

B | Complexity of Single Battles

This appendix derives the order-of-magnitude figures quoted in Section 2.1: the per-turn branching factor, the size of a player's information set, and the number of legal team configurations, all for the Generation 9 OU *singles* format. In this format exactly one Pokémon per side is on the field, and — with team preview from the fifth generation onward (Section 2.2) — all six of each player's Pokémon are revealed at the start, so no bench selection is hidden. The figures are deliberately loose: they are intended to convey scale, not an exact count.

B.1. Per-turn branching factor

Consider first the nondeterminism of a single move. Damage is sampled from one of sixteen rolls, and a move can additionally hit normally, miss, land a critical hit, trigger a secondary effect, or both crit and trigger a secondary effect. Considering only these possibilities gives

$$16 \times 5 = 80 \tag{B.1}$$

distinguishable outcomes for each damaging move used.

Each Pokémon has up to four moves ($\mathrm{n_moves} = 4$). A player may also switch to one of the (up to) five benched Pokémon ($\mathrm{n_switches} = 5$). When Terastallization is still available, the number of distinguishable (action, outcome) combinations available to one side is

$$2 \cdot 80 \cdot \mathrm{n_moves} + \mathrm{n_switches} = 2 \cdot 80 \cdot 4 + 5 = 645, \tag{B.2}$$

where the factor of 2 accounts for the option to Terastallize or not. Once the Tera option has been spent, this reduces to

$$80 \cdot \mathrm{n_moves} + \mathrm{n_switches} = 80 \cdot 4 + 5 = 325. \tag{B.3}$$

Because both players commit to an action simultaneously (Section 2.1), the worst-case per-turn branching factor — both sides still holding their Tera option — is on the order

of

$$645^2 = 4.16 \times 10^5 \approx 10^5. \tag{B.4}$$

B.2. Size of a player's information set

Partial observability makes many true states indistinguishable to a player. With team preview, the six species of each side are known, so the dominant hidden quantity is each opposing Pokémon's *statistical configuration* — the EVs, IVs, and nature that fix its final stats. There are 25 natures (of which 5 are duplicates in effect, leaving 21 distinct), 32 possible values for each of the six IVs, and 510 Effort Value points to distribute over six stats subject to a cap of 252 per stat.

Since EVs affect a stat only in multiples of four, the EV allocation can be formalized as distributing $\lfloor 510/4 \rfloor = 127$ units across six stats with at most $\lfloor 252/4 \rfloor = 63$ units per stat — that is, counting the integer solutions of

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 127, \qquad 0 \le x_i \le 63. \tag{B.5}$$

By the inclusion–exclusion principle for bounded compositions, this yields 246,774,528 spreads. Leaving IVs aside (most competitive Pokémon run the maximum 31 in every stat), the number of realistic stat configurations per Pokémon is

$$21 \cdot 246{,}774{,}528 = 5.182 \times 10^9. \tag{B.6}$$

Because all six Pokémon are brought and revealed, a player's information set — under this stats-only simplification — has size on the order of

$$(21 \cdot 246{,}774{,}528)^{\text{n_mons}} = \left(5.182 \times 10^9\right)^6 = 1.937 \times 10^{58} \approx 10^{58}. \tag{B.7}$$

This estimate is conservative, however: it omits the hidden moves, items, abilities, and Tera types that further enlarge the set, and which the Bayesian Predictor of Chapter 5 is designed to infer.

B.3. Number of team configurations

Finally, consider the size of the design space from which a team is drawn. Of the roughly 937 moves in the game, about 100 are available in a typical species' learnset; a species has up to 3 abilities and may hold one of 223 competitively relevant items, has 2 genders,

a choice of 19 Tera types (the eighteen elemental types plus Stellar), and the 5.182×10^9 stat configurations derived above. The number of configurations for a single Pokémon is therefore

$$\binom{100}{4} \cdot 3 \cdot 223 \cdot 2 \cdot 19 \cdot 5.182 \times 10^9 = 5.166 \times 10^{20}. \tag{B.8}$$

With approximately 750 usable species, the number of six-Pokémon teams is

$$\binom{750}{6} \cdot \left(5.166 \times 10^{20}\right)^6 = 4.604 \times 10^{138} \approx 10^{139}. \tag{B.9}$$

By way of comparison, the initialization space of other well-studied games is far smaller: chess and Go offer only the choice of color, Dota's two teams of five drafting from 126 heroes give $\binom{126}{5}\binom{121}{5} \approx 10^{17}$ configurations, and StarCraft's three races over nine maps give $9 \cdot 3^2 = 81$. The combination of an enormous design space, a large per-turn branching factor, and a vast information set is what makes competitive Pokémon a uniquely demanding testbed.

List of Figures

5.1 PolimiBot's three-prompt decision flow with majority voting. The Switch Sub-Agent can short-circuit the Merger by returning "Nothing". 28
5.2 The hierarchy of statistical models in the Bayesian Predictor. 30
5.3 Predictability of each set component across the 449 Generation 9 OU species in the Smogon usage record. Left: distribution (across species) of the usage share held by the most common choice of each component. Right: distribution of the Shannon entropy of each component's usage distribution. Boxes span the interquartile range with the median marked; whiskers extend to the most extreme non-outlying species. Abilities are close to deterministic, whereas items, EV spreads, and Tera types are intrinsically dispersed. 32
5.4 The adaptive fusion weight λ of Equation (5.1) as a function of the number of replay observations c_{replay}, for three representative Smogon usage counts (note the logarithmic horizontal axis). Sparsely observed species are prior-dominated (λ near 1); abundantly observed species are replay-dominated, with λ clamped at the floor of 0.1 so that the metagame-wide signal is never discarded entirely. 34
5.5 Adaptive fusion in action on the held-item distributions of two species (Smogon prior, replay estimate, and fused result, restricted to the top fused items). Left: Great Tusk, a data-rich species whose fused distribution follows its abundant replay evidence. Right: Pineco, a data-poor species (10 replay observations) whose over-concentrated replay estimate is shrunk toward the broader Smogon prior, which also restores plausible items absent from the sparse replay sample. 35

5.6 Teammate co-occurrence network of the forty most-used Generation 9 OU Pokémon, built from the Smogon teammate statistics. Node size is proportional to usage; edge thickness to co-occurrence weight; colors mark the communities found by greedy modularity maximization. The communities align with the archetypes of competitive parlance, motivating a team-composition model that captures cluster membership explicitly. . . . 36
5.7 The twenty archetypes learned by the LDA model, as the matrix $P(\text{species} \mid \text{archetype})$. Rows are topics, ordered by corpus prevalence (shown in parentheses) and labeled with their three most probable species; columns are the union of each topic's top species, ordered to align signature species with their topic. The block-diagonal structure shows that each archetype concentrates on a small coherent core, recovering recognized team styles (sun, rain, stall, balance) without supervision. 37
5.8 Teammate inference conditioned on a real corpus team as it is progressively revealed (left to right: one, two, and three revealed Pokémon). Bars give the predicted probability of the top eight unrevealed candidates; bars in green mark species that are actually on the team in question. With only the archetype-ambiguous Great Tusk revealed, no true member is ranked highly; once Walking Wake identifies the sun archetype, the true sun core (Ninetales, Torkoal, then Venusaur) climbs into the top predictions. 38
5.9 Teammate prediction given the revealed core of Great Tusk and Kingambit, under the independent-co-occurrence (naïve Bayes) estimator and the LDA archetype model, for the top ten candidates by archetype score. The naïve estimator double-counts the strongly correlated evidence of the two revealed species and concentrates nearly 24% of the mass on Gholdengo; the archetype model spreads its mass across the plausible members of the implied archetype, yielding a calibrated rather than overconfident distribution. 39

6.1 Species prediction accuracy as a function of the number of revealed Pokémon, for the old and redesigned predictors. 44
6.2 Move prediction accuracy as a function of the number of observed moves. . 45
6.3 Component prediction accuracy (ability, item, tera type) with all four moves observed. 46
6.4 Calibration of the redesigned predictor's confidence scores against observed accuracy. 48

6.5 Overall comparison of the old and redesigned predictors across the prediction tasks. 49
6.6 Confidence calibration of the redesigned predictor measured on held-out teams: 53,991 item, ability, and Tera-type predictions for 3,000 `gl_05_26` teams disjoint from the training split. Left: reliability diagram of observed accuracy against predicted top-1 probability (marker size proportional to bin population); the dashed diagonal is perfect calibration. Right: observed accuracy within each of the three confidence labels exposed to the agent, with the label thresholds of the summary layer drawn as horizontal reference lines. 50
6.7 Effect of evidence conditioning, measured on held-out teams: mean Shannon entropy of the predicted distribution over a Pokémon's remaining moves as a function of the number of moves already observed ($n = 17{,}997$ per condition; error bars give one standard deviation). Entropy decreases monotonically from 3.28 to 2.09 bits, i.e. the effective number of plausible remaining moves shrinks from roughly ten to about four as three moves are revealed. 51
6.8 PolimiBot win rate against AbyssalPlayer across all team matchups (4 games each, 100 total). 53
6.9 PolimiBot win rate against PokéChamp across all team matchups (4 games each, 100 total). 54

List of Tables

4.1 The fifteen weaknesses identified in PokéChamp, grouped by theme, each addressed by an element of PolimiBot or the Bayesian Predictor. The full point-by-point comparison is reproduced in Chapter A. 24

5.1 An abridged Smogon usage record for Great Tusk (raw count 668,457; viability ceiling 92), showing the top entries of each component the prior consumes. Percentages are usage shares; the catch-all "Other" bucket is omitted. 31

6.1 The five reference teams and their strategic styles. 42
6.2 Species prediction accuracy by number of revealed Pokémon k. "Old" is the naïve Bayes baseline, "New" the redesigned predictor; Δ is New − Old in percentage points (pp). 44
6.3 Move prediction accuracy by number of observed moves k. 45
6.4 Component prediction accuracy when all four moves are observed. 46
6.5 Confidence calibration of the redesigned predictor: accuracy within each confidence tier. The tier is assigned from the predicted probabilities alone (Confident: $p_1 \geq 0.60$ and $p_1 - p_2 \geq 0.20$; Likely: $p_1 \geq 0.35$ or $p_1 - p_2 \geq 0.15$; Uncertain: otherwise). "Count" is the number n of component predictions (item, ability, or Tera type) the predictor assigned to that tier; accuracy is the share of those whose top candidate was correct. 47
6.6 Mean prompt tokens per turn for PolimiBot and PokéChamp. 50
6.7 PolimiBot win rate against AbyssalPlayer, by team (20 games each). 52
6.8 PolimiBot win rate against PokéChamp, by team (20 games each). 52

A.1 Full comparison of PokéChamp weaknesses and PolimiBot improvements. . 67
A.2 PolimiBot vs AbyssalPlayer: games and win rate per team matchup. 69
A.3 PolimiBot vs PokéChamp: games and win rate per team matchup. 70