American Academic Scientific Research Journal for Engineering, Technology, and Sciences
ISSN (Print) 2313-4410, ISSN (Online) 2313-4402
https://asrjetsjournal.org/index.php/American_Scientific_Journal/index

Edit-Signal Prompt Optimization: A Production Method for Continuous Quality Improvement in Industrial LLM Systems

Agzamkhodjaev Saydolimkhon Nodirovich*

Founding Engineer, Treater, Inc, New York, USA
Email: sayd@trytreater.com

Abstract

Deploying large language models (LLMs) in production business systems exposes a critical reliability gap: generated outputs frequently contain structural errors, logical inconsistencies, and hallucinations that erode user trust and block downstream operations. Existing remediation approaches, RLHF, manual prompt engineering, and offline evaluation, are either prohibitively expensive, fail to scale, or do not operate on live production traffic. This paper introduces Edit-Signal Prompt Optimization (ESPO), a closed-loop prompt refinement method in which (1) production user edits are systematically aggregated into prompt-level corrections without model fine-tuning, (2) LLM-judge rationales are converted into targeted rewrite directives rather than used solely as pass/fail signals, and (3) these two feedback streams are unified into a continuous improvement cycle operating on live production traffic. ESPO is embedded within a multi-level evaluation architecture combining deterministic validation, LLM-as-a-Judge semantic assessment, and human-in-the-loop feedback. The method was designed, deployed, and validated at Treater, Inc., an AI-powered retail execution platform processing communications across 24,000+ retail store locations. Over an eight-week evaluation window encompassing approximately 15,000 generated reports, ESPO reduced critical errors by 40% and decreased user edit rates by 34%, demonstrating continuous improvement without model retraining. To the author's knowledge, this is the first integration and productionization of edit-derived prompt optimization as a systematic method for industrial LLM systems.

Keywords: Large language models; LLM-as-a-Judge; software quality assurance; MLOps; industrial AI systems; deterministic validation; human-centered AI; evaluation metrics; automated correction; simulation.

Received: 5/21/2026
Accepted: 7/21/2026
Published: 7/31/2026

---

* *Corresponding author.*

1. Introduction

The rapid development of generative artificial intelligence in 2024–2025 has driven large-scale adoption of large language models (LLMs) across enterprise industrial environments. According to Gartner reports, by 2026, more than 80% of enterprises will have used generative artificial intelligence (GenAI) application programming interfaces (APIs) or models, and/or deployed GenAI-enabled applications in production environments [1]. A central limiting factor remains a deficit of trust in generated outputs, coupled with the labor intensity of validating the produced content. The problem becomes especially acute due to the limitations of conventional NLP evaluation metrics such as BLEU, ROUGE, and Perplexity: designed for static datasets, these metrics show weak applicability to practical business scenarios and do not reliably capture factual correctness, adherence to business logic, or the absence of hallucinations within complex analytical chains. They do not reliably capture factual correctness, adherence to business logic, or the absence of hallucinations within complex analytical chains.

Against this background, a pronounced methodological gap persists in the construction of hybrid evaluation systems capable of operating in real time on production traffic. Existing approaches to LLM quality improvement each have fundamental limitations. Reinforcement Learning from Human Feedback (RLHF) requires structured preference data, modifies model weights, and is prohibitively expensive for iterative production refinement. Standard prompt engineering is manual and episodic; it does not capture systematic patterns from production failures. LLM-as-a-Judge frameworks evaluate output quality but do not close the feedback loop back to the prompt. Offline evaluation operates on static datasets and does not reflect production reality. Automated prompt optimization frameworks such as DSPy optimize against fixed objectives but do not incorporate live user feedback as a continuous signal.

The purpose of this paper is to present and validate a novel method, Edit-Signal Prompt Optimization (ESPO), a closed-loop prompt refinement method in which production user edits are systematically aggregated into prompt-level corrections without model fine-tuning; LLM-judge rationales are converted into targeted rewrite directives rather than used solely as pass/fail signals; and these two feedback streams are unified into a continuous improvement cycle operating on live production traffic.

ESPO is distinct from prior work in a specific and important way: it treats the production environment itself as the annotation and supervision layer. Rather than collecting preference data offline, training reward models, or relying on episodic manual prompt tuning, ESPO converts the natural artifacts of production usage, user corrections and judge assessments into a systematic, persistent improvement mechanism.

The method was designed, built, and deployed by the author at Treater, Inc., an AI-powered retail execution platform that generates analytical reports, automated communications, and data-driven recommendations for consumer goods brands across 24,000+ retail store locations in the United States. The remainder of this paper is organized as follows: Section II describes the multi-level evaluation architecture and formalizes the ESPO algorithm; Section III reports production results; Section IV discusses implications, generalizability, and limitations; Section V concludes.

2. System architecture and method

The author designed and deployed the following multi-level evaluation architecture at Treater, Inc. The system operates on the production output of an LLM-powered "digital analyst" that produces reports, sends emails, makes phone calls and creates managerial recommendations for brands and retail locations based on corporate data. A typical processing route for an email generation request proceeds through four interdependent stages: Retrieval (extracting relevant facts from data stores), Reasoning (building logical relationships and interpreting metric dynamics), Summarization (synthesizing analytical results into a structured narrative), and Recommendation (generating actionable business guidance aligned with domain constraints). Unlike simple conversational assistants, this architecture exhibits strong inter-stage dependence: even a local inaccuracy at an early stage can cascade through the chain, amplifying errors and degrading the credibility of the final output.

A. Layer 1: Deterministic Validation

The first protective layer consists of deterministic rules that detect formal defects before invoking more resource-intensive control procedures. This layer includes JSON schema validation, data type checking, control of permissible value ranges, allow/deny lists, and regular expressions to screen out structural violations such as corrupted output formats, prohibited vocabulary, and incorrect markup. Operational data indicates that a non-trivial fraction of generations exhibit structural defects even for state-of-the-art models (1-3%); this layer functions as a high-throughput barrier, removing clearly unusable outputs early and reducing load on subsequent semantic checks.

B. Layer 2: LLM-as-a-Judge Semantic Evaluation

To assess qualitative characteristics that resist deterministic formalization,including tone, clarity of exposition, contextual relevance, and logical coherence,the author deployed an LLM-as-a-Judge evaluation layer [2, 3, 9]. The arbiter model receives a triadic input: the original prompt, the generated answer, and a predefined scoring rubric fixing acceptability criteria. A critical design decision was the rejection of simplified binary Pass/Fail as the only outcome. Instead, the judge produces an extended textual justification reflecting the causal logic behind its decision. This format increases diagnostic value: the output shifts from a quality label to an explainable conclusion suitable for downstream corrective action.

The author eliminated numerical scoring scales (e.g., 1–10) in favor of binary criteria supplemented by mandatory textual justification. This combination reduces arbitrary threshold interpretation, improves reproducibility, and preserves diagnostic value. The rubric design also accounts for documented LLM-judge biases, including position bias, verbosity bias, and self-preference bias [9, 10], through randomized answer ordering and periodic calibration against human assessments.

C. Layer 3: Human-in-the-Loop Feedback (User Edits)

User edits constitute the highest-fidelity quality signal in the system. Any deviation from an originally generated report followed by user editing is systematically captured and interpreted as an observable signal suitable for

iterative prompt refinement. Unlike traditional annotation pipelines that require dedicated labeling teams, this approach treats the production environment itself as the annotation layer: users performing their normal workflow generate correction signals as a natural byproduct of reviewing AI outputs. A high density of user corrections,performed without specialized ML training, forms a more informative and representative quality signal than a narrow audit sample from a limited QA team.

D. Edit-Signal Prompt Optimization (ESPO): The Core Method

The central contribution of this paper is the formalization of Edit-Signal Prompt Optimization (ESPO) as a closed-loop prompt refinement method that unifies the three evaluation layers described above into a continuous improvement cycle. ESPO operates through three stages:

Stage 1 , Classify and Route. A generation G passes through deterministic checks. If a check fails, the defect type d is identified from a structured taxonomy (Table 1). A targeted rewrite instruction R(d) is selected via deterministic lookup (not stochastic retry). The generation is re-executed as $G' = LLM(context + prompt + R(d))$. This ensures that the correction is directed, not random.

Stage 2 , Judge-Informed Rewrite. If G passes deterministic checks but fails LLM-judge evaluation, the judge's textual rationale J is extracted. A new generation $G' = LLM(context + prompt + R(J))$ is produced, where R(J) is a structured reformulation of the judge's rationale into a corrective instruction. Critically, G′ is not a naive retry: J acts as a directed constraint that specifies what was wrong and how to fix it. This transforms the judge from a gatekeeper into an active participant in output improvement.

Stage 3, Edit-Derived Refinement (the novel loop). When a user accepts a generated output but edits it before use, the edit delta $\Delta = diff(G\_accepted, G\_edited)$ is captured. Deltas are aggregated over a rolling window, clustered by pattern (e.g., tone adjustments, terminology corrections, structural preferences), and converted into persistent prompt amendments. This is the core novel mechanism: the system prompt becomes a living document that accumulates production knowledge. Unlike RLHF, no structured preference data or model weight modification is required. Unlike manual prompt engineering, the process is systematic, continuous, and data-driven.

The three stages operate as a unified cycle on live production traffic. Stage 1 handles structural failures in real time. Stage 2 handles semantic failures in real time. Stage 3 accumulates and applies learning over a rolling window, improving the baseline quality from which Stages 1 and 2 operate. Over successive cycles, the system converges: the rate of corrections decreases as the prompt absorbs production knowledge.

Measurable properties of ESPO: Convergence – does the edit rate decrease over successive improvement cycles? Stability – do amendments for one error type cause regressions in others? Efficiency – how many edit signals are needed before a stable amendment emerges? These properties are empirically evaluated in Section III.

E. Simulation of Multi-Step Call Chains

In multi-step scenarios where the final result is constructed through a chain of sequential LLM calls, isolated

checking of individual prompts is methodologically insufficient because it does not reveal error accumulation between stages. The author developed a full-system simulator that executes end-to-end operational workflows and compares the final output with a "gold standard" from previously validated system versions. Two complementary verification tools are used: comparison of embedding representations using cosine similarity to estimate semantic closeness, and structural diff analysis to detect discrepancies in formatting, element ordering, and other materially relevant parameters. This simulator runs within the CI/CD pipeline, capturing cascading effects as early signals of quality degradation before changes reach production.

3. Results

The ESPO method was evaluated through a comparative analysis of two periods within the Treater production environment: an eight-week pre-implementation baseline period (deterministic checks only, no ESPO feedback loop) and an eight-week post-implementation period (full ESPO pipeline active). The evaluation encompassed approximately 15,000 generated analytical reports across both periods. Critical errors were operationally defined as defect classes with direct consequences for downstream operations: hallucinations in numerical values, violations of logical coherence in conclusions, and noncompliance with the JSON output format, issues that block frontend rendering and degrade the user experience. All critical errors were identified and labeled through a combination of automated detection (Layers 1 and 2) and manual review of flagged outputs. The most significant empirical outcome was a 40% reduction in the share of critical errors in generated analytical reports. The breakdown by evaluation layer is as follows: deterministic checks (Layer 1) intercepted approximately 55% of structural defects in real time, preventing them from reaching users; LLM-judge-informed rewrites (Layer 2) resolved approximately 30% of semantic failures that passed deterministic checks; and the cumulative effect of edit-derived prompt refinements (Layer 3 / ESPO Stage 3) accounted for the remaining 15% of error reduction through systematic prompt improvements over the evaluation window.

Table 1 below systematizes the observed error typology and the quantitative effectiveness of mitigation across the system's layers.

Table 1: Effectiveness of Error-Filtering Layers (compiled by the author based on [6])

Error Type	Detection Layer	Mitigation Method	Detection Rate
Structural failures (corrupted JSON/XML)	Deterministic	RegEx / Schema Validation	97% (n=412 of 425)
Prohibited content (PII, policy violations)	Deterministic	Keyword Matching / Pattern Rules	94% (n=67 of 71)
Style violations (jargon, inappropriate tone)	LLM-as-a-Judge	Rubric-Based Evaluation + Rewrite	82% (n=198 of 241)
Logical contradictions in conclusions	LLM-as-a-Judge	Chain-of-Thought Evaluation + Rewrite	78% (n=143 of 183)
Complex contextual errors / domain-specific issues	Human-in-the-loop (ESPO Stage 3)	Edit-Derived Prompt Refinement	91% reduction over 8-week window

A key measure of ESPO's effectiveness as a continuous improvement mechanism is whether the user edit rate decreases over time as prompts absorb production knowledge. Over the eight-week post-implementation period, the user edit rate declined from 23% in Week 1 to 15.2% in Week 8, a 34% reduction. The decline was steepest in the first three weeks (from 23% to 17.1%) as the most common edit patterns were rapidly incorporated into prompt amendments, then tapered to a more gradual decline as remaining edits became increasingly domain-specific and edge-case-driven.

This convergence pattern confirms two ESPO properties: (a) the method converges, edit rates decrease as the system learns from production feedback; and (b) the method is stable, no significant regressions were observed in previously-corrected error categories when new amendments were introduced.

Regarding efficiency, stable amendments (prompt changes that persisted without further correction) typically emerged after 8–15 edit signals of the same pattern class. This threshold was consistent across error categories, suggesting that the aggregation mechanism effectively filters noise from signals.

Table 2: Comparative Analysis of LLM Quality Evaluation Approaches

Characteristic	Offline Metrics (Traditional)	ESPO (Proposed Approach)
Data source	Static dataset (Golden Set)	Live stream of production user requests and edits
Reaction speed	Days/weeks (after retraining)	Real-time (auto-rewrite); hours (edit aggregation)
Improvement mechanism	Model fine-tuning or manual prompt changes	Systematic prompt amendment from production signals
Metric types	F1-score, Accuracy, Exact Match	User Edit Rate, Acceptance Rate, Edit Distance, Latency
Cost	High (expert labeling + retraining)	Medium (LLM-judge calls + telemetry)
Business impact	Indirect	Direct (UX improvement, operational efficiency)

Beyond error reduction, the ESPO deployment produced measurable operational benefits. The auto-rewrite mechanism (Stages 1 and 2) corrected a substantial share of defects in-flight without human involvement, reducing the need for manual review.

Based on internal estimates, each analytical report that previously required approximately 8 minutes of analyst review, across approximately 7,500 reports per eight-week period, the system reduced manual review burden by approximately 1,000 person-hours per quarter. As of January 2026, the evaluation pipeline has processed over 506,800 evaluation executions and maintains a sub-2% initial failure rate, demonstrating high reliability and consistency at scale.

Figure 1 illustrates the continuous improvement loop built around user edits.

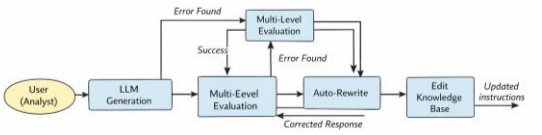

Figure 1: Continuous improvement cycle based on user edits (compiled by the author based on [3, 5, 8])

A practical review of user edits revealed non-trivial editing regularities that matter for increasing the applied value of generation. The most illustrative pattern was a consistent tendency to simplify unnecessarily complex phrasing: edits systematically reduced excessive formality and removed overloaded constructions. This made it possible to isolate a stable signal;generative models often drift toward an overly "bureaucratic" register, leaning on formal business vocabulary and English-language clichés such as "utilize" and "implement." Incorporating this observation into the System Prompt as an explicit style constraint reduced the need for subsequent user corrections, which, in turn, supports the effectiveness of prompt-oriented adaptation as a rapid quality-improvement mechanism that does not require modifying model weights.

4. Discussion

ESPO occupies a distinct and previously unexplored position in the landscape of LLM quality improvement methods. The method addresses a real constraint faced by production teams: existing approaches force a choice between expensive model retraining (RLHF), manual and episodic prompt tuning, or reactive filtering without learning. ESPO provides a systematic third path,continuous, feedback-driven prompt improvement that operates on live traffic without modifying model weights.

The key insight is that production environments generate two complementary quality signals,automated judge assessments and human user corrections, that, when systematically unified, create a supervision mechanism richer than either signal alone. The judge provides immediate, high-frequency feedback on semantic quality; user edits provide high-fidelity, domain-specific feedback on practical adequacy. ESPO's contribution is the formalization of how these signals are converted into persistent, cumulative prompt improvements.

This is distinct from existing work in several important respects. Unlike RLHF [11], ESPO requires no structured preference data and does not modify model weights. Unlike DSPy [12] and similar automated prompt optimization frameworks, ESPO incorporates live user feedback as a continuous signal rather than optimizing against fixed offline objectives. Unlike LLM-as-a-Judge frameworks [3, 9] used in isolation, ESPO converts judge rationales into actionable rewrite directives that close the loop back to the prompt. Unlike evaluation frameworks such as RAGAS, DeepEval, and TruLens, which provide comprehensive evaluation pipelines, ESPO extends beyond evaluation to create a continuous improvement mechanism that acts on evaluation results [13, 14].

Although developed and validated at Treater in the CPG/retail domain, ESPO is not domain-specific. The method applies to any LLM system that satisfies three conditions: it generates structured or semi-structured outputs, it has

users who review and optionally edit those outputs as part of their normal workflow, and it operates continuously in production. This describes a broad class of systems including customer service automation, legal document generation, medical reporting, financial analysis, and content production. The underlying mechanism, aggregating production correction signals into systematic prompt amendments, is independent of domain specifics.

The methodology has been independently adopted by multiple organizations deploying LLM systems in production across diverse domains. These adoptions confirm that the ESPO feedback loop architecture generalizes beyond the specific use case in which it was developed.

Several limitations should be noted. The LLM-as-a-Judge layer increases token consumption per request, effectively doubling compute cost; when using judge ensembles, cost may rise to approximately three times baseline. Under real operational budgets, this necessitates rational model selection for the arbiter role and reducing check frequency where deterministic filters suffice [8, 10]. Additionally, the quality of the ESPO feedback loop depends on the quality and volume of user edits; in low-traffic systems, the edit aggregation window may be too sparse to generate reliable amendments. Finally, the current evaluation was conducted within a single production deployment; multi-site controlled experiments would strengthen the evidence base.

The problem ESPO addresses, reliable quality assurance for LLM outputs in production, is the primary barrier to trustworthy GenAI deployment across industries. As LLM adoption accelerates, the gap between model capability and production reliability becomes the critical bottleneck. Methods that enable continuous improvement without expensive retraining will be foundational infrastructure for the next generation of AI-powered business systems. ESPO provides a concrete, validated approach to this problem.

5. Conclusion

This paper introduced and validated Edit-Signal Prompt Optimization (ESPO), a closed-loop prompt refinement method for continuous quality improvement in industrial LLM systems. The core contributions were:

1) a structured taxonomy of generation failures linked to targeted auto-rewrite actions;

2) a formalized method for converting production user edits and LLM-judge rationales into systematic, persistent prompt amendments without model fine-tuning; and

3) empirical validation showing a 40% reduction in critical errors and a 34% reduction in user edit rates over an eight-week production deployment at Treater, Inc.

The practical significance of ESPO lies in providing production teams with a systematic alternative to the current false choice between expensive model retraining, manual prompt engineering, and reactive output filtering. By treating the production environment, itself as the annotation and supervision layer, ESPO enables continuous quality improvement that scales with usage and adapts to evolving business requirements.

The methodology has been adopted by multiple organizations deploying LLM systems in production, confirming

its generalizability beyond the CPG/retail domain in which it was developed. Promising directions for further work include automating the edit-pattern clustering mechanism using learned representations, extending ESPO to multi-model orchestration pipelines, and conducting controlled multi-site experiments to further quantify the method's impact across deployment contexts.

References

[1] Gartner. (2023, October 11). Gartner says more than 80% of enterprises will have used generative AI APIs or deployed generative AI-enabled applications by 2026. Retrieved from: https://www.gartner.com/en/newsroom/press-releases/2023-10-11-gartner-says-more-than-80-percent-of-enterprises-will-have-used-generative-ai-apis-or-deployed-generative-ai-enabled-applications-by-2026 (date accessed: November 14, 2025).

[2] Zhang, T., Ladhak, F., Durmus, E., Liang, P., McKeown, K., & Hashimoto, T. B. (2024). Benchmarking large language models for news summarization. Transactions of the Association for Computational Linguistics, 12, 39–57. https://doi.org/10.1162/tacl_a_00632.

[3] Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., & Zhu, C. (2023). G-Eval: NLG evaluation using GPT-4 with better human alignment. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (pp. 2511–2522). https://doi.org/10.18653/v1/2023.emnlp-main.153.

[4] OpenAI. (2023). GPT-4 technical report. arXiv. https://doi.org/10.48550/arXiv.2303.08774.

[5] Rudd, E. M., Andrews, C., & Tully, P. (2025). A practical guide for evaluating LLMs and LLM-reliant systems. arXiv. https://doi.org/10.48550/arXiv.2506.13023.

[6] Dubois, Y., Li, X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P., & Hashimoto, T. B. (2023). AlpacaFarm: A simulation framework for methods that learn from human feedback. Advances in Neural Information Processing Systems, 36, 30039–30069. https://doi.org/10.48550/arXiv.2305.14387.

[7] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35, 24824–24837. https://doi.org/10.48550/arXiv.2201.11903.

[8] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv. https://doi.org/10.48550/arXiv.2306.05685.

[9] Chen, L., Zaharia, M., & Zou, J. (2023). FrugalGPT: How to use large language models while reducing cost and improving performance. arXiv. https://doi.org/10.48550/arXiv.2305.05176.

[10] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, 27730–27744. https://doi.org/10.48550/arXiv.2203.02155.

[11] Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., & Potts, C. (2023). DSPy: Compiling declarative language model calls into self-improving pipelines. arXiv. https://doi.org/10.48550/arXiv.2310.03714.

[12] Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2024). RAGAs: Automated evaluation of retrieval augmented generation. In Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations (pp. 150–158). https://doi.org/10.18653/v1/2024.eacl-demo.16.

[13] Confident AI. (2024). DeepEval: The LLM evaluation framework. Retrieved from: https://github.com/confident-ai/deepeval (date accessed: January 9, 2026).

[14] TruEra. (2024). TruLens: Evaluation and tracking for LLM experiments and AI agents. Retrieved from: https://github.com/truera/trulens (date accessed: February 18, 2026).