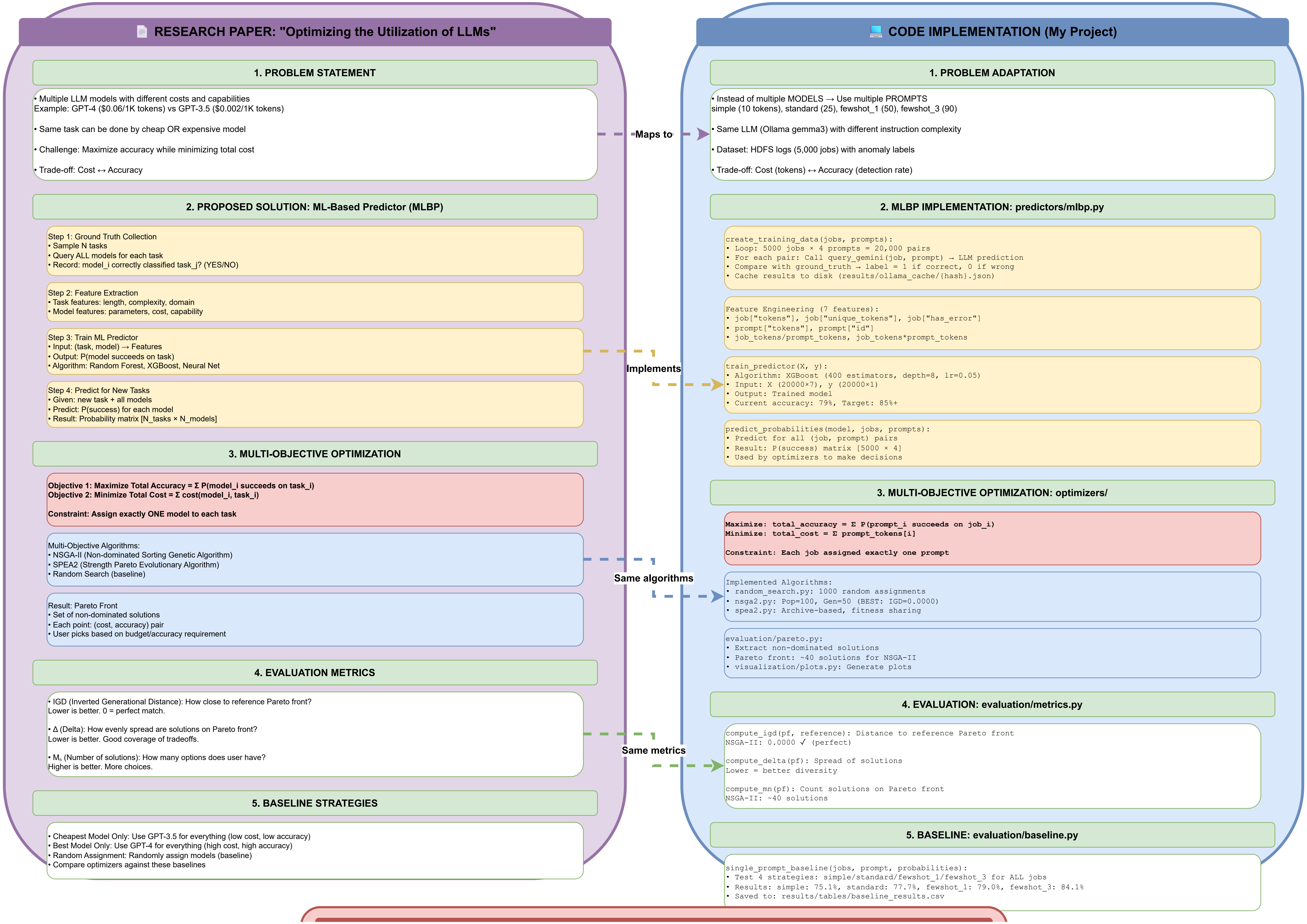
Research Paper Theory vs Code Implementation



📄 RESEARCH PAPER: "Optimizing the Utilization of LLMs"

1. PROBLEM STATEMENT

- Multiple LLM models with different costs and capabilities
Example: GPT-4 ($0.06/1K tokens) vs GPT-3.5 ($0.002/1K tokens)
- Same task can be done by cheap OR expensive model
- Challenge: Maximize accuracy while minimizing total cost
- Trade-off: Cost ↔ Accuracy

2. PROPOSED SOLUTION: ML-Based Predictor (MLBP)

Step 1: Ground Truth Collection
- Sample N tasks
- Query ALL models for each task
- Record: model_i correctly classified task_j? (YES/NO)

Step 2: Feature Extraction
- Task features: length, complexity, domain
- Model features: parameters, cost, capability

Step 3: Train ML Predictor
- Input: (task, model) → Features
- Output: P(model succeeds on task)
- Algorithm: Random Forest, XGBoost, Neural Net

Step 4: Predict for New Tasks
- Given: new task + all models
- Predict: P(success) for each model
- Result: Probability matrix [N_tasks × N_models]

3. MULTI-OBJECTIVE OPTIMIZATION

Objective 1: Maximize Total Accuracy = Σ P(model_i succeeds on task_i)
Objective 2: Minimize Total Cost = Σ cost(model_i, task_i)

Constraint: Assign exactly ONE model to each task

Multi-Objective Algorithms:
- NSGA-II (Non-dominated Sorting Genetic Algorithm)
- SPEA2 (Strength Pareto Evolutionary Algorithm)
- Random Search (baseline)

Result: Pareto Front
- Set of non-dominated solutions
- Each point: (cost, accuracy) pair
- User picks based on budget/accuracy requirement

4. EVALUATION METRICS

- IGD (Inverted Generational Distance): How close to reference Pareto front?
Lower is better. 0 = perfect match.
- Δ (Delta): How evenly spread are solutions on Pareto front?
Lower is better. Good coverage of tradeoffs.
- M_s (Number of solutions): How many options does user have?
Higher is better. More choices.

5. BASELINE STRATEGIES

- Cheapest Model Only: Use GPT-3.5 for everything (low cost, low accuracy)
- Best Model Only: Use GPT-4 for everything (high cost, high accuracy)
- Random Assignment: Randomly assign models (baseline)
- Compare optimizers against these baselines

💻 CODE IMPLEMENTATION (My Project)

1. PROBLEM ADAPTATION

- Instead of multiple MODELS → Use multiple PROMPTS
simple (10 tokens), standard (25), fewshot_1 (50), fewshot_3 (90)
- Same LLM (Ollama gemma3) with different instruction complexity
- Dataset: HDFS logs (5,000 jobs) with anomaly labels
- Trade-off: Cost (tokens) ↔ Accuracy (detection rate)

2. MLBP IMPLEMENTATION: predictors/mlbp.py

```
create_training_data(jobs, prompts):
• Loop: 5000 jobs × 4 prompts = 20,000 pairs
• For each pair: Call query_gemini(job, prompt) → LLM prediction
• Compare with ground_truth → label = 1 if correct, 0 if wrong
• Cache results to disk (results/ollama_cache/{hash}.json)
```

```
Feature Engineering (7 features):
• job["tokens"], job["unique_tokens"], job["has_error"]
• prompt["tokens"], prompt["id"]
• job_tokens/prompt_tokens, job_tokens*prompt_tokens
```

```
train_predictor(X, y):
• Algorithm: XGBoost (400 estimators, depth=8, lr=0.05)
• Input: X (20000×7), y (20000×1)
• Output: Trained model
• Current accuracy: 79%, Target: 85%+
```

```
predict_probabilities(model, jobs, prompts):
• Predict for all (job, prompt) pairs
• Result: P(success) matrix [5000 × 4]
• Used by optimizers to make decisions
```

3. MULTI-OBJECTIVE OPTIMIZATION: optimizers/

```
Maximize: total_accuracy = Σ P(prompt_i succeeds on job_i)
Minimize: total_cost = Σ prompt_tokens[i]

Constraint: Each job assigned exactly one prompt
```

```
Implemented Algorithms:
• random_search.py: 1000 random assignments
• nsga2.py: Pop=100, Gen=50 (BEST: IGD=0.0000)
• spea2.py: Archive-based, fitness sharing
```

```
evaluation/pareto.py:
• Extract non-dominated solutions
• Pareto front: ~40 solutions for NSGA-II
• visualization/plots.py: Generate plots
```

4. EVALUATION: evaluation/metrics.py

```
compute_igd(pf, reference): Distance to reference Pareto front
NSGA-II: 0.0000 ✓ (perfect)

compute_delta(pf): Spread of solutions
Lower = better diversity

compute_mn(pf): Count solutions on Pareto front
NSGA-II: ~40 solutions
```

5. BASELINE: evaluation/baseline.py

```
single_prompt_baseline(jobs, prompt, probabilities):
• Test 4 strategies: simple/standard/fewshot_1/fewshot_3 for ALL jobs
• Results: simple: 75.1%, standard: 77.7%, fewshot_1: 79.0%, fewshot_3: 84.1%
• Saved to: results/tables/baseline_results.csv
```

🔍 KEY DIFFERENCES: Paper vs Implementation

Aspect	Paper Theory	Your Implementation
Models/Prompts	Multiple LLMs	Single LLM, 4 prompts
Dataset	General NLP tasks	HDFS anomaly detection
LLM API	OpenAI GPT-3.5/4	Local Ollama (gemma3)
Cost measure	$ per 1K tokens	Token count (proxy)