CASPER: Bridging Discrete and Continuous Prompt Optimization through Feedback-Guided Gradient Descent

Aryan Jain, Pushpendu Ghosh, Promod Yenigalla

RBS Tech Sciences, Amazon

{arynjn, gpushpen, promy}@amazon.com

Abstract

Workflow automation is critical for reducing manual efforts in industries, yet existing pipelines fail to handle generative tasks like summarization and extraction without pre-built tools, forcing human intervention. While LLM-based agents offer solutions, their creation depends heavily on prompt engineering—a resource-intensive process often yielding suboptimal results. Current automated approaches face a fundamental trade-off: discrete optimization produces overfitted prompts without convergence guarantees due to non-convex landscapes, while continuous gradient-based methods generate semantically incoherent prompts through embedding optimization. We propose CASPER, a framework bridging discrete and continuous prompt optimization through feedback-guided gradient descent in embedding space. CASPER employs a feedback module producing detailed error analyses that capture failure modes as optimization signals. These insights are projected with prompt tokens into embedding space to steer gradient descent. To preserve interpretability, we incorporate fluency regularization that penalizes incomprehensible tokens. We further accelerate convergence through synthetic data generation that oversamples failure cases, while also addressing data scarcity in industrial settings. We evaluate CASPER on WDC, DROP, GSM8K with F1 improvements of 2.3%, 1.6%, 2.3% and VQA, internal benchmarks showing accuracy improvements of 1.1%, 3%, demonstrating cross-domain generalizability.

1 Introduction

Modern industries are increasingly shifting towards automation of redundant workflows through agentic solutions. However, they face a fundamental obstacle: workflows now depend on generative AI capabilities—summarization, information extraction, and content generation which are non-deterministic in nature. While Large Language Model (LLM) agents offer a path toward end-to-end automation, their effectiveness critically hinges on prompt engineering, a process demanding extensive manual effort, domain expertise, and costly iterative refinement to achieve task-optimal performance.

This challenge persists despite recent automated prompt optimization advances. Discrete optimization approaches (Zhou et al., 2023; Yang et al., 2024) iteratively refine prompts through LLM-generated feedback, but the non-convex optimization landscape offers limited control, frequently yielding overly complex, suboptimal prompts without convergence guarantees. Continuous optimization methods (Wen et al., 2023; Pryzant et al., 2023) learn soft prompt embeddings through gradient-based optimization, enabling targeted search through continuous embedding space. However, these produce model-specific prompts that hamper cross-model portability and often yield incomprehensible results, impacting scalability and interpretability.

Our key insight is that discrete and continuous prompt optimization approaches are complementary; by representing prompts as continuous embeddings and optimizing via gradients while injecting the LLM's textual feedback about its errors to accelerate convergence to the optimal prompt, we treat it as a co-optimization problem where textual feedback guides gradient descent toward effective convergence.

Our main contributions include: 1) A novel framework combining textual feedback as optimization signals for gradient-based prompt refinement, enabling faster convergence than continuous methods. 2) A failure amplification data generation strategy that synthetically boosts error case distributions, accelerating gradient optimization while also making it achievable with sparse industrial datasets. 3) A fluency-preserving loss function that penalizes random token insertion during gradient descent, ensuring optimized prompts remain com-

prehensible and editable.

Comprehensive evaluation across diverse benchmarks — WDC, DROP, GSM8K and VQA demonstrate broad applicability showing substantial improvements in execution accuracy with lesser rollout budget.

2 Related Works

Discrete prompt optimization: These methods iteratively refine prompts through search-based strategies. APE (Zhou et al., 2023) employs Monte Carlo search, OPRO (Yang et al., 2024) frames optimization as meta-optimization using error feedback, EvoPrompt (Guo et al., 2024) applies evolutionary algorithms, and PromptAgent (Wang et al., 2024) uses expert trajectories. DSPy (Khattab et al., 2024) introduces a programmatic framework compiling declarative pipelines into optimized prompts through bootstrapping. However, the exponential token space creates non-convex landscapes prone to local minima, often requiring hundreds of iterations without convergence, producing overly complex prompts that overfit and generalize poorly (Fernando et al., 2024).

Continuous prompt optimization: Gradient-based methods address discrete optimization's inefficiencies by operating in continuous embedding space. Prefix-tuning (Li and Liang, 2021) and prompt-tuning (Lester et al., 2021) learn soft prompts through backpropagation, while AutoPrompt (Shin et al., 2020) uses gradient-guided token substitution. BBT (Sun et al., 2022) and BDPL (Deng et al., 2022) perform black-box gradient estimation. These methods produce embeddings lacking semantic coherence that cannot be decoded into interpretable prompts (Wen et al., 2024), hindering cross-model transfer (Khashabi et al., 2022).

Recent works try to bridge both paradigms. GrIPS (Prasad et al., 2023) alternates between gradient descent and discrete projection but struggles with fluency. InstructZero (Pryzant et al., 2023) combines Bayesian optimization with soft tuning but requires extensive meta-learning. RLPrompt (Deng et al., 2022) uses reinforcement learning, though credit assignment remains challenging.

3 CASPER

Let $\mathcal{D}_{\text{seed}} = \{(x_i, y_i)\}_{i=1}^{N}$ denote a seed dataset of input contexts x and task-specific targets y. Our goal is to synthesize a prompt $P = (t_1, \ldots, t_L)$ from a given task description t that, when provided to an LLM M, maximizes task performance. We represent prompts in both discrete token space and as continuous embeddings $z = \phi(P) \in \mathbb{R}^d$, enabling gradient-based optimization while incorporating discrete textual feedback signals f from the LLM on failure cases. The optimization objective is:

$$\min_{z} \; \mathcal{L}_{\text{task}}(z; \mathcal{D}) + \lambda_{\text{fluency}} \mathcal{L}_{\text{fluency}}(z) \qquad (1)$$

where $\mathcal{L}_{\text{task}}$ is the primary task loss and $\mathcal{L}_{\text{fluency}}(z)$ penalizes uninterpretable tokens.

Figure 1 illustrates the CASPER framework architecture. In workflow automation, individual steps explicitly specify actions (e.g., Produce concise summaries of positive signals and highlight root causes of negative signals). We use this description as our initial prompt P_0. since individual steps may lack sufficient context about the task.

The CASPER framework consists of three interconnected modules operating iteratively:

1. **Feedback Generation Module ($\mathcal{M}_f$):** Analyzes error cases $\mathcal{E}_i = \{(x_j, y_j, \hat{y}_j) \mid \hat{y}_j = M_\theta(x_j, P_i), \hat{y}_j \neq y_j\}$ to generate textual feedback $f^{(i)} = \mathcal{M}_f(\mathcal{E}_i, P_i \mid D_i)$ describing common failure patterns and potential improvements.

2. **Failure Amplification Data Generation Module ($\mathcal{M}_D$):** Augments the seed dataset to mitigate data sparsity while expanding the distribution of failure scenarios via $D_{train} = D_{\text{seed}} \cup \mathcal{M}_D(D_{\text{success}}, \mathcal{E}_i)$, where $\mathcal{M}_D$ samples synthetic examples (x', y') informed by the current error distribution.

3. **Soft Prompt Optimization Module ($\mathcal{M}_s$):** Optimizes the prompt by leveraging textual feedback and augmented data. The optimization is performed in continuous embedding space $\phi(P) \in \mathbb{R}^d$ using gradient descent.

3.1 Feedback Generation Module

Recent studies demonstrate that textual feedback from LLMs regarding their failures significantly enhances prompt optimization, providing rich learning signals when combined with quantitative metrics (Pryzant et al., 2023; Fernando et al., 2023). This feedback provides several advantages: (1) it acts as a signal offering directional guidance by identifying critical points of error in the prompt, (2) it provides a comprehensible way of changing

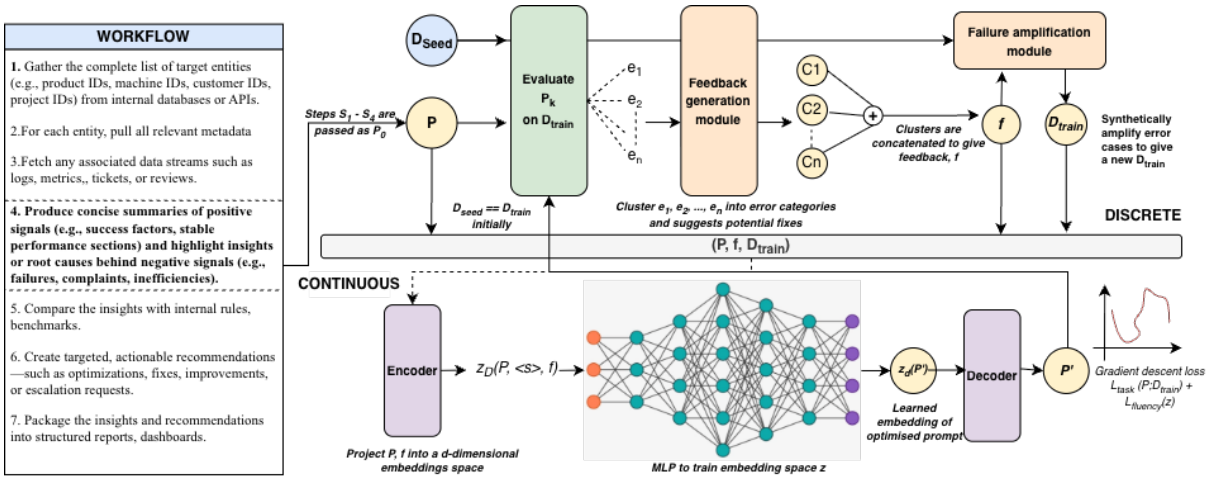

Figure 1: Illustration of how CASPER bridges the gap between continuous and discrete optimization, enabling the generation of more optimal prompts.

the prompt allowing for better knowledge on how prompt evolution is happening, and (3) it provides interpretable optimization trajectories for tracing decisions regarding prompt updates.

This motivates us to use the textual feedback signal as a source for guiding the gradient descent and enable faster convergence.

Formulation. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ denote input-output pairs where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Let $P_k \in \mathcal{P}$ denote the prompt at iteration k in the discrete prompt space $\mathcal{P}$. The target LLM is modeled as $M_\theta : \mathcal{P} \times \mathcal{X} \to \mathcal{Y}$ with fixed parameters θ.

At iteration k, we evaluate P_k on batch $\mathcal{D}_k \subseteq \mathcal{D}$ to obtain predictions $\hat{y}_i = M_\theta(P_k, x_i)$. Using task-specific metric $\mathcal{E} : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$, we identify failures:

$$\mathcal{F}_k = \{(x_i, y_i, \hat{y}_i) \mid \mathcal{E}(y_i, \hat{y}_i) < t_h,\ (x_i, y_i) \in \mathcal{D}_k\} \tag{2}$$

where t_h is the threshold below which a prediction is classified as erroneous. A critic LLM $\mathcal{M}_\phi$ then generates feedback:

$$f^{(k)} = \mathcal{M}_\phi(P_{\text{critic}}, P_k, \mathcal{F}_k) \tag{3}$$

where P_{critic} instructs the critic to analyze failures and $f^{(k)}$ is the textual feedback. This feedback, combined with performance metrics, informs P_{k+1} generation until convergence.

Feedback format. We cluster failures into categories with descriptions, patterns, and examples that guide correction. The critic prompt is in Appendix A.5. Each cluster follows:

Error Cluster Template

Cluster [number]: Temporary name based on error pattern
Pattern: Common error pattern description
Error Samples: Cases belonging to this cluster
Key Features: Distinctive characteristics

The idea behind clustering errors in this way is to reduce the complexity in textual feedback while incorporating all error scenarios in a compact manner.

3.2 Failure Amplification Module

Gradient descent optimization generally requires large training samples and significant iterations to converge which is impractical for industrial workflow automation where testing and failing quickly are key to building new systems. To accelerate convergence and reduce rollout budgets, we propose a failure amplification module which oversamples error-prone cases. Let $\mathcal{D}_{\text{seed}} = \{(x_i, y_i)\}_{i=1}^{N}$ denote the original dataset.

Implementation: Given prompt P_0 and seed dataset $\mathcal{D}_{\text{seed}}$, we identify initial failures $\mathcal{F}_k$ as given by eq 2. We then construct the amplified training set by sampling with replacement:

$$\mathcal{D}_{\text{train}} = \mathcal{D}_{\text{seed}} \cup \text{Sample}(\mathcal{D}_{\text{success}}, \alpha \cdot |\mathcal{D}_{\text{error}}|) \tag{4}$$

where $\alpha \geq 0$ determines the replication factor and $Sample$ is a sampling function where we randomly select samples from the distribution with replacement. This biases the optimization process toward correcting systematic errors while maintaining diversity from correct examples.

Validation and Hyperparameter Selection: We retain $\mathcal{D}_{\text{seed}}$ as the validation set to ensure performance is measured on the original data distribution, preventing overfitting to the modified distribution and providing an unbiased estimate of generalization. The resampling ratio α balances error sample weightage against distribution preservation: higher values accelerate convergence but risk distribution shift, while lower values maintain the original distribution but may require more iterations.

3.3 Soft prompt optimization module

Gradient based methods have shown to produce more targeted and optimal prompts in the past. The idea of learning embeddings by minimising an objective loss function and projecting those learned embeddings to the discrete token space to give an optimal prompt has shown promising results. We use these works as motivation to build upon our solution.

Embedding Projection. We map the discrete prompt P_k and feedback f^k into a shared continuous embedding space. Let $\mathcal{E}_{\text{enc}} : \mathcal{L} \to \mathbb{R}^d$ denote a learnable encoder that projects natural language text into a d-dimensional latent space:

$$\mathbf{z}_k = \mathcal{E}_{\text{enc}}(P_k), \quad \mathbf{z}_f = \mathcal{E}_{\text{enc}}(f^{(k)}) \tag{5}$$

where $\mathbf{z}_k, \mathbf{z}_f \in \mathbb{R}^d$ are the prompt and feedback embeddings, respectively.

Optimization Architecture. We construct a composite representation by concatenating prompt and the textual feedback, separated by a fixed token $< s >$:

$$\mathbf{z}_{\text{combined}} = [\mathbf{z}_k;\ \mathcal{E}_{\text{enc}}(< s >);\ \mathbf{z}_f] \tag{6}$$

This combined embedding is passed through an encoder-decoder MLP $\mathcal{G}_\psi$ parameterized by ψ:

$$\mathbf{z}_{k+1} = \mathcal{G}_\psi(\mathbf{z}_{\text{combined}}) \tag{7}$$

The output embedding $\mathbf{z}_{k+1}$ is then decoded back to natural language via a learned decoder $\mathcal{D}_{\text{dec}} : \mathbb{R}^d \to \mathcal{L}$ to obtain the refined prompt:

$$P_{k+1} = \mathcal{D}_{\text{dec}}(\mathbf{z}_{k+1}) \tag{8}$$

Training Objective. We optimize the encoder-decoder parameters $\{\psi, \theta_{\text{enc}}, \theta_{\text{dec}}\}$ via backpropagation to minimize a task-specific loss $\mathcal{L}_{\text{task}}$ evaluated on training set $\mathcal{D}_{\text{train}}$:

$$\min_{\psi,\theta_{\text{enc}},\theta_{\text{dec}}} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{train}}} \left[\mathcal{L}_{\text{task}}(y, \mathcal{M}_\theta(P_{k+1}, x))\right] \tag{9}$$

This gradient-based approach enables end-to-end learning of the embedding space and transformation function, directly optimizing for task performance while incorporating structured feedback from the critic model. For text-to-text tasks, we employ cosine similarity as the primary loss function. Let $\mathbf{e}_y$ and $\mathbf{e}_{\hat{y}}$ denote the embeddings of ground-truth output y and predicted output $\hat{y} = \mathcal{M}_\theta(P_{k+1}, x)$. The task loss is:

$$\mathcal{L}_{\text{task}}(y, \hat{y}) = 1 - \frac{\mathbf{e}_y \cdot \mathbf{e}_{\hat{y}}}{\|\mathbf{e}_y\|\|\mathbf{e}_{\hat{y}}\|} \tag{10}$$

However, soft prompt optimization is prone to generating random tokens that render prompts uninterpretable. We provide an example for this in Table 8 in Appendix. This occurs due to the disconnect between the learned continuous embedding space and the discrete token space—optimized embeddings may drift to regions far from any valid token embedding. To address this, we introduce a fluency regularization term that constrains learned embeddings to remain proximal to the discrete token manifold.

Let $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_{|\mathcal{V}|}\}$ denote the vocabulary embedding matrix. We use the same vocabulary as used by the encoder and decoder to maintain uniformity. For each learned embedding $\mathbf{z}_{k+1}$, we compute the distance to its nearest token embedding:

$$\mathcal{L}_{\text{fluency}}(\mathbf{z}_{k+1}) = \min_{\mathbf{v}\in\mathcal{V}} \|\mathbf{z}_{k+1} - \mathbf{v}\|^2 \tag{11}$$

The complete training objective combines task performance with fluency regularization:

$$\min_{\psi,\theta_{\text{enc}},\theta_{\text{dec}}} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{train}}} \Big[\mathcal{L}_{\text{task}}\big(y, \mathcal{M}_\theta(P_{k+1}, x)\big) + \lambda_{fluency}\, \mathcal{L}_{fluency}(\mathbf{z}_{k+1})\Big] \tag{12}$$

where $\lambda > 0$ controls the strength of the fluency constraint.

4 Experiments

We evaluate CASPER across five diverse datasets:

Text-based Tasks: (1) **WDC Product Corpus** (Brinkmann et al., 2024) for product attribute extraction from e-commerce descriptions; (2) **DROP** (Dua et al., 2019) for reading comprehension requiring discrete reasoning over text passages, including numerical operations and multi-hop reasoning; (3) **GSM8K** (Cobbe et al., 2021) for multi-step mathematical reasoning problems

Dataset	Metrics	Manual Prompting					Discrete Prompt Optimisation			Soft Prompt Optimisation		CASPER*	CASPER
		Claude 4.0 Sonnet	Claude 3.7 Sonnet	Claude 3.5 Haiku	DeepSeek+	GPT-4o	IPC	APE	OPRO	PEZ	RLPrompt	(w/o reg.)	
WDC	P	82.3	81.7	78.5	79.2	83.1	84.2	85.7	86.4	87.8	88.9	90.2	87.6
	R	79.8	80.2	76.4	77.8	81.2	82.5	83.9	84.7	86.3	87.5	89.1	86.8
	F1	81	81	77.4	78.5	82.1	83.3	84.8	85.5	87	87.2	89.6	87.2
DROP	P	90.2	91.5	87.3	88.6	92.1	91.8	92.5	93.2	94.1	94.8	95.7	94.3
	R	88.7	89.8	85.9	87.1	90.4	90.2	91.1	91.7	93	93.3	94.5	92.9
	F1	89.4	90.6	86.6	87.8	91.2	91	91.8	92.4	92.3	93.5	95.1	93.6
VQA	Acc	76.8	77.5	72.1	73.9	78.2	78.9	80.3	81.5	82.7	83.6	85.9	84.1
GSM8K	Acc	92.8	93.2	88.5	90.1	92	93.5	94.2	95	95.8	96.2	97.3	96.5
Internal	Acc	71.3	72.8	68.2	69.5	73.4	74.1	75.8	77.2	78.4	79.6	82.6	80.7

Table 1: Comparison of CASPER with other state-of-the-art approaches. CASPER* denotes ablation without fluency regularization. DeepSeek+ refers to DeepSeek-R1-Distill-Qwen-32B. While CASPER* achieves highest accuracy through unrestricted continuous optimization, CASPER (Full) trades marginal performance for interpretable prompts

requiring arithmetic computation and logical reasoning chains. **Vision-Language Tasks:** (4) **VQA** (Agrawal et al., 2016) for visual question answering over image-question pairs (5) **Expiry-Date** (internal) for identifying expiration dates from product images in various formats, comprising 150 annotated samples with varying image quality and orientations. We randomly sample 100 instances from the training set of each publicly available dataset and use official test sets. For Expiry-Date, we use a 100/50 train-test split.

5 Results and Discussions

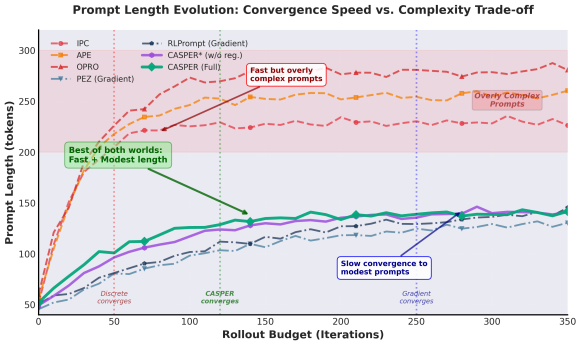

Figure 2: Prompt length evolution demonstrating CASPER's efficiency-quality trade-off. Discrete methods (dashed) rapidly converge to complex prompts (220-280 tokens, <50 iterations), while gradient methods (dash-dot) slowly reach 140 tokens in 250 iterations. CASPER achieves comparable quality in 120 iterations—52% fewer than continuous optimization.

CASPER achieves superior performance without fluency regularization. Table 1 demonstrates that CASPER without fluency loss outperforms all state-of-the-art methods across both discrete and continuous prompt optimization paradigms, achieving 2.4% improvement on WDC F1, 1.6% on DROP F1, 2.3% on VQA, 1.1% on GSM8K and 3% on our internal dataset. Beyond accuracy gains, CASPER exhibits faster convergence compared to continuous optimization methods given in Table 3 while generating more compact prompts than discrete approaches as shown in Figure 2.

Fluency weightage, λ	WDC (F1)	DROP (F1)	VQA (Acc)	GSM8K (Acc)	Internal (Acc)
0	**89.6**	**95.1**	**85.9**	**97.3**	**82.6**
0.2	89.2	94.7	85.6	97.1	82.3
0.4	88.5	94.2	84.8	96.8	81.7
0.6	87.2	93.6	84.1	96.5	80.7
0.8	86.8	93.1	83.5	96.2	80.1
1	86.3	92.5	83.2	95.8	79.5

Table 2: Impact of fluency loss on performance with Claude 4.0 Sonnet. Increasing λ reduces performance while improving prompt interpretability. $\lambda = 0.6$ balances reasonable performance with readable prompts.

Fluency regularization trades performance for interpretability. Incorporating fluency loss consistently degrades CASPER's performance across all datasets, suggesting that optimal embeddings lie distant from discrete token representations, implying a trade-off between prompt comprehensibility and task performance. Table 2 quantifies this trade-off on all the datasets, showing how increasing regularization strength improves interpretability at the cost of accuracy.

Textual feedback stabilizes gradient-based optimization. Figure 3 compares optimization trajectories under different configurations: textual feedback, failure amplification, and fluency loss. Textual feedback provides the strongest stabilization effect, substantially accelerating convergence. While failure amplification also improves conver-

Method	IPC	APE	OPRO	PEZ	RLPrompt	CASPER*	CASPER
WDC	150	180	220	420	580	280	320
DROP	140	170	200	380	520	250	290
VQA	160	190	240	450	610	300	340
GSM8K	130	160	190	360	490	240	270
Internal	170	200	250	480	640	320	360

Table 3: Rollout budget comparison for convergence across datasets. Discrete methods are most sample-efficient but achieve lower final accuracy while continuous methods (PEZ, RLPrompt) require 2-4× more budget. CASPER uses only 50% of continuous optimization budget giving superior performance while fluency loss adds a 12-15% overhead.

Model		WDC		DROP		VQA		GSM8K		Internal	
	$\mathcal{L}_f =$	0.6	0	0.6	0	0.6	0	0.6	0	0.6	0
Claude 4.0 Sonnet		87.2	89.6	93.6	95.1	84.1	85.9	96.5	97.3	80.7	82.6
Claude 3.7 Sonnet		84.8	81.2	91.5	88.9	81.6	78.3	95.2	93.1	78.9	75.4
GPT-4o		85.3	82.5	92.1	89.7	82.3	79.1	95.8	93.8	79.5	76.2
DeepSeek		82.7	78.9	90.3	86.5	79.8	75.7	94.3	91.2	76.8	72.1

Table 4: Impact of fluency loss on cross-model portability. Prompts with fluency regularization $\mathcal{L}_f$ transfer better to other models despite lower source-model scores. Without $\mathcal{L}_f$, prompts overfit to Claude 4.0 Sonnet, causing 3-5% degradation

gence speed, we attribute this primarily to the more targeted textual feedback it enables. Fluency loss slows convergence relative to variants with textual feedback but still outperforms vanilla gradient descent.

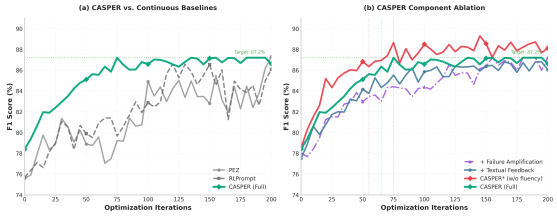

Figure 3: Convergence analysis on WDC dataset. (a) CASPER substantially outperforms continuous baselines (PEZ, RLPrompt), converging faster and to higher F1 scores. (b) Component ablation reveals textual feedback as the primary stabilizing factor, while fluency loss slightly slows convergence.

Fluency regularization enhances cross-model transferability. Table 4 evaluates prompts trained on model A when deployed to other models. Prompts optimized without fluency regularization exhibit poor transfer, indicating model-specific overfitting. In contrast, fluency-regularized prompts maintain near-training-time performance across models, with degradation remaining minimal or absent. This suggests that non-interpretable

Figure 4: Shows the impact of each word in the prompt towards the final predicted output as calculated through the GlobalEnc method on Internal dataset.

tokens (distant from discrete embeddings) encode model-specific idiosyncrasies that fail to generalize, while human-readable prompts capture more universal task semantics.

Non-interpretable tokens contribute meaningfully to model predictions. We analyze token-level importance using the GlobalEnc attribution method (Modarressi et al., 2022), which computes each token's contribution by measuring output sensitivity to perturbations in its embedding. Figure 4 visualizes these importance scores, revealing that non-interpretable tokens generated through continuous optimization make substantial contributions to correct predictions which validates that optimal embeddings need not reside near discrete token manifolds.

Performance gains justify computational overhead. Table 3 shows the rollout budgets required to obtain optimal prompts. While discrete methods require the fewest iterations, they consistently produce inferior prompts. CASPER's moderate computational cost—falling between discrete and continuous baselines—delivers the strongest performance, offering favorable accuracy-efficiency trade-offs for practical deployment.

Conclusion

We present CASPER, a prompt optimization framework bridging discrete and continuous paradigms through gradient-based optimization in embedding space while preserving interpretability. Evaluation across five diverse benchmarks demonstrates CASPER's effectiveness for automated agent creation, reducing manual overhead and enabling rapid cross-domain workflow automation.

Limitations

While CASPER effectively bridges continuous and discrete prompt optimization, CASPER currently supports only single-agent optimization, whereas many industrial workflows require multiple agents coordinating across subtasks. Extending the framework to handle multi-agent interactions and shared optimization remains an important next step.

CASPER also does not yet support tool integration, which is central to real workflow automation. Many tasks depend on selecting and calling external tools or APIs and stitching their outputs together through frameworks like MCP. Incorporating tool planning and execution would make CASPER more suitable for production settings.

Future work could integrate throughput-oriented techniques—such as pruning or caching—to reduce the overhead of running optimized prompts at scale. Overall, these limitations outline clear avenues for extending CASPER toward more comprehensive workflow automation.

References

Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Dhruv Batra, and Devi Parikh. 2016. Vqa: Visual question answering. *Preprint*, arXiv:1505.00468.

Alexander Brinkmann, Nick Baumann, and Christian Bizer. 2024. *Using LLMs for the Extraction and Normalization of Product Attribute Values*, page 217–230. Springer Nature Switzerland.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In *International Conference on Learning Representations*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *Preprint*, arXiv:2309.16797.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*.

Daniel Khashabi, Yashar Kordi, and Hannaneh Hajishirzi. 2022. Prompt waywardness: The curious case of discretized interpretation of continuous prompts. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3631–3643.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.

Ali Modarressi, Mohsen Fayyaz, Yadollah Yaghoobzadeh, and Mohammad Taher Pilehvar. 2022. GlobEnc: Quantifying global token attribution by incorporating the whole encoder layer in transformers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 258–271, Seattle, United States. Association for Computational Linguistics.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models.

In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3845–3864.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235.

Tianxiang Sun, Zhengfu Liu, Xiangyang Yan, Xipeng Qiu, and Xuanjing Huang. 2022. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pages 20841–20855. PMLR.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2024. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2023. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. In *Advances in Neural Information Processing Systems*, volume 36, pages 51008–51025. Curran Associates, Inc.

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*.

A Appendix

A.1 Ablations without different components of CASPER to show each component's impact

We compare CASPER's performance with and without its components: textual feedback, failure amplification and fluency loss to show the importance of each part and its impact on the output. The results are given in Table 5.

Textual feedback is the most critical component, with its removal causing 3.4-4.0 F1 point drops across models and 40-50% increases in rollout budget (from 320-412 to 450-527 rollouts). Without semantic error analyses, gradient descent relies solely on scalar rewards, leading to inefficient exploration of the embedding space.

Failure amplification also proves essential, with its removal degrading performance by 2.1-2.5 F1 points while requiring 8-14% more rollouts. Strategic oversampling of failure cases accelerates convergence, particularly valuable in data-scarce industrial settings.

Removing fluency regularization improves task performance by 2.4 F1 points on Claude 4.0 Sonnet and reduces rollout budget by 12.5% (from 320 to 280) but produces incoherent prompts (see CASPER* in Table 1). The intermediate setting ($\lambda = 0.4$) achieves 1.3 F1 points improvement with only 6.25% reduction in budget (from 320 to 300 rollouts), confirming that moderate regularization ($\lambda = 0.4 - 0.6$) optimally balances performance, interpretability, and convergence efficiency for real-world deployment.

A.2 Error distribution ablation

We experiment with different values of error ratio, α, in the training data distribution to find the optimal balance between successful and failed examples. The error ratio controls the proportion of failure cases in synthetic data generation. The results are shown in Table 6.

Performance peaks at $\alpha = 0.6$, where 60% of training data consists of failure cases. At $\alpha = 0$, with only successful examples, the model lacks exposure to failure modes, resulting in 3.4-4.3 point drops across datasets. This confirms the importance of failure amplification for effective gradient guidance. Conversely, at $\alpha = 1$ with only failures, performance degrades by 2.3-2.8 points as the model loses reference to correct behaviors. The optimal 60:40 failure-to-success ratio provides sufficient failure mode coverage while maintaining positive examples for contrast, enabling CASPER to learn nuanced error boundaries in the embedding space.

Dataset	CASPER		Without Textual Feedback		Without Failure Amplification		Without fluency loss		With fluency loss = 0.4	
	F1	Rollout	F1	Rollout	F1	Rollout	F1	Rollout	F1	Rollout
Claude 4.0 Sonnet	87.2	320	83.8	450	85.1	365	89.6	280	88.5	300
Claude 3.7 Sonnet	81.0	346	77.2	454	78.9	355	89.6	301	88.5	325
DeepSeek+	78.5	412	74.8	527	76.4	462	89.6	429	88.5	408
GPT-4o	82.1	377	78.5	502	80.1	341	89.6	331	88.5	362

Table 5: Performance comparison of CASPER by removing different components on WDC dataset averaged across 5 trials. Rollout budget indicates the number of LLM calls required for convergence. Without fluency loss achieves highest F1 but produces uninterpretable prompts.

Error ratio, α	WDC	DROP	VQA	GSM8K	Internal
0	83.8	90.5	80.2	93.7	76.4
0.2	85.4	92.1	82.3	95.1	78.6
0.4	86.7	93.2	83.6	96.0	79.9
0.6	**87.2**	**93.6**	**84.1**	**96.5**	**80.7**
0.8	86.3	92.8	83.2	95.8	79.8
1	84.9	91.4	81.7	94.6	77.9

Table 6: Analysis of impact of error ratio on the performance of the prompt generated through CASPER with Claude 4.0 Sonnet.

A.3 Fluency loss impact on interpretability

Table 7 shows the impact of fluency loss weightage on incomprehensible tokens in optimized prompts. Without regularization ($\lambda = 0$), continuous optimization produces 15-18% gibberish tokens—semantically incoherent sequences exploiting model artifacts. While achieving highest performance (Table 2), these prompts are uninterpretable and fail cross-model transfer (Table 4).

At $\lambda = 0.6$, gibberish tokens reduce to 3-5% with only 1.5-2.5% performance cost. The fluency term $\mathcal{L}_f$ penalizes low-probability tokens, constraining gradient descent to natural language regions. Increasing λ to 1.0 eliminates gibberish but costs additional 2-3% performance. Simpler tasks like GSM8K show smaller gaps (1.3%) while complex tasks like WDC exhibit larger gaps (2.4%), confirming $\lambda = 0.6$ as optimal.

Fluency weightage, λ	WDC (%)	DROP (%)	VQA (%)	GSM8K (%)	Internal (%)
0	17.8	15.3	18.2	14.7	16.9
0.2	13.5	11.2	14.1	10.8	12.7
0.4	8.9	7.3	9.5	6.7	8.2
0.6	4.7	3.8	5.2	3.1	4.3
0.8	2.1	1.6	2.5	1.3	1.9
1.0	0.6	0.4	0.8	0.3	0.5

Table 7: Proportion of incomprehensible tokens vs. fluency loss weightage λ. At $\lambda = 0.6$, gibberish reduces to 3-5% with minimal performance cost.

A.4 Prompt evolution sample

Table 8, 9 shows how the prompt for the internal dataset of finding the expiry date from product images, evolves across iterations when optimized with CASPER. We include ablations, both with and without fluency loss to show the impact of the same on the generated prompts.

A.5 Feedback Generation module prompt

Cluster formation prompt

```
You are an expert at analyzing and categorizing errors in language model outputs.
Your task is to cluster similar error cases together based on the fundamental nature of the errors,
where each cluster should represent a distinct type of failure mode.

For each error case, you will receive:
1. Input: The original input given to the LLM
2. Prediction: What the LLM output
3. Ground Truth: The correct output that was expected

Task Description: $task

Guidelines for clustering:
- Create clusters based on the root cause or pattern of the error, not surface-level similarities
- Ensure clusters are mutually exclusive - an error should clearly belong to one primary cluster
- Focus on systematic patterns rather than one-off mistakes
- Consider both the type of mistake (e.g., hallucination, missing information) and
the context in which it occurs
- Each cluster should be distinct enough that it could be addressed with a specific intervention

For each error case, please:
1. Analyze the nature of the error
2. Identify the key characteristics that define this type of error
3. Assign it to an existing cluster or create a new cluster if it represents a distinct error pattern
4. Provide a brief explanation of why this error belongs to that cluster

After clustering all cases, list each cluster with:
1. A summary of the common pattern
2. Representative examples
3. Key distinguishing features from other clusters

Output Format:
<output>
<cluster>
{
    "Cluster #[number]": [Temporary cluster name based on error pattern],
    "Pattern": [Brief description of the common error pattern],
    "Error_Samples": [List of all error cases that belong to this cluster.],
    "Key_Features": [What makes this cluster unique],
    "Index": "[Indices of all the input samples belonging to this cluster]"
}
</cluster>
</cluster> ... </cluster>
</output>
```

A.6 Failure amplification module prompt

Synthetic data generation

```
You are tasked with generating synthetic training examples to augment a dataset for prompt
optimization. Your goal is to create examples that are similar to identified failure cases to
accelerate model convergence on difficult instances.

**Context:**
We are optimizing prompts for the following task: [TASK_DESCRIPTION]

**Seed Dataset Examples:**
[3-5 REPRESENTATIVE EXAMPLES FROM seed dataset]

**Identified Failure Cases:**
[CURRENT FAILURE CASES WITH INPUT-OUTPUT PAIRS AND ERROR DESCRIPTIONS]

**Your Task:**
Generate [N] new synthetic examples that exhibit similar characteristics to the
failure cases while introducing controlled variations. For each synthetic example:

1. **Identify the target failure mode**: Select one of the identified failure
   patterns above
2. **Create a challenging input**: Design an input that would likely trigger
   this failure mode, incorporating:
   - Similar structural patterns to failed cases
   - Edge cases and boundary conditions
   - Realistic variations (noise, ambiguity, multiple candidates)
3. **Provide the ground truth output**: Give the correct expected output
4. **Explain the difficulty**: Briefly describe why this example is challenging
   and which failure mode it targets

**Output Format:**
For each synthetic example, provide:
Example [N]:
- Input: [Generated input]
- Expected Output: [Ground truth]
- Failure Mode Targeted: [Which failure pattern this addresses]
- Difficulty Explanation: [Why this is challenging - 1-2 sentences]

**Quality Constraints:**
- Examples must be realistic and plausible for the domain
- Maintain diversity: don't generate near-duplicates of existing failures
- Balance difficulty: examples should be challenging but solvable with better prompts
- Ensure ground truth labels are unambiguous and correct

**Generate [N] synthetic examples following these guidelines.**
```

Iter	Prompt (Without Fluency Reg.)	Acc
1	Extract the expiry date from the product label image.	52%
10	Extract the expiry date from the product label image. Look carefully at the label and identify text containing date informationflumexisiehs Search for common expiry indicators including "EXP", "Best Before", "USE BY", "Expiry Date", "BB" ... Pay attention to the positioning of dateshhasgdhvexoral on the label. Manufacturing dates often appear with labels like "MFG", "Manufactured on", or "Production Date" - these are different from expiry dates. zynthiche expiry date is typically what you need to find. When you locate dates, gravmont determine which one represents when the product expires. Consider that dates might be written in various formats such as DD/MM/YYYY, MM/DD/YYYY, or with abbreviated month names. pleximarmonj Some labels may have dates in non-standard positions or orientations. Check differentskelrix areas of the label including corners, edges, and sides. Return the expiry date you identify in YYYY-MM-DD format. **[6 gibberish tokens]**	73%
100	Carefully analyze the product label image to locate and extract the expiry date. luxvasdasdcxdfar Follow this systematic approach: First, scan the entire label for expiry date indicators. Common phrases include "EXP", "Best Before", "USE BY", "BB", "Expiry Date" ... vriminalinexpas and similar variations ... When examining the label, you will likely encounter multiple dates. nexthroskjsks It is critical to distinguish between manufacturing dates and expiry dates. Manufacturing dates are usually marked with "MFG", "Manufactured", "Production Date" ... kassorthen Expiry dates indicate when the product should no longer be used ... Pay attention to spatial relationships between dates on the label. gravshiftIn most cases, expiry dates appear below, after, or to the right of manufacturing dates. vrimesh. However, this is not always consistent ... Date formats vary significantly. plendoral You may encounter formats such as DD/MM/YYYY, MM/DD/YYYY ... or dates with abbreviated or full month names like "15-Jan-2025" ... flaxidor When you see ambiguous numeric dates like "03/05/24", praximal default to interpreting them as DD/MM/YYYY unless contextual clues suggest otherwise. Some labels print dates in non-standard orientations. skelmorph Check for vertically printed text, diagonal text ... or dates printed on label edges and seams. zynthwave Examine the entire visible surface systematically. vexthros Once you identify the expiry date, convert it to YYYY-MM-DD format. If no expiry date can be found, return "NOT FOUND" with a brief explanation. **[14 gibberish tokens]**	89%
280	Systematically analyze the product label image to accurately identify and extract the expiry date. nexiform Follow these detailed steps: **Step 1:** vexalith Examine the entire visible label surface carefully ... gravion Expiry dates can appear anywhere on the label including front panels, side panels, bottom edges, caps, or seals. **Step 2:** plexmor Search for common expiry date markers: "EXP", "Expiry", "Best Before" ... zynthcore **Step 3:** luxmorph Product labels often display multiple dates. skelthren Carefully differentiate between Manufacturing dates (labeled "MFG", "Manufactured" ...) vrimthros and Expiry Dates ... korthenix Manufacturing dates indicate when the product was made; expiry dates indicate when it expires ... flaxvern **Step 4:** praxithol Dates appear in various formats: DD/MM/YYYY, MM/DD/YYYY ... text formats like "15-Jan-2025" ... gravex For ambiguous dates (e.g., "03/05/24"): nexalith Default to DD/MM/YYYY. Consider product origin. vexthral Look for contextual clues. **Step 5:** plexion Dates may be printed vertically, skelmorion diagonally, on curved surfaces, zynthex embossed/debossed ... luxthren Systematically check all visible areas. **Step 6:** vrimshore If partially obscured: reconstruct from visible digits. korthal Use contextual logic with reasonable shelf-life ranges. flaxthros Common shelf lives: beverages (6-24 months) ... **Step 7:** praxven Convert to YYYY-MM-DD format. gravshore Ensure logical validity. nexvorn If not found: Return "NOT FOUND". vexilith Provide brief explanation. **Edge Cases:** skelven Multiple expiry dates: extract earliest ... zynthral Batch codes: parse carefully. plexthral "Best Before" vs "Use By": prioritize "Use By". **[24 gibberish tokens]**	**94%**

Table 8: Prompt evolution without fluency regularization for expiry date extraction. Red tokens are incomprehensible insertions from gradient descent in embedding space. Gibberish token count increases (0→6→14→24) as optimization prioritizes performance over interpretability, achieving 94% final accuracy. We include truncated version of the prompt, only showing important details.

Iter	Prompt (With Fluency Reg.)	Acc
1	Extract the expiry date from the product label image.	52%
10	Extract the expiry date from the product label image. Look carefully at the label and identify text containing date information. Search forexpdatsies common expiry indicators including "EXP", "Best Before", "USE BY", or "dasdasdsaExpiry Date". Pay attention to the positioning of dates on the label. Manufacturing dates often appear with labels like "MFG" or "Manufactured on" - these are different from expiry dates. iuyweruysdg When you locate dates, determine which one represents when the product expires. Consider that dates might be written in various formats such as DD/MM/YYYY or MM/DD/YYYY. Check different areas of the label including corners and edges. Return the expiry date in YYYY-MM-DD format. **[3 gibberish token]**	68%
100	Carefully analyze the product label image to locate and extract the expiry date. Follow this systematic approach: First, scan the entire label for expiry date indicators. Common phrases include "EXP", "Best Before", "USE BY", "BB", "Expiry Date", "Expires", "Best By", and similar variations. hhdgsudus These indicators are typically placed near the actual date information. When examining the label, you will likely encounter multiple dates. It is critical to distinguish between manufacturing dates and expiry dates. Manufacturing dates are usually marked with "MFG", "Manufactured", "Production Date", or "Made on". Expiry dates indicate when the product should no longer be used, while manufacturing dates indicate when it was produced. Pay attention to spatial relationships between dates on the label. In most cases, expiry dates appear below, after, or to the right of manufacturing dates. However, this is not always consistent acrossghbggfgfh all product types and regions. Date formats vary significantly across different regions and manufacturers. You may encounter opitumklonformats such as DD/MM/YYYY, MM/DD/YYYY, YYYY-MM-DD, or dates with abbreviated or full month names like "yuitrks15-Jan-2025" or "March 20, 2024". When you see ambiguous numeric dates like "03/05/24", default to interpreting them as DD/MM/YYYY unless contextual clues suggest otherwise. Some labels print dates in non-standard orientations. Check for vertically printed text, diagonal text, text along curved surfaces, or dates printed on label edges and seams. Examine the entire visible surface of the product label systematically. Once you identify the expiry date, convert it to the standardized YYYY-MM-DD format. If no expiry date can be found on the label, return "NOT FOUND" along with a brief explanation. **[4 gibberish token]**	84%
320	Systematically analyze the product label image to accurately identify and extract the expiry date. Follow these detailed steps: **Step 1: Initial Label Scan** Examine the entire visible label surface carefully. Look for any text, numbers, or symbols that might indicate date information. Expiry dates can appear anywhere on the label including front panels, side panels, bottom edges, caps, or seals. **Step 2: Identify Expiry Date Indicators** Search forinresthib common expiry date markers and phrases: "EXP", "Expiry", "Expiry Date", "Expires", "Best Before", "Best By", "BB", "USE BY" ... **Step 3: Distinguish Between Date Types** Product labels often display multiple dates. Carefully differentiate between Manufacturing/Production Dates (labeled "MFG", "Mfg Date", "Manufactured" ...) and Expiry Dates (the date when the product should no longer be consumed or used). Key distinction: Manufacturing dates indicate when the product was made; expiry dates indicate when it expires. On most labels, manufacturing dates appear chronologically before expiry dates, and spatially they often appear above, to the left, or before the expiry date. **Step 4: Handle Ambiguous Date Formats** Dates appear in various formats: Numeric formats (DD/MM/YYYY, MM/DD/YYYY ...), textfopertys formats ("15-Jan-2025", "January 15, 2025") ... For ambiguous numeric dates (e.g., "03/05/24"): Default to DD/MM/YYYY interpretation; consider product origin (US products may use MM/DD/YYYY); look for contextual clues. **Step 5: Check Non-Standard Orientations** Dates may be printed: vertically along side edges, diagonally or curved on cylindrical packages, as small print on caps/necks/seals, embossed or debossed ... Systematically check all visible areas and orientations. **Step 6: Handle Partial or Degraded Text** If the expiry date is partially obscured, faded, or damaged: Attempt to reconstruct missing digits from visiblechisofdod portions; use contextual logic (expiry dates should be after manufacturing dates and within reasonable shelf-life ranges). **Step 7: Format and Return Result** Once identified: Convert to YYYY-MM-DD format; ensure the dateplkskfjuj is logically valid ... If no expiry date can be confidently identified: Return "NOT FOUND" with brief explanation. **[4 gibberish tokens]**	**91%**

Table 9: Prompt evolution with fluency regularization for expiry date extraction. Red tokens are rare incomprehensible insertions that fluency regularization progressively eliminates. Gibberish token count remains less (0→3→4→4) as fluency constraints enforce natural language, achieving 91% final accuracy while maintaining complete human interpretability.