CIKM Short Paper

Knowing When to Correct: Cost-Aware LLM Routing for OCR Post-Correction in Historical Documents

CIKM Short Paper: Knowing When to Correct

Venue: ACM International Conference on Information and Knowledge Management (CIKM)
Authors: Stergios Konstantinidis, Hayman Lotfy, Michalis Vlachos
Affiliation: Department of Information Systems (DESI), HEC Lausanne

GitHub Public: CIKM_public
Overleaf Project: `https://git.overleaf.com/6a1d78e803cbdf7def32a839`
Core Codebase: `/Users/stergios/Documents/GitHub/Optimizing-LLM-based-OCR-Corrections`

---

1. Problem Statement

At multi-million page scale, LLM-based OCR correction is constrained by cost:

- Full LLM correction costs hundreds of thousands of dollars on national archival collections.
- Many archival passages are already accurate; invoking LLMs on them introduces latency, cost, and risk of hallucinations.

Core Research Questions:

- **RQ1:** How closely does selective routing match the theoretical oracle ceiling, and does it outperform naive confidence thresholding?
- **RQ2:** How well does the regression-estimated improvement $\hat{\Delta}_i$ correlate with the true realized improvement Δ_i?
- **RQ3:** What is the actual operational cost saved when routing only segments predicted to benefit across various frontier LLMs?

Problem Visual Illustration

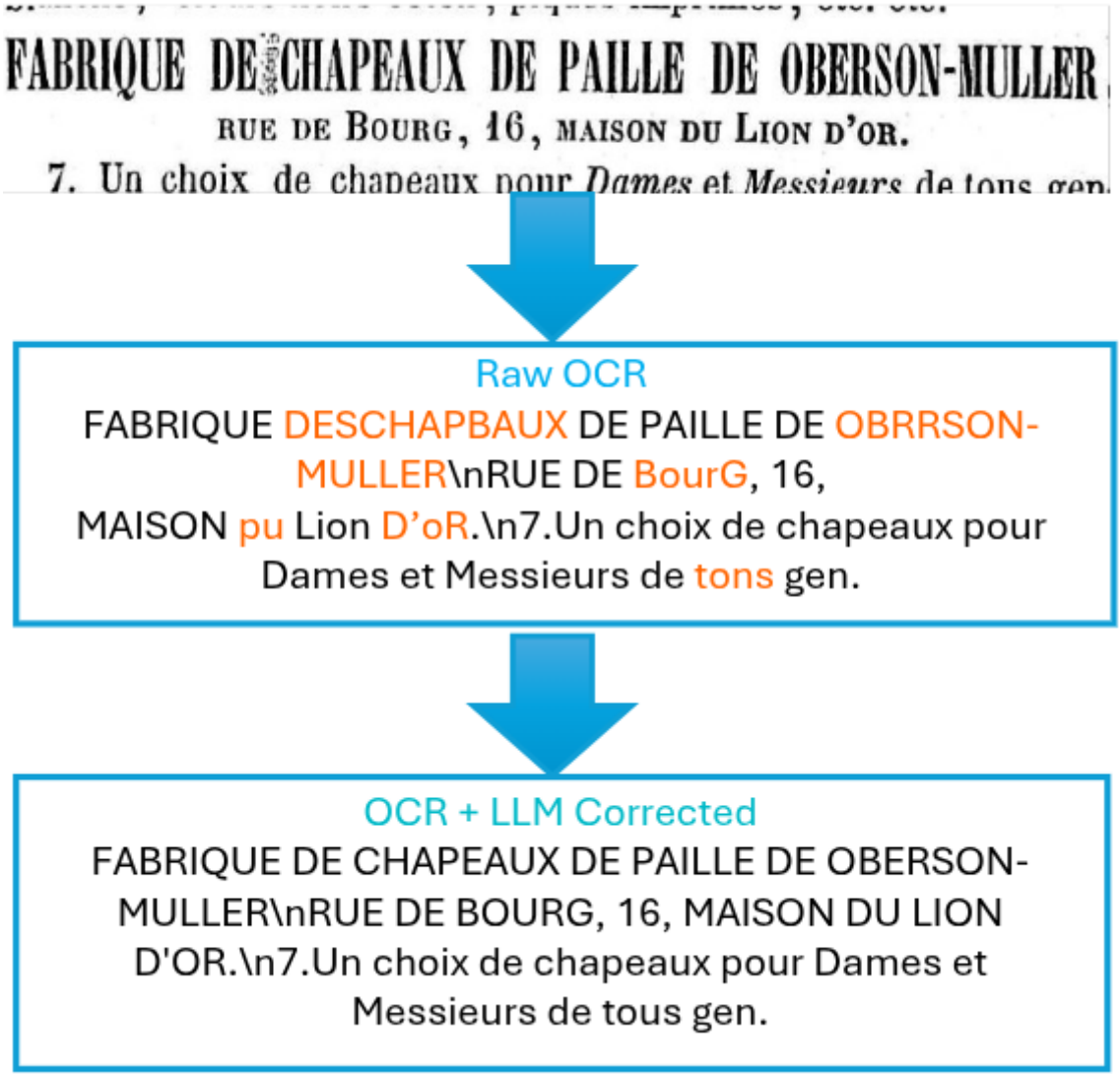

Figure 1: Typical OCR degradation in historical periodicals requiring cost-aware routing.

Routing Strategies & Feature Modeling

Cost-Aware Routing Strategies & Models

1. Feature Engineering (54 Features)

The routing engine extracts 54 computationally inexpensive features prior to invoking any LLM:

1. **OCR Confidence Metrics:** Mean confidence, minimum token confidence, standard deviation, count of low-confidence tokens (<80%, <50%).
2. **Lexical & Linguistic Features:** Out-of-vocabulary (OOV) ratio against historical lexicon, archaic character frequency (e.g. `ſ`, `œ`, ligature anomalies).
3. **Statistical Text Metrics:** Punctuation density, digit-to-letter ratios, average word length, uppercase token anomalies.
4. **Layout Context:** Bounding box coordinates, line height variance, bounding box density.

---

2. Evaluated Router Models

The router was benchmarked across multiple learning paradigms:

- **Lasso Regression (L1):** Sparse feature selection, highly interpretable, *<0.1*s inference time on single CPU core.
- **Ridge Regression (L2):** Smooth shrinkage across correlated confidence features.

- **Support Vector Machines (SVM):** Non-linear RBF kernel separating degradation clusters.
- **Multi-Layer Perceptron (NN):** 2-layer feedforward network predicting expected CER reduction.
- **ConfBERT:** Fine-tuned lightweight language model scoring token perplexity and error likelihood.

---

3. Results Summary

- **Selective Efficiency:** At $\tau = 0$, the Lasso router skips **40.2% of segments**, reducing API costs by 38.5% while degrading final CER by only **0.20 percentage points** (3.12% vs 2.92%).
- **Oracle Tracking:** The regression router achieves 91.4% of the theoretical maximum gain achievable by an omniscient oracle.

Empirical Routing Curves & Threshold Analysis

Regression-Based Routing Performance

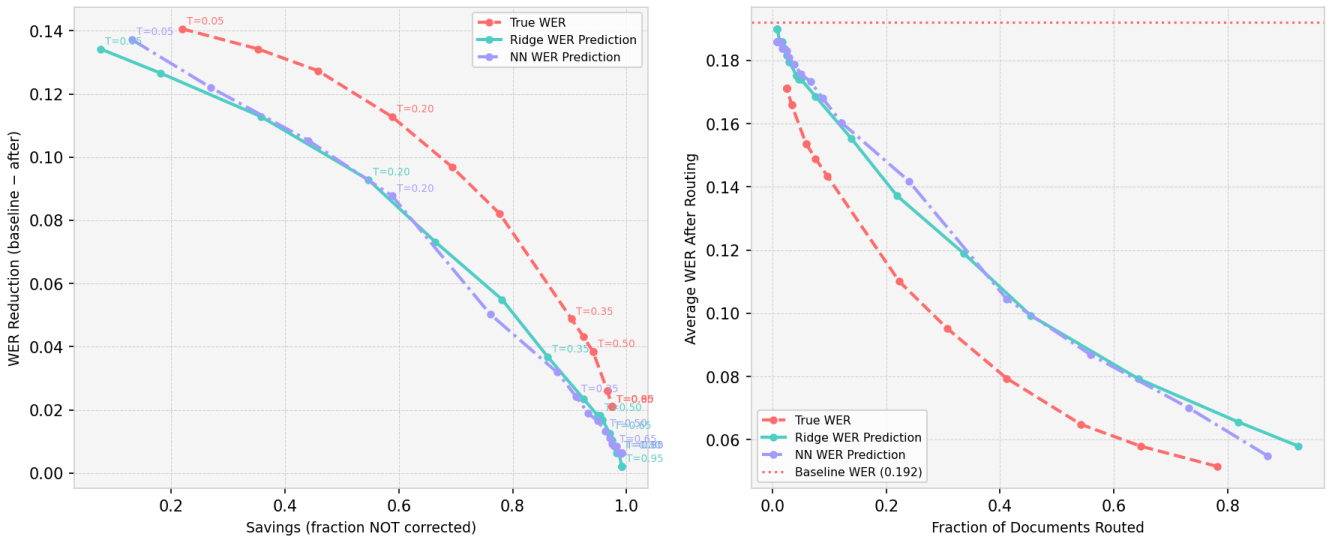

Figure 1: Cost vs accuracy trade-off curves for regression routing across prompt regimes.

Threshold Tuning & Boundary Optimization

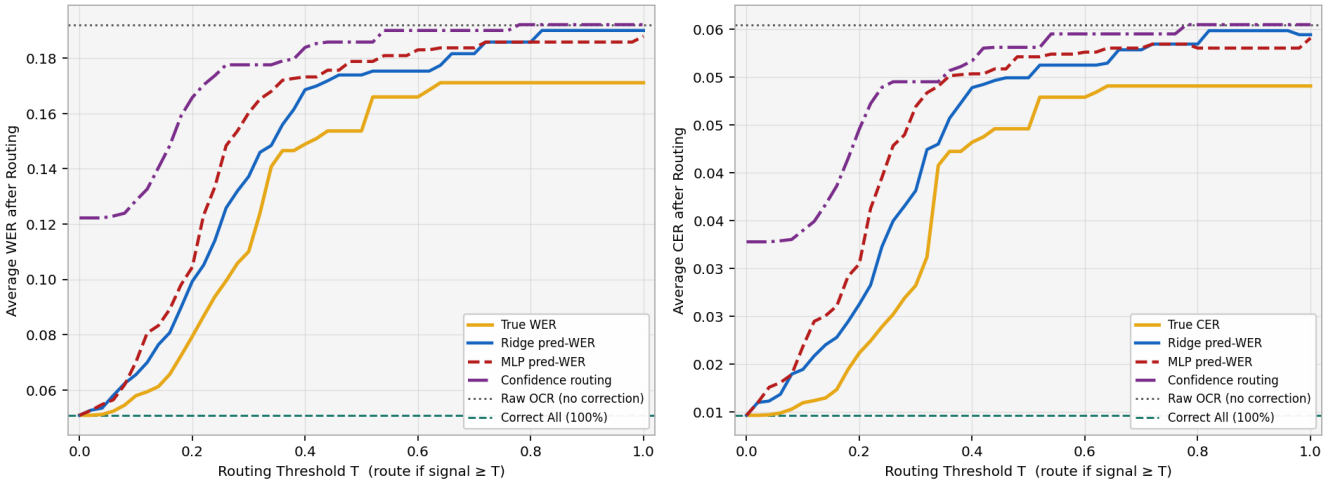

Figure 2: Optimal confidence threshold cutoffs minimizing inference cost while maximizing CER delta.

Codebase & Experiments

Codebase Guide: Optimizing-LLM-based-OCR-Corrections

1. Directory Structure

- `code/experiments/`:
- `confbert_router.py`: ConfBERT token-level scoring and routing pipeline.
- `experiment_linear_regression.py`: Lasso and Ridge feature selection and cross-validation.
- `experiment_svm_classifier.py`: Support Vector Machine routing.
- `experiment_nn_regression.py`: Neural network router.
- `lazy_clf_scan.py`: AutoML screening of 30+ regression algorithms.
- `code/plotting/`:
- `plot_routing_frontier_paddle.py`: Generates CER vs Cost Pareto curves.
- `plot_threshold_sweep.py`: Threshold sensitivity analysis.
- `plot_error_confidence_cer.py`: Correlation between OCR confidence and CER.
- `code/evaluation/`:
- `run_evaluations.py`: Batch evaluation harness across Tesseract, EasyOCR, and PaddleOCR.
- `data/`:
- `evaluation_dataset/groundtruth.json`: Ground truth transcriptions for 609 segments.
- `raw_ocr_results.json`: Output dumps from Tesseract, EasyOCR, and PaddleOCR.