

# Pokemon Battle Predictor - Report

Alessandro Gautieri

gautieri.2041850@studenti.uniroma1.it

Tommaso Federici

federici.2214368@studenti.uniroma1.it

Lorenzo Colombini

colombini.1973692@studenti.uniroma1.it

## Abstract

Sistema di predizione per battaglie Pokémon competitive basato su ensemble di gradient boosting models (LightGBM, CatBoost, XGBoost). L'architettura combina feature engineering avanzata (numerose features multi-scala temporali, predizione tipi non visti, interaction features) con strategie di training robuste (10-fold CV, side-swap augmentation, isotonic calibration). L'ottimizzazione Bayesiana parallela degli iperparametri e il weighted averaging ensemble garantiscono generalizzazione ottimale. Performance finale: **84.71% accuracy, 92.14% AUC, 0.355 LogLoss**.

## 1. Feature Engineering (339 features)

### 1.1. Finestre Temporali Multi-Scale

Il battle log viene segmentato in tre finestre ( $w_1$ : turni 1–10,  $w_2$ : 11–20,  $w_3$ : 21–30) per catturare dinamiche early/mid/late game. Per ogni finestra tracciamo:

- **Damage dealt/taken:** pressione offensiva per fase
- **KO e switches:** momentum e controllo del match
- **Status afflictions:** debuff strategici
- **Super-effective hits:** sfruttamento matchup tipo

Questo approccio permette al modello di distinguere strategie aggressive early-game da comebacks nel late-game.

### 1.2. Predizione Tipi Non Visti dell'Avversario

Innovazione chiave: prediciamo i tipi dei Pokémon non ancora rivelati usando la distribuzione globale dei tipi (da predict.csv). Calcoliamo il **vantaggio di tipo atteso** moltiplicando le probabilità dei tipi non visti per i moltiplicatori offensivi del nostro team. La feature `p1_expected_type_advantage_unseen_p2` cattura il potenziale strategico contro Pokémon nascosti. Questo è possibile sapendo che il dataset è composto da battaglie reali usando i primi 151 pokémon, poiché le battaglie in tornei ufficiali hanno una distribuzione tipica dei tipi nei team.

### 1.3. Feature Chiave Aggiuntive

- **Type Coverage Metrics:** super-effective/immune/resist counts per team, identificano punti deboli strutturali
- **Offensive/Defensive Ratio:**  $(Atk+SpA)/(HP+Def+SpD)$  misura bilanciamento team
- **HP Trajectory:** avg\_hp\_pct\_start, avg\_hp\_pct\_end, delta quantificano trade efficiency
- **Interaction Features:** prodotti non-lineari tra feature correlate (es. damage × speed, status × HP\_advantage) che i GBDT non scoprono facilmente da soli

## 2. Training Strategy

### 2.1. Ensemble Architecture

Tre gradient boosting models complementari:

- **LightGBM** (peso 0.431): veloce, ottimo su feature numeriche, gestisce class imbalance con `class_weight='balanced'`
- **CatBoost** (peso 0.284): robusto su feature categoriche, riduce overfitting
- **XGBoost** (peso 0.284): regolarizzazione L1/L2 forte, stabilità predittiva

### 2.2. Cross-Validation e Augmentation

- **10-fold Stratified CV:** preserva distribuzione classi (50/50)
- **Side-Swap Augmentation:** raddoppia il training set invertendo prospettiva P1P2 (matchup speculare con label invertito)
- **Seed Bagging** (3 seeds): media predizioni con seed diversi per ridurre varianza
- **Isotonic Calibration:** per-fold e finale, migliora probabilità calibrate

### 2.3. Ensemble Method: Weighted Average

Grid search su OOF predictions identifica pesi ottimali (0.431, 0.284, 0.284). Superiore a stacking poiché più robusto e generalizza meglio su test set (0.84710 vs 0.84480).

### 3. Hyperparameter Optimization

#### 3.1. Optimizer Parallel

Tre script Optuna indipendenti (`optimizer_lightgbm.py`, `optimizer_cat.py`, `optimizer_xgb.py`) eseguiti in parallelo per 200+ trials ciascuno. Search space ottimizzato:

- **LightGBM**: `num_leaves` (64–256), `learning_rate` (0.01–0.05), `feature_fraction` (0.6–1.0)
- **CatBoost**: `depth` (6–10), `l2_leaf_reg` (1–10), `learning_rate` (0.01–0.05)
- **XGBoost**: `max_depth` (6–10), `eta` (0.01–0.05), `reg_lambda` (0.1–1.0)

**Metric ottimizzata:** LogLoss (direttamente correlata a probabilità calibrate).

#### 3.2. Threshold Optimization

Grid search fine-grained (0.2–0.8, step 0.01) su OOF predictions per massimizzare accuracy. Threshold ottimale:

**0.510** (leggermente sbilanciato verso classe 1 per dataset bilanciato).

### 4. Pipeline End-to-End

```
train.jsonl → Feature Engineering (339) → 10-Fold CV →  
→ LGBM+Cat+XGB (seed bagging) → Isotonic Calibration →  
→ Weighted Average (grid search) → Threshold Optimization →  
→ Ensemble Calibrator → submission.csv
```

#### Punti di forza:

1. **Robustezza**: side-swap + CV averaging elimina overfitting
2. **Scalabilità**: finestre temporali + interaction features cateturano pattern complessi
3. **Generalizzazione**: weighted average supera stacking, ensemble calibration finale
4. **Riproducibilità**: `RANDOM_STATE=42`, configurazione centralizzata in `config.py`

### Risultato Finale

Accuracy **84.71%** @ threshold **0.510** — AUC **92.14%** —  
LogLoss **0.355**