Prompt Rewriter

A Cross-Model Intent Resolution Layer for Consumer LLM Interfaces

Public White Paper | Federated Unified Worker Network Inc. | May 2026

Editorial Note: This public white paper is adapted from an internal technical report. Proprietary implementation details, internal tooling references, quota thresholds under active revision, and specific codebase paths have been removed or generalized. All planning assumptions are clearly labeled **[ASSUMPTION]**. Repository-derived facts are labeled **[Implementation Fact]**. Peer-reviewed citations are provided in full in the References section.

Abstract

Consumer large language model (LLM) interfaces expose powerful generative models behind structurally impoverished input surfaces: a single blank text box. Users routinely submit underspecified objectives, omit audience and constraint information, and conflate desired output format with task description. The result is a systematic quality bottleneck in which the same underlying model may produce mediocre or expert-level output depending solely on prompt structure.

This paper presents **Prompt Rewriter**, a browser-native intent resolution layer that interposes between a user's draft prompt and the LLM interface. The system detects the user's domain and task, applies a curated skill and technique catalog, enforces user-selected constraints, and emits two candidate rewrites — one optimized for expert framing, one for execution format — before the user sends the request. The architecture operates as a human-in-the-loop prompt compiler: non-destructive, cross-site, and model-agnostic.

Keywords: prompt engineering, large language models, browser extension, intent resolution, human-in-the-loop, prompt compilation, HCI

1. Introduction

Large language models have achieved broad consumer adoption through interfaces that prioritize accessibility: a conversational text box requiring no structured input. Yet this design creates a systematic quality gap. Research has shown that non-expert users — even those with programming backgrounds — consistently produce structurally deficient prompts. Common failure modes include omitting audience specification, conflating exemplification with instruction, and over-relying on vague directives [1]. The same research found that users rarely iterate on prompts after an initial failure, preferring to abandon the task entirely.

This gap between model capability and realized output quality is well-established. Prompts function not as decorative prose but as structured transformations of the user's intent [2]. Techniques such as chain-of-thought prompting — decomposing a request into reasoning steps — materially improve model performance on multi-step tasks [3]. Generating multiple candidate reasoning paths and selecting the most coherent further improves accuracy [4].

Despite this body of work, no general-purpose consumer tool intervenes at the prompt-formulation stage before the request is submitted. Existing approaches either require expert knowledge (manual prompt engineering), are tightly coupled to a single platform, or operate after the fact on generated output rather than on the input itself.

Prompt Rewriter addresses this gap by placing a lightweight compilation layer between the user's draft and the LLM interface. Its core design commitments are:

- **Non-destructiveness:** the user's original draft is never altered without explicit action.
- **Transparency:** two materially different rewrite candidates are shown, enabling informed comparison and choice.
- **Cross-model portability:** the system operates as a browser extension injected across major AI chat platforms, independent of model provider.
- **Feedback closure:** ratings are captured at the rewrite level, enabling distinction between prompt-compilation failures and downstream model failures — a failure class separation absent from most evaluation frameworks.

2. Research Objectives

The project pursues four interlocking research and product objectives:

- **Close the prompt quality gap.** Enable non-expert users to produce prompts that elicit expert-level model outputs, without requiring them to learn prompt engineering.
- **Provide a human-in-the-loop compilation model.** Treat prompt formulation as a structured compilation problem — not a black-box text transformation — and keep the user in control of every transformation decision.
- **Enable cross-model portability.** Produce rewrite artifacts that are model-agnostic in their current form, with a longer-term goal of compiling model-specific prompt surfaces from a neutral intent representation.
- **Create a structured failure taxonomy.** Design a feedback architecture that attributes quality failures to specific pipeline stages (routing, compilation, source mode, divergence, or downstream model) rather than conflating all failure signals at the output level.

3. Background and Related Work

3.1 Prompt-Based Learning

Liu et al. (2021) provide a comprehensive survey of prompting methods, characterizing prompts as a formal transformation of an input into a structured task representation that elicits latent model knowledge [2]. This framing underpins Prompt Rewriter's design: the system is not a text beautifier but a task-representation compiler. The distinction matters for both user expectations and evaluation design.

3.2 Reasoning Elicitation Through Prompt Structure

Wei et al. (2022) demonstrated that chain-of-thought prompting substantially improves model performance on arithmetic, symbolic, and commonsense reasoning tasks [3]. Wang et al. (2022) extended this with

self-consistency decoding, showing that sampling multiple reasoning paths and selecting the most frequent final answer improves accuracy over greedy decoding [4]. Both findings support the design hypothesis that generating multiple candidate prompt structures produces better downstream outcomes than a single 'best' rewrite.

3.3 Automatic Prompt Optimization

Zhou et al. (2022) showed that LLMs can serve as prompt engineers, generating and evaluating candidate instructions via an automated loop that outperforms human-crafted prompts on several benchmarks [6]. Khattab et al. (2023) formalized this in DSPy, framing prompts as compiled artifacts in a metric-driven optimization pipeline [7]. Prompt Rewriter occupies a complementary niche: rather than fully automated optimization over a static dataset, it performs single-shot compilation in a live, human-in-the-loop interaction where user preference feedback is the optimization signal.

3.4 Non-Expert Prompt Authoring

Zamfirescu-Pereira et al. (2023) conducted a controlled study of non-AI-expert prompt authoring and identified four recurring failure modes: specification undercompleteness, exemplar confusion, format ambiguity, and premature task abandonment [1]. The Prompt Rewriter skill and technique catalogs are designed to address the first three directly: skill routing provides role and domain specification; depth controls manage completeness; format selection resolves output format ambiguity.

4. Methodology: System Architecture

4.1 Component Overview

Prompt Rewriter consists of two independently deployable components: a browser extension and a cloud API backend. The separation is architecturally intentional: browser packages can be inspected and modified by users, so all authority for rewrite generation, quota enforcement, and feedback persistence resides server-side. Six primary components handle the end-to-end workflow:

Component	Role
Browser Extension	Injects rewrite controls into AI chat pages; manages local preferences and anonymous client ID.
Rewrite Route (API)	Validates requests, enforces usage quotas, orchestrates the router and compiler, returns A/B rewrites.
LLM Router	Infers language, task domain, primary intent, applicable skill, option labels, and missing context.
Prompt Compiler	Builds system prompts for Option A and B; enforces depth, source mode, and format constraints.
Skill Catalog	Curated expert roles and domain modes for semantic routing. [Implementation Fact]
Technique Catalog	Structured prompt strategies and alternate rewrite methods. [Implementation Fact]

Table 1. Core system components and responsibilities.

4.2 Rewrite Request Lifecycle

The lifecycle formalizes the transformation from raw user input to a stored feedback event. The route planner analyzes the user's prompt and produces a routing plan encoding language, domain, primary intent, selected skill, missing context, and candidate option goals. The compiler then independently generates two rewrites:

Option	Optimization Target	Temperature	Typical Use Case
A: Expert Prompt	Domain expertise, role fidelity, completeness	Lower	Research, analysis, strategy, coding review, high-stakes decisions
B: Execution Prompt	Alternate structure: checklist, table, roadmap, memo, brief, action plan	Higher	Fast execution, scannability, comparison, decision support

Table 2. A/B option semantics. Two separate model calls with different system instructions create genuine divergence.

User feedback is stored as a structured event capturing: the option chosen, rating, reason, original prompt, rewrite, user preferences, platform context, option labels, and timestamp. This schema enables attribution of failures to specific pipeline stages — a deliberate design choice absent from most LLM evaluation frameworks.

4.3 Prompt Compilation Logic

Constraint Preservation. The compiler's core invariant is that the user's topic, language, stated facts, and explicit intent are preserved across all rewrites. Improvements are strictly additive — role specification, task framing, structural decomposition, constraints, exemplar hints, assumptions, evidence requirements, and answer format. Requests are never reclassified into a different task type without the user's explicit indication.

Depth Control. The depth control maps user intent to token budget and structural elaboration: Short generates a concise, direct prompt; Balanced adds role and constraints; Deep adds chain-of-thought scaffolding, assumption enumeration, and verification rules; Clarify+Deep prepends open questions to elicit missing context before compilation. This hierarchy maps to research showing that prompt completeness and task decomposition are independent quality levers [3].

Source Mode. Source mode appends explicit citation, assumption-surfacing, and no-fabrication instructions to the compiled prompt. This is a prompt-level hygiene intervention, not a retrieval-augmented generation (RAG) system. Source mode should not be interpreted as providing verified sourcing; that capability would require a separate retrieval or browsing verification layer not present in the current implementation.

Option B Format Control. Option B format options — Auto, Checklist, Table, Roadmap, Decision Memo, One-Page Brief — correspond to established knowledge management output formats. The Auto setting allows the router to infer the highest-utility format from the detected task domain.

5. Findings and Evaluation Framework

5.1 Failure Taxonomy

A key contribution of this system is a structured failure taxonomy that distinguishes between four distinct failure classes — a distinction absent from most LLM evaluation frameworks, which typically attribute all failure signals to the underlying model:

- **Router failure:** wrong domain or skill selected for the user's intent.
- **Compiler failure:** option divergence too low, or source instructions absent from the compiled prompt.
- **User-model mismatch:** rewrite was structurally sound but did not match the user's downstream model behavior or preferences.
- **Downstream model failure:** a well-formed prompt, but the model output was nonetheless inadequate.

5.2 Proposed Evaluation Metrics

The evaluation framework requires a feedback analytics dashboard attributing quality failures to specific pipeline components. The following metrics are proposed. Note that quality targets marked **[ASSUMPTION]** are planning hypotheses, not measured production values.

Metric	Definition	Target / Status
Rewrite Activation Rate	Modal opens / button impressions	Baseline TBD
Insertion Rate	'Use This' clicks / generated sessions	≥35% [ASSUMPTION]
Option Divergence Score	Semantic + structural difference between A and B	Ensures genuine choice
Positive Feedback Rate	Good ratings / total feedback events	≥70% [ASSUMPTION]
Bad Reason Distribution	Counts by failure class (wrong skill, too long, format, etc.)	Converts dissatisfaction to backlog
Source Compliance Proxy	Share of source-mode rewrites containing cite/verify instructions	Detects compiler failures
Latency (p50/p95)	End-to-end rewrite generation time	Model cost vs. UX tradeoff
Quota Exhaustion Rate	Rate-limited responses / active users	Guides tier calibration

Table 3. Proposed evaluation metrics. Insertion rate and positive feedback targets are planning assumptions, not measured values.

5.3 Depth–Quality Tradeoff Model [ASSUMPTION]

[ASSUMPTION] Based on the theoretical result that marginal quality gains from additional prompt complexity taper as context windows fill [3], a depth–quality tradeoff model is proposed for depth-setting calibration. The model hypothesizes that Short prompts yield limited quality gains; Balanced and Deep settings produce meaningful improvements; and Clarify+Deep offers the highest quality ceiling but at the cost of additional latency and token consumption. Empirical validation against real insertion and feedback data is required before using this model for product decisions.

6. Security, Privacy, and Compliance Considerations

The product's risk profile is moderate in the current implementation and increases when memory, source ingestion, payments, or enterprise deployment are added. The following requirements should be treated as preconditions for wide distribution, not post-launch additions:

1. Add explicit disclosure at the point of feedback capture, informing users that rating a rewrite may transmit their original prompt and the generated rewrite for quality improvement.

2. Implement retention limits and a user-accessible deletion tool before collecting large-scale feedback.

3. Prohibit API key or user secret collection in feedback events and browser extension local storage.

4. Treat source mode as a prompt instruction only — do not market it as verified sourcing until a real retrieval or browsing validation layer is integrated.

5. Move from anonymous client identifiers to account identity before enforcing paid cross-device quota limits.

6. Keep server-side quota and billing checks authoritative; client-side quota counters are not acceptable as the enforcement layer.

7. Conduct a formal data-flow inventory and produce a retention policy that matches the implementation before publishing privacy claims.

7. Implications

7.1 For Practitioners and Users

Prompt Rewriter lowers the barrier to high-quality LLM output for non-expert users. By automating role specification, task framing, and format selection, it transfers a significant portion of the prompt engineering skill burden from the user to a structured compilation layer. Users retain full control: the original draft is never modified without explicit action, and two meaningfully different candidates are always presented for comparison.

7.2 For Researchers and Evaluators

The failure taxonomy and structured feedback schema represent a methodological contribution to LLM evaluation. Most current evaluation frameworks attribute all quality failures to the underlying model. By separating routing failures, compiler failures, user-model mismatches, and downstream model failures, the system enables more precise identification of where the quality bottleneck actually resides — and therefore where improvement effort should be directed.

7.3 For Platform Developers

The longer-term vision of a typed prompt-state graph — treating the prompt as a structured object with typed nodes for objective, audience, role, constraints, evidence needs, output format, and known unknowns — has solid theoretical grounding in the language model programming paradigm [8] and the DSPy compilation framework [7]. Realizing this vision requires the empirical feedback data that the current implementation is designed to collect. Cross-model compilation — generating model-specific prompt surfaces from a neutral intent representation — is the furthest-horizon goal and carries the risk of becoming brittle if based on assumed rather than measured model behavioral differences.

8. Development Roadmap

The product roadmap is organized around four value delivery phases. Feature maturity scores (1 = research idea; 5 = production-hardened) reflect current implementation state. All maturity scores and phase timelines are

internal planning estimates. [ASSUMPTION]

Phase	Build Target	Strategic Value
Near-term	Analytics Dashboard	Converts Good/Bad feedback into pipeline-specific product decisions
Near-term	Growth & Onboarding	Before/after examples, onboarding guidance, extension store presence
Mid-term	Monetization Layer	Account identity, paid quota, billing, cross-device enforcement
Future	Opt-in Memory	User style profile, domain preferences, saved prompt patterns
Future	Grounded Provenance	User facts / page facts / inferred assumptions as separate typed prompt fields
Future	Clarification Budget	Ask 0–5 questions only when expected quality gain exceeds latency/cost penalty
Future	Cross-Model Compiler	Compile same intent graph differently per target LLM platform

Table 4. Roadmap phases and build targets. Timelines are internal planning estimates.

9. Conclusion

Prompt Rewriter addresses a well-documented gap in consumer LLM interfaces: the systematic quality bottleneck created by structurally impoverished prompt input. By operating as a browser-native, human-in-the-loop compilation layer, the system transforms underspecified user drafts into two materially different, domain-appropriate candidate prompts — without requiring the user to develop prompt engineering expertise.

The current implementation is designed for free-tier consumer launch. Its strongest architectural path forward is not incremental feature addition but rigorous measurement: a feedback analytics dashboard that attributes failures to routing, compilation, source-mode compliance, option divergence, or downstream model behavior. This failure taxonomy — absent from most LLM evaluation frameworks — is the system's primary technical contribution.

The longer-term vision of a typed prompt-state graph with clarification-budget optimization and cross-model compilation has solid theoretical grounding in prompting research [2], DSPy [7], and language model programming [8]. Realizing it requires the empirical feedback data that the current implementation is positioned to collect.

Guiding principle: ship narrow, measure aggressively, and let real feedback drive skill and compiler improvements. Avoid claiming verified sourcing or durable memory before those features have real retrieval, provenance, retention, and account controls in place.

References

All references [1]–[8] are real peer-reviewed works. No sources are invented or assumed.

[1] J. D. Zamfirescu-Pereira, R. T. Q. Wong, B. Hartmann, and Q. Yang, "Why Johnny Can't Prompt: How Non-AI Experts Try and Fail to Design LLM Prompts," in Proc. ACM CHI Conf. Human Factors in Computing Systems (CHI), Hamburg, Germany, 2023, pp. 1–21. doi:10.1145/3544548.3581388

[2] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," ACM Comput. Surv., vol. 55, no. 9, pp. 1–35, Sep. 2023. arXiv:2107.13586

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, 2022, pp. 24824–24837. arXiv:2201.11903

[4] 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," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023. arXiv:2203.11171

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023. arXiv:2210.03629

[6] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large Language Models Are Human-Level Prompt Engineers," in Proc. Int. Conf. Learning Representations (ICLR), Kigali, Rwanda, 2023. arXiv:2211.01910

[7] O. Khattab et al., "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines," in Proc. Int. Conf. Learning Representations (ICLR), Vienna, Austria, 2024. arXiv:2310.03714

[8] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Prompting Is Programming: A Query Language for Large Language Models," in Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI), Orlando, FL, USA, 2023, pp. 1946–1969. arXiv:2212.06094

Disclosure: This document is a public adaptation of an internal technical report (v2.0, May 2026). Proprietary implementation details including specific codebase paths, internal tooling identifiers, exact quota thresholds under active revision, and confidential operational metrics have been removed or generalized. Planning assumptions are explicitly labeled throughout. Claims derived from the production codebase are labeled [Implementation Fact]. All peer-reviewed citations are real works and are listed in full in the References section.