A Single Rewrite Suffices: Empirical Lessons from Production Skill Description Optimization

Yangqiaoyu Zhou, Mohammad Alqudah, Kwei-Herng Lai, Aaron Halfaker
Yingqi Xiong, Yaar Harari
Microsoft
{yangqzhou, yaarharari}@microsoft.com

Abstract

Enterprise AI agents route user queries to specialized skills by matching queries against natural language skill descriptions. When two skills share overlapping descriptions, the routing LLM misroutes queries, a failure we term skill collision. As agents scale to dozens of skills, manually tuning descriptions to maintain routing accuracy becomes a significant engineering bottleneck. We deploy an automated description optimization pipeline on a production enterprise group chat agent (9 skills, 372 regression cases). The pipeline produces descriptions averaging 79.2% F1, matching manually tuned descriptions at 79.4% F1 (average per-skill difference -0.20%, within the $\pm 0.78\%$ multi-seed noise floor), while reducing per-skill engineering effort from 120 minutes to 3.8 minutes ($32\times$ speedup). We then examine which pipeline components actually drive this match. Systematic ablation on both the production system and ToolBench ($\sim$16k tools) reveals that a single LLM rewrite using any available false-positive and false-negative cases captures most of the available improvement. Other design choices we tested (iteration budget, feedback signal composition, dual editing of confused pairs, and training set size) each affect final F1 by less than 0.5%. Description optimization addresses skill collisions caused by overlapping descriptions but cannot resolve cases where two skills' intended scopes genuinely overlap. We identify a diagnostic (a large train–validation F1 gap) that flags the latter cases for architectural rather than text-level intervention.

1 Introduction

Modern enterprise AI assistants are built as orchestrated agent systems where an LLM-based planner routes user queries to specialized skills (Yao et al., 2023; Wu et al., 2023). Routing decisions are driven by comparing the query against each skill's natural language description. When descriptions are insufficiently discriminative, the planner misroutes queries, causing what we term *skill collision*: semantically overlapping skills compete for the same query population (Figure 1).

Skill collision is most acute during onboarding. Adding a new skill to a deployed agent forces its description to be precisely positioned relative to all incumbent skills. In practice, this requires iterative manual tuning: a developer writes a description, deploys the skill, observes routing errors from real or synthetic traffic, and edits accordingly. This loop is slow, requires expertise in both the skill's functionality and the planner's decision boundaries, and does not scale as agent capabilities grow.

We deploy an automated description optimization pipeline that replaces this manual loop. The pipeline initializes a candidate description with an LLM, then iteratively refines it using false-positive and false-negative cases from a labeled training set. On a production enterprise group chat agent (9 skills, 372 regression cases), the automated descriptions match manually tuned ones in routing quality (79.2% vs. 79.4% average F1, per-skill differences within the $\pm 0.78\%$ multi-seed noise floor) while reducing per-skill engineering effort from 120 minutes to 3.8 minutes ($32\times$ speedup).

Given that the automated pipeline reaches manual quality, we ask which of its components actually matter. As originally implemented, the pipeline combined several mechanisms motivated by intuition: contrastive feedback presenting FP, FN, and TP cases simultaneously; iterative refinement up to a fixed budget; dual editing of the most-confused rival skill; tuned training set sizes. Through systematic ablation on the production system and at scale on ToolBench (Qin et al., 2023) ($\sim$16k tools), spanning closed-world (per-query candidate pool) and open-world (retrieval over the full corpus) routing regimes, we find that most of these design choices change final routing F1 by less than 0.5%. A single LLM rewrite using any available FP/FN cases captures the bulk of

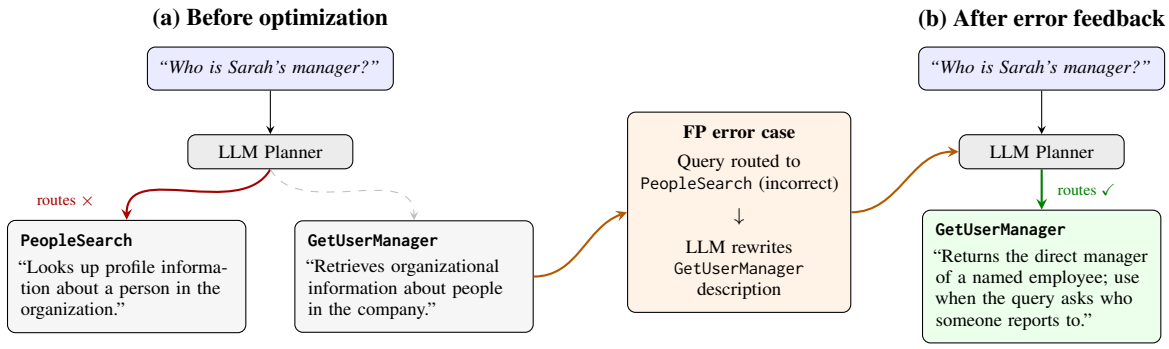

Figure 1: Skill collision and automated resolution via error feedback. **(a)** Before optimization: two skills share nearly identical descriptions; the LLM planner cannot distinguish them and misroutes the query to `PeopleSearch` (incorrect, ×). **(b)** After error-feedback-guided rewriting: the false-positive case is fed back to an LLM, which produces a discriminative description for `GetUserManager`; the planner now routes correctly (✓).

available improvement: on production, single-shot achieves 79.2% F1 (per-skill -0.12% from human-written descriptions, matching iterative refinement within noise); on ToolBench open-world, single-shot achieves a +4.45% F1 gain, within 0.2% of iterative refinement.

Not all routing failures are description failures. Two further findings characterize where description optimization applies, and where it does not. First, closed-world (fixed candidate pool) and open-world (retrieval-based) routing require separate optimization: descriptions tuned in one regime transfer poorly to the other, and LLM initialization helps in one setting but hurts the other. Second, when two skills' intended scopes genuinely overlap (as opposed to merely sharing under-discriminative wording), no description rewrite can resolve the collision; these skills exhibit large training-validation F1 gaps regardless of optimization signal quality. We identify this gap as a diagnostic signal that flags skills for architectural intervention (scope restructuring, intent-specific routing rules) rather than continued description refinement.

Taken together, these findings yield a compact operational picture for skill onboarding: optimize with a single-shot LLM rewrite, match the optimization regime to the deployment regime, and use the train-validation F1 gap to triage which skills need architectural intervention instead of more description tuning. We make three contributions: (1) a deployed production system that replaces manual description tuning at enterprise scale, validated as non-inferior to manually tuned baselines with a 32× engineering speedup; (2) systematic ablation across production and ToolBench showing that pipeline complexity—feedback type, iteration budget, dual editing, training size—has negligible impact, and a single LLM rewrite suffices; (3) characterization of where description optimization applies, including a diagnostic signal for skills requiring architectural intervention and the separation of closed-world and open-world routing into distinct optimization regimes.

2 Related Work

Tool selection for LLM agents. LLM tool selection encompasses routing models and planners over large API collections (Qin et al., 2023; Shen et al., 2023; Li et al., 2023), generating accurate calls from documentation (Patil et al., 2023), and retrieving candidate tools for orchestrators (Liu et al., 2025; Jia and Li, 2025; Lumer et al., 2025). These upstream retrieval methods complement our work; we optimize the descriptions used within the established candidate pools.

Tool description optimization. Recent methods use LLMs and execution traces to rewrite tool documentation, aiming to improve downstream task completion (Yuan et al., 2024; Qu et al., 2025; Fang et al., 2025; Guo et al., 2026). Instead of targeting downstream execution success via execution traces, we target upstream routing decisions using objective false positive and negative cases to diagnose skill collisions. Furthermore, we rigorously evaluate against manually tuned descriptions in a production deployment and demonstrate that a single LLM rewrite captures the vast majority of available improvements.

Prompt optimization & self-refinement Numerous frameworks optimize overarching task prompts or refine outputs using textual gradients, demonstrations, or LLM self-criticism (Khattab et al.,

2023; Opsahl-Ong et al., 2024; Yang et al., 2024; Zhou et al., 2023; Yuksekgonul et al., 2025; Pryzant et al., 2023; Madaan et al., 2023; Shinn et al., 2023; Agrawal et al., 2026). Instead of optimizing aggregate task prompts, we focus exclusively on per-skill descriptions evaluated via discrete routing errors. Furthermore, while many of these methods rely on multi-step evolutionary search or iterative closed-loop refinement, we demonstrate that a single-shot rewrite using objective routing feedback captures the vast majority of available improvements for tool description optimization.

3 Method

The skill onboarding pipeline takes as input a skill name and outputs an optimized routing description for use by the LLM planner. The pipeline has two stages: LLM initialization and error-feedback refinement. We additionally describe a single-shot variant which section 5.2 shows is sufficient to match the full iterative pipeline.

Stage 1: LLM initialization. We prompt an LLM with the skill name to generate a candidate routing description. The initialization prompt requests a description of what the skill does. The initialized description serves as both the starting point for refinement and a standalone baseline for evaluation.

Stage 2: Error-feedback refinement. The initialized description is evaluated on a labeled training set of queries with ground-truth skill labels. We collect false positives (FP: queries routed to the skill that should not be), false negatives (FN: queries the skill misses), and true positives (TP: correctly routed queries). At each iteration, we prompt the LLM with the current description, up to 5 FP and 5 FN cases (a practical token-budget choice; §5.2 confirms performance is insensitive to this limit), and a TP set matched to the number of negative cases (to balance the prompt context across positive and negative examples). The LLM is asked to identify routing failure patterns and revise the description. The refined description replaces the current one, and the process repeats up to a fixed iteration budget or until per-skill training F1 exceeds a 90% threshold (in practice, the iteration budget is the effective stopping criterion; see §5.2). At each iteration t, the routing evaluation produces error cases $\mathcal{E}_t = (\text{FP}_t, \text{FN}_t, \text{TP}_t)$; the description with the highest training F1 across iterations is selected. Algorithm 1 summarizes the loop.

Single-shot variant. We additionally evaluate a single-shot configuration where the LLM is given the initialized description and all available FP and FN cases in a single prompt, producing one revised description without further iteration. This simulates the approach a developer would take with a complete training set, and §5.2 shows it is sufficient to match iterative refinement.

Algorithm 1 Error-Feedback Refinement

Input: skill s, training queries Q, budget T, threshold τ
Output: optimized description $\hat{d}$

$d_0 \leftarrow \textsc{Initialize}(s)$ ▷ *Stage 1*
$\hat{d} \leftarrow d_0$, $\hat{f} \leftarrow 0$
for $t = 1, \ldots, T$:
  $f_t, \mathcal{E}_t \leftarrow \textsc{Evaluate}(d_{t-1}, Q)$
  if $f_t > \hat{f}$: $\hat{d} \leftarrow d_{t-1}$, $\hat{f} \leftarrow f_t$
  if $f_t \geq \tau$: **break**
  $d_t \leftarrow \text{LLM}\textsc{Rewrite}(d_{t-1}, \textsc{Sample}(\mathcal{E}_t))$
return $\hat{d}$

Open-world adaptation. In the retrieval setting, all tools are indexed by description and candidates are retrieved per query via hybrid sparse-dense retrieval (BM25 (Robertson and Zaragoza, 2009) combined with `text-embedding-ada-002` (OpenAI, 2022) cosine similarity, with per-query min-max normalization). The refinement loop runs identically, but FP cases now correspond to tools retrieved into the top-20 candidate pool that should not be invoked, providing a targeted disambiguation signal that reflects retrieval errors. Only tools that appear in at least one FP query's retrieved pool are eligible for refinement.

4 Experimental Setup

Production setting. The production system is an enterprise group chat agent whose LLM-based planner routes queries to one of 9 skills spanning people search, web search, calendar scheduling, internal knowledge retrieval, organizational hierarchy lookup, email, and document generation. We use 372 synthetic test cases with ground-truth skill labels, created by product and engineering experts. Each query may target one or more skills; all targeted skills are labeled as positives. Per-skill training queries are synthesized separately from the 372-case test set; positive examples range from 10 to 119 per skill depending on skill scope, with negative examples sampled from queries targeting other skills to ensure label balance. We compare four conditions: **HUMAN** (currently deployed manually tuned descriptions, our primary comparison baseline), **OLD** (skill disabled, system-level pre-deployment baseline), **INIT** (LLM-initialized de-

Table 1: Per-skill skill-selection F1 on the production agent. Each row corresponds to replacing a single skill's description with an automatically generated variant, while keeping all other skill descriptions fixed at their human-written versions. HUMAN: all skills use the currently deployed, manually tuned descriptions (hence identical F1 across rows); INIT: the target skill's description is LLM-initialized with no error feedback; SS: single-shot LLM rewrite given all training FP/FN cases; Iter: iterative refinement loop (max 10 iterations). Average Δ(HUMAN$\rightarrow$SS) of -0.12% and Δ(HUMAN$\rightarrow$Iter) of -0.20% are both well within the multi-seed per-skill noise floor of $\pm 0.78\%$ (Appendix E). Skills sorted by Δ(HUMAN$\rightarrow$Iter) in descending order.

Skill	HUMAN	INIT	SS	Iter	Δ(H$\rightarrow$SS)	Δ(H$\rightarrow$Iter)
IntKnowledge	79.4	77.1	80.1	81.0	$+0.7\uparrow$	$+1.6\uparrow$
PeopleSearch	79.4	79.2	78.6	79.9	$-0.8\downarrow$	$+0.6\uparrow$
Email	79.4	78.6	80.4	79.8	$+1.1\uparrow$	$+0.4\uparrow$
UserManager	79.4	79.4	78.8	79.7	$-0.6\downarrow$	$+0.4\uparrow$
WorkbackPlan	79.4	78.3	80.8	79.7	$+1.5\uparrow$	$+0.3\uparrow$
DirectReports	79.4	79.5	79.1	78.6	$-0.2\downarrow$	$-0.7\downarrow$
WebSearch	79.4	78.4	81.0	78.1	$+1.6\uparrow$	$-1.3\downarrow$
Calendar	79.4	77.6	76.9	77.9	$-2.5\downarrow$	$-1.5\downarrow$
StatusReport	79.4	77.6	77.5	77.7	$-1.9\downarrow$	$-1.7\downarrow$
Average	**79.4**	78.4	79.2	79.2	$\mathbf{-0.12}$	$\mathbf{-0.20}$

scription, no error feedback), and **Iter** (optimized description after refinement). Training set is the full set of available examples per skill (referred to as **trainmax**); a smaller **train20** variant is reported in Appendix C.

ToolBench setting. ToolBench (Qin et al., 2023) covers $\sim$16k RESTful API tools across I1, I2, I3 splits. I2 is the primary split for skill collision evaluation, because it requests multiple tools from the same API category to fulfill the query. ToolBench descriptions are developer-facing API stubs (e.g., *"Search Book by its name", "This endpoint will return back all news about Climate Change from all over the world"*) rather than routing instructions, which makes description quality directly measurable but also inflates the absolute gain numbers relative to systems with already-curated descriptions. We frame the task as single-turn tool routing: given a query and a candidate pool, predict the correct tool, with no API execution. We treat the first non-terminal tool call from the official ToolBench trajectories as ground truth; the original trajectories are LLM-generated and we did not manually verify each label. We study two candidate pool regimes. **Closed-world** uses the per-query predefined `available_tools` list (3 to 15 tools) from the ToolBench dataset as the full candidate pool. **Open-world** retrieves the top-20 candidates per query from the full $\sim$16k corpus via hybrid retrieval. For each split, we select the top-100 tools by positive sample count (requiring at least 25 positive and 25 negative queries); these subsets are fixed across all ablations. Per tool, we allocate 20 positive and 20 negative queries as the training set and hold out a balanced validation set of up to 50 positive and 50 negative queries, identical across all training-size ablation runs so that comparisons reflect only the change in training signal. We track three description states: **ORIG** (iteration 0, original placeholder), **INIT** (iteration 1, LLM-generated with no error feedback), and **Iter** (best description across up to 10 refinement iterations).

Ablation dimensions. On ToolBench we ablate: error signal type (FP+FN+TP / FP-only / FN-only / FP+FN without TP), iteration budget ($\{5, 10, 30\}$), per-iteration sampling (5 cases per category vs. all available), training size ($\{5, 10, 20\}$ examples per class), dual editing (simultaneously refining the rival tool's description), and three BM25 to dense retrieval weight combinations $(0.2, 0.8)$, $(0.5, 0.5)$, $(0.8, 0.2)$. Default settings unless specified are train=20, max iterations=10, BM25=0.2, EMB=0.8.

5 Results

5.1 Automated optimization matches manual tuning at production scale

Table 1 reports per-skill HUMAN, INIT, single-shot (SS), and iterative (Iter) F1 on the production agent. The average Δ(HUMAN$\rightarrow$Iter) across the 9 skills is -0.20% and Δ(HUMAN$\rightarrow$SS) is -0.12%. To characterize run-to-run variance of the live regression API, we ran 3 independent seeds of the trainmax pipeline; the average per-skill standard deviation of ΔF1 is $\pm 0.78\%$, with 8 of 9 skills below $\pm 1\%$ std (Appendix E). Both averages are well below this per-skill noise floor; the largest

Table 2: Five design choices originally motivating the pipeline complexity, evaluated on ToolBench I2 open-world (BM25=0.2, EMB=0.8, max-10 iterations, train=20 unless ablated). Δ is the gain from orig to best across variants of each design choice. n is the tools eligible for optimization (those with at least one retrieval false positive in the candidate pool); n varies across rows because eligibility depends on the configuration. Detailed per-variant numbers in Appendix A.

Design choice	Variants	Δ range	n
Feedback signal	FP+FN+TP / FP-only / FN-only / no TP	4.51 - 4.60	48
Iteration vs. Single-Shot	SS / max-10 sampled / max-10 full	4.45 - 4.85	47
Dual editing	on / off	4.51 - 4.54	48
Sampling	5 cases / all cases	4.51 - 4.85	47
Training size	5 / 10 / 20 per tool	4.51 - 4.92	31–48

per-skill differences are $\pm 1.7\%$ for Iter and $\pm 2.5\%$ for SS (Table 1). Per-skill outcomes split nearly evenly under both methods: 5 skills favor Iter and 4 favor manual tuning; SS splits 4 vs. 5. We interpret this as both single-shot and iterative automated optimization being non-inferior to manual tuning at production scale, but not exceeding it.

The value of automated onboarding is therefore operational. An applied scientist manually tuned descriptions for one skill using the standard process (write, regression test, observe failures, edit), recorded wall-clock time, and compared against the automated pipeline. Manual tuning required 120 minutes for this skill, whereas the automated pipeline produced comparable descriptions in 3.8 minutes, which is a 32× speedup.

Notably, the single-shot variant achieves this match without iteration or any of the additional pipeline mechanisms. The next section examines which components of the pipeline actually contribute to this result.

5.2 A single LLM rewrite captures most of the improvement

Table 2 summarizes ablations of five design choices originally motivating the elaborate pipeline: feedback signal composition, iteration budget, dual editing of confused tool pairs, per-iteration case sampling limit, and training set size. All five vary final F1 within 0.5% on ToolBench I2 open-world. Specifically: feedback signal type (FP+FN+TP vs. FP-only vs. FN-only vs. FP+FN without TP) yields gains within 0.1% of each other; dual editing provides negligible benefit; sampling 5 cases per category per iteration achieves similar gain as passing all available cases; training size effects (5 vs. 10 vs. 20 examples per tool) are within 0.5%, with train=10 marginally best. The implication is that the per-example error feedback signal is the dominant driver of optimization quality. How that signal is packaged into the LLM rewrite prompt is has less effect. Detailed per-variant numbers are in Appendix A.

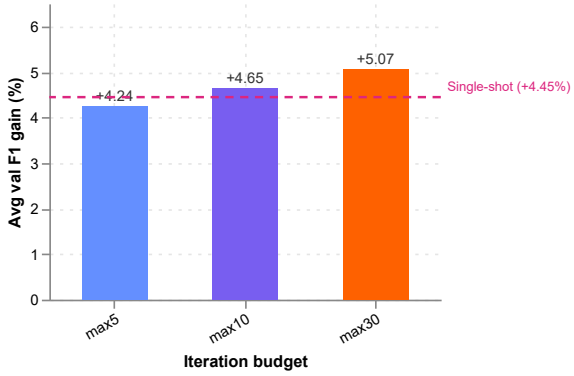

Figure 2: Average F1 gain on validation across different max iteration budget on ToolBench I2 open-world (BM25=0.2, EMB=0.8). Single-shot rewriting matches max-10 iterative refinement within 0.2%.

Figure 2 shows the iteration budget sweep. On ToolBench open-world, gains increase monotonically with iteration budget: $+4.24\%$ at max-5, $+4.65\%$ at max-10, $+5.07\%$ at max-30. Approximately 90% of tools hit the iteration cap without reaching the 90% stopping criterion at max-10, indicating optimization is still progressing when truncated. However, single-shot rewriting (one LLM call given all available training FP and FN cases) achieves $+4.45\%$, matching max-10 within 0.2% and exceeding max-5. Iterative refinement with sampled feedback adds little when single-shot has access to the full training signal. The production pattern reported in Section 5.1 confirms this finding at production skill.

For practitioners trading compute against performance, single-shot is the dominant choice across both settings. Long iteration budgets capture marginal additional gains in open-world ToolBench but the additional cost is rarely justified.

5.3 Optimization regimes diverge between retrieval-based and fixed-pool routing

Closed-world baseline F1 is high (∼91% on I2) and gains from optimization are correspondingly modest (0.6-0.9% across I1, I2, I3, Appendix H). Open-world baseline F1 is lower (∼70% on I2) and gains are substantially larger (+4.5%, Appendix B). More striking than the magnitude difference is a directional one: LLM-initialized descriptions *hurt* closed-world F1 by 0.8-1.4% comparing to the original placeholder text, while *improving* open-world F1 by 1.5-2.1%.

The two regimes require separate optimization. We re-evaluated closed-world-optimized descriptions in the open-world setting without further refinement: gains are modest (0.4-1.0%), well below in-setting open-world optimization (3.7-4.5%). Descriptions tuned against a fixed candidate pool do not generalize to the dynamic retrieved pool: closed-world optimization improves routing precision among a small known set of competitors, while open-world optimization additionally improves retrieval recall.

Description optimization is qualitatively different in retrieval-based vs. fixed-pool routing. Practitioners should determine which regime their deployment matches and optimize within that regime: LLM initialization is a useful starting point for retrieval-based routing but should be avoided when the candidate pool is fixed and small.

5.4 Initial training F1 as a diagnostic signal

Figure 3 examines whether iter-0 training F1 (the routing F1 of the original placeholder description on the training set, before any optimization) predicts the value of optimization on held-out data. On ToolBench open-world, tools with iter-0 training F1 below 65% ($n = 33$) achieve an average held-out gain of +6.27%, approximately 10 times the gain of tools with iter-0 training F1 at or above 65% ($n = 15$, +0.63%; t-test $p < 0.001$). The pattern is robust to threshold choice (Appendix G), and all tools with validation gains above 10% fall in the low iter-0 region. Iter-0 training F1 is a useful prioritization signal: tools below 65% are the candidates where optimization adds substantial value.

On ToolBench, most training improvements transfer to held-out performance. Per-tool training and validation F1 gains on ToolBench are positively correlated (Spearman $\rho = 0.66$); only 2 of 48 tools (4%) exhibit an overfitting signature where training gains substantially exceed validation gains (Appendix I).

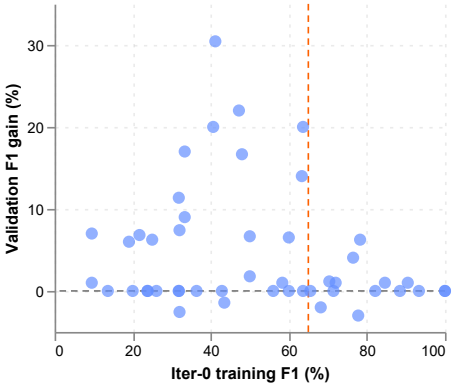

Figure 3: Each dot is one ToolBench I2 tool ($n = 48$). Tools with iter-0 training F1 below 65% (vertical dashed line) achieve substantially higher validation F1 gains.

A subset of skills cannot be resolved by description optimization alone. On the production agent, two skills (`IntKnowledge` and `WorkbackPlan`) follow this pattern: low iter-0 training F1, large training gains, but failure to exceed the HUMAN baseline on the held-out test (Table 1; multi-seed confirmation in Appendix E). Both skills carry inherent semantic overlap with other skills in the deployment: `IntKnowledge` retrieves person-related organizational data, overlapping `PeopleSearch`; `WorkbackPlan` produces structured plans, overlapping `StatusReport` in surface form. Description text alone cannot disambiguate skills whose intended scopes overlap, regardless of training signal quality. Practitioners encountering a skill with low iter-0 training F1 and a large Δtrain versus Δval gap should consider architectural intervention (e.g., intent-specific routing rules, restructured skill scopes) rather than description refinement.

6 Conclusion

We deployed an automated description optimization pipeline on a production enterprise group chat agent and validated findings at scale on ToolBench. Systematic ablation showed most of the originally designed mechanisms are unnecessary; a single-pass LLM rewrite given any FP/FN cases captures the bulk of available improvement and is non-inferior to manually tuned descriptions at production scale.

Limitations

We acknowledge several limitations. Our production system has 9 skills and 372 synthetic test cases, limiting statistical power for per-skill claims. The ToolBench original descriptions are placeholder text, which inflates the absolute gain numbers compared to systems with already-curated descriptions; the system's value in such settings remains to be characterized. Open-world ablations are conducted on I2 only; I1 and I3 open-world experiments are left for future work. The semantic overfitting observation in production is based on 2 of 9 skills; while ToolBench data falsifies the original "low iter-0 predicts failure" hypothesis at scale, the production-specific pattern in low-data settings warrants validation on additional production deployments.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Wei Fang, Yang Zhang, Kaizhi Qian, James Glass, and Yada Zhu. 2025. Play2prompt: Zero-shot tool instruction optimization for llm agents via tool play. *Preprint*, arXiv:2503.14432.

Ruocheng Guo, Kaiwen Dong, Xiang Gao, and Kamalika Das. 2026. Learning to rewrite tool descriptions for reliable llm-agent tool use. *Preprint*, arXiv:2602.20426.

Jingyi Jia and Qinbin Li. 2025. Autotool: Efficient tool selection for large language model agents. *Preprint*, arXiv:2511.14650.

O. 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. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *ArXiv*, abs/2310.03714.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A comprehensive benchmark for tool-augmented LLMs. *Preprint*, arXiv:2304.08244.

Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Syed Fahad Allam Shah, and Dan Roth. 2025. Toolscope: Enhancing llm agent tool use through tool merging and context-aware filtering. *Preprint*, arXiv:2510.20036.

Elias Lumer, Faheem Nizar, Anmol Gulati, Pradeep Honaganahalli Basavaraju, and Vamse Kumar Subbiah. 2025. Tool-to-agent retrieval: Bridging tools and agents for scalable llm multi-agent systems. *Preprint*, arXiv:2511.01854.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shrimai Prabhumoye, Bodhisattwa Prasad Majumder, Ximing Lu, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*.

OpenAI. 2022. New and improved embedding model. `https://openai.com/blog/new-and-improved-embedding-model`.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. *Preprint*, arXiv:2406.11695.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive APIs. *Preprint*, arXiv:2305.15334.

Reid Pryzant, Dan Iter, Jerry Li, Yin 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*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Yujia Qin, Shihao Liang, Yining Ye, Kunliang Zhang, Yankai Lin, Xin Sun, Qingyang Li, Zikun Liu, Qing Sun, Qingyan Zeng, Qinglin Wei, Shengding Hu, Zhiyuan Liu, Shengqi Han, Weize Chen, Jing Yi, Weilin Zhao, Tao Gui, Zheng Zhang, and 1 others. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *Preprint*, arXiv:2307.16789.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. From exploration to mastery: Enabling llms to master tools via self-driven interactions. In *The Thirteenth International Conference on Learning Representations*.

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3:333–389.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. *Preprint*, arXiv:2303.17580.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *Preprint*, arXiv:2401.06201.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616.

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 *The Eleventh International Conference on Learning Representations*.

A Ablation Tables

We ablate on the error signal type, iterative refinement, dual editing, and training size in Tables 3, 4, 5, 6, 7, and 8.

Table 3: Error signal type ablation on I2 open-world (48-tool intersection, BM25=0.2/EMB=0.8, train_size=20, max10). All conditions within 0.09%.

Feedback signal	Orig	Best	$\Delta_{0\to F}$
FP + FN + TP (full)	70.6	75.1	+4.5↑
FP-only (TP + FP)	70.8	75.3	+4.5↑
FN-only (TP + FN)	70.8	75.3	+4.5↑
FP + FN (no TP)	70.8	75.4	+4.6↑

B Open-World Detail (ToolBench)

Table 9 reports the full open-world results referenced in §5.3. **Transfer** re-evaluates closed-world-optimized descriptions in the open-world retrieval setting without further refinement. **In-setting** runs the full optimization loop using the retrieved candidate pool. Embedding-heavy retrieval (BM25=0.2, EMB=0.8) consistently outperforms BM25-heavy across both transfer and in-setting; BM25 fails to differentiate tools with generic placeholder names. n is the tools with at least one retrieval false positive; transfer evaluates all 100 tools (no FP requirement), while in-setting requires at least one FP per tool to be optimizable.

Table 4: Error signal type ablation on the production agent (9 skills, trainmax). All conditions within 0.69%, inside production API non-determinism noise. Δ = avg. New F1 − avg. OLD F1 across all 9 skills.

Feedback signal	Avg. Δ
FP + FN + TP (full)	+3.65↑
FP-only (TP + FP)	+2.99↑
FN-only (TP + FN)	+3.47↑
FP + FN (no TP)	+2.96↑

Table 5: Single-shot vs. iterative refinement on I2 open-world (47-tool overlap, BM25=0.2/EMB=0.8, train_size=20).

Method	n	Orig	$\Delta_{0\to F}$
Single-shot	47	71.0	+4.4↑
Iterative, sampled (max10)	47	70.7	+4.6↑
Iterative, full-context (max10)	47	70.9	+4.9↑

C Production train20 Results

Trainmax (used in the main paper, §5.1) uses all available positive examples per skill (10 to 119, varying by skill). For comparison, train20 caps the training set at 20 positive plus 20 negative examples per skill (or fewer when the skill has fewer positives available; `SendEmail`, `generate_status_report`, `WorkbackPlan` fall back to all available data). Table 10 reports per-skill F1 for the train20-standard run. On average, train20 yields Δ(OLD→Iter) = +1.71% versus trainmax-standard's +3.65%, indicating that more training data helps in aggregate but not uniformly: 5 skills are within 1% of their trainmax result, while `GetUserManager` regresses sharply at train20 (an outlier we attribute to API non-determinism, since the multi-seed trainmax results show `GetUserManager` is otherwise stable; Appendix E).

D OLD Baseline (Production Agent)

OLD measures the system-level F1 with the new skill disabled, evaluated on the subset of test cases that do not target the skill. Because OLD uses a different test subset than HUMAN/INIT/Iter (which

Table 6: Dual editing ablation on ToolBench I2 open-world (BM25=0.2/EMB=0.8, train_size=20, max10). n differs because dual editing fails on one tool whose training cases trigger the Azure content filter. Difference of 0.03% is well within experimental noise.

Setting	n	Orig	Best	$\Delta_{0\to F}$
Standard	48	70.6	75.1	$+4.51\uparrow$
Dual editing	47	70.3	74.9	$+4.54\uparrow$

Table 7: Dual editing ablation on the production agent (trainmax). Δ = Iter F1 $-$ Avg OLD baseline (averaged across 4 settings, 75.6% overall). Average effect of dual editing is -0.11%, well within the per-skill $\pm 0.78\%$ noise floor.

Skill	Std Δ	Dual Δ
IntKnowledge	$-1.7\downarrow$	$-3.5\downarrow$
PeopleSearch	$+13.3\uparrow$	$+12.5\uparrow$
Email	$+1.0\uparrow$	$-0.3\downarrow$
UserManager	$+7.5\uparrow$	$+6.9\uparrow$
WorkbackPlan	$-0.1\downarrow$	$+0.6\uparrow$
DirectReports	$+3.2\uparrow$	$+3.8\uparrow$
WebSearch	$+5.7\uparrow$	$+5.8\uparrow$
Calendar	$+2.7\uparrow$	$+3.6\uparrow$
StatusReport	$+0.9$	$+2.0\uparrow$
Average	$\mathbf{+3.6}\uparrow$	$\mathbf{+3.5}\uparrow$

evaluate on the full 372 cases with the skill enabled), OLD→Iter gains conflate skill availability with description quality and are not directly comparable to HUMAN→Iter. We report OLD here for completeness; §5.1 uses HUMAN→Iter as the primary comparison.

E Multi-Seed Variance (Production Agent)

To characterize run-to-run variance, we ran 3 independent seeds of the trainmax-standard pipeline. Each seed includes a fresh optimization run (LLM calls at temperature > 0) and a fresh regression evaluation (live API calls at temperature > 0). The HUMAN baseline (79.4%) was evaluated in a single regression run; we did not characterize HUMAN-specific run variance. The reported per-skill noise floor of ΔF1 ($\pm 0.78\%$ std on average; Table 12) bounds Iter variance directly; Δ(HUMAN→Iter) variance includes additional HUMAN evaluation noise we did not measure. The mean Δ(OLD→Iter) is $+3.27 \pm 0.78\%$ across the 3 seeds, with 8 of 9 skills below $\pm 1\%$ std. `IntKnowledge` regresses against OLD in all 3 seeds (-1.90%, -3.72%, -2.04%), confirming this is a robust failure rather than a single-run artifact.

Table 8: Training size ablation on ToolBench I2 open-world (BM25=0.2/EMB=0.8, max10). n varies because tool eligibility (at least one retrieval false positive in the train set) increases with train size. Δ values within 0.5%.

Train size	n	Orig	Best	$\Delta_{0\to F}$
5	31	67.8	72.7	$+4.9\uparrow$
10	42	69.8	74.8	$+5.0\uparrow$
20	48	70.6	75.1	$+4.5\uparrow$

Table 9: Open-world results on ToolBench I2. Transfer re-evaluates closed-world descriptions; in-setting re-optimizes against the retrieved candidate pool.

Setting	BM25/EMB	n	Orig	Best	$\Delta_{0\to F}$
Transfer (re-evaluate closed-world descriptions)					
	0.2/0.8	100	70.9	71.8	$+1.0$
	0.5/0.5	100	71.8	72.6	$+0.7$
	0.8/0.2	100	70.0	70.5	$+0.4$
In-setting optimization (max10)					
	0.2/0.8	48	70.6	**75.1**	$\mathbf{+4.5}\uparrow$
	0.5/0.5	45	68.9	72.8	$+3.9\uparrow$
	0.8/0.2	43	70.0	73.7	$+3.7\uparrow$

fact.

F Production Training F1 Dynamics

Table 13 reports per-skill training-set F1 at iteration 0 (the initial LLM-generated description) and the best across iterations on the trainmax-standard production setting. Two skills exhibit large training F1 gains: `IntKnowledge` ($+30.8$pp) and `WorkbackPlan` ($+28.1$pp). Both also start with low iter-0 training F1 ($\leq 67\%$), and neither translates the training gain into a held-out F1 improvement above the HUMAN baseline (Table 1). The other 7 skills show no training F1 movement: each begins at or above the 90% stop criterion and the optimization loop terminates at iter-0. The pattern matches the small-data overfitting characterization in §5.4.

G Iter-0 Training F1 as a Failure Predictor (ToolBench)

We test whether the production observation in §5.4, that low iter-0 training F1 predicts failure, generalizes to the 48 ToolBench open-world tools (I2, BM25=0.2/EMB=0.8, train_size=20, max10). For each tool, we extract iter-0 training F1 (**train_f1@0**) and the held-out validation gain Δval from the existing optimization logs; no new LLM calls are required.

Table 10: train20-standard production results (per-skill F1; train pos/neg = 20/20, capped by availability). Δ = Iter − OLD.

Skill	OLD	INIT	Iter	Δ
PeopleSearch	66.7	78.4	78.5	+11.8↑
WebSearch	73.0	78.5	78.0	+5.0↑
Calendar	75.2	75.7	77.0	+1.8↑
IntKnowledge	82.6	78.7	79.1	−3.4↓
UserManager	71.5	78.9	66.7	−4.9↓
DirectReports	76.0	79.3	79.3	+3.2↑
Email	79.1	78.8	79.6	+0.6↑
StatusReport	77.1	77.8	78.1	+1.0↑
WorkbackPlan	80.4	79.0	79.3	−1.1↓
Average				**+1.7**↑

The 65% threshold derived from production is highly significant on ToolBench (Welch's t, $p = 0.0008$, $N = 48$): tools below 65% iter-0 training F1 gain +6.3% on validation on average, vs. +0.6% for tools above (Table 14). However, the direction is opposite to the production failure mode: at ToolBench scale, low iter-0 F1 predicts *larger* held-out gains, not failure to generalize. Spearman correlation between Δtrain and Δval is +0.66 ($p < 10^{-4}$): in the open-world retrieval setting, large training gains generalize. We interpret the production overfitting (`IntKnowledge`, `WorkbackPlan`) as a small-data-regime artifact of low-positive-sample skills (10–119 positives), not a general property of the pipeline.

H Closed-World Results

Table 15 shows validation F1 in the closed-world setting, where the candidate pool is the per-query `available_tools` list (3–15 tools) rather than the full retrieved corpus. A consistent pattern emerges across all splits: LLM initialization (**init**) *reduces* F1 by 0.8–1.4% relative to the original placeholder descriptions (**orig**), while iterative refinement recovers and surpasses the original. Despite being completely uninformative text, placeholder descriptions perform comparably to zero-shot LLM-generated ones in this setting; the gains come entirely from error feedback, not initialization. This contrasts with the open-world setting (§5.3) where initialization helps, suggesting description quality matters more when retrieval recall is the bottleneck.

On I2, the training size sweet spot is 10 examples per class (+1.0%); additional examples do not consistently improve the refinement signal. Iterative refinement (+0.9%) modestly outperforms single-shot (+0.6%), consistent with per-example signal being weaker in the fixed small candidate pool.

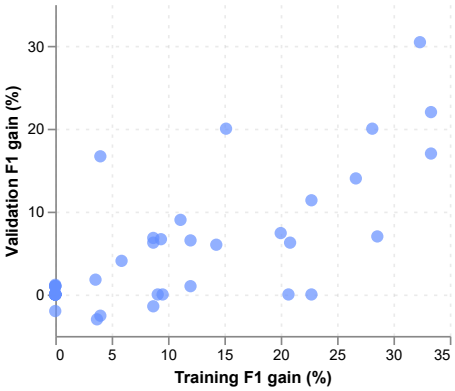

Figure 4: Per-tool Δtrain versus Δval on ToolBench I2 open-world ($n = 48$). Spearman $\rho = 0.66$. Two outliers in the lower-right exhibit overfitting.

I Training versus validation gain at ToolBench scale

Figure 4 plots per-tool training F1 gain against validation F1 gain for the 48 tools optimized in the open-world setting. Training and validation gains are positively correlated (Spearman $\rho = 0.66$); 2 tools (4%) exhibit an overfitting signature (Δtrain $> 20\%$, Δval $< 5\%$).

Table 11: Production results (trainmax, standard optimization) including the OLD baseline. OLD evaluates the system with the new skill disabled on a different test subset; it is not directly comparable to HUMAN/INIT/NEW. ΔDeploy = OLD→NEW (deployment + quality). Averaged OLD baseline across runs.

Skill	OLD	HUMAN	INIT	NEW	ΔDeploy
PeopleSearch	66.7	79.4	79.2	79.9	+13.3↑
WebSearch	72.4	79.4	78.4	78.1	+5.7↑
Calendar	75.3	79.4	77.6	77.9	+2.7↑
IntKnowledge	82.7	79.4	77.1	81.0	−1.7↓
UserManager	72.2	79.4	79.4	79.7	+7.5↑
DirectReports	75.4	79.4	79.5	78.6	+3.2↑
Email	78.8	79.4	78.6	79.8	+1.0↑
StatusReport	76.8	79.4	77.6	77.7	+0.9↑
WorkbackPlan	79.8	79.4	78.3	79.7	−0.1
Average		**79.4**	78.4	79.2	**+3.6**↑

Table 12: Per-skill ΔF1 (NEW − OLD) across 3 independent seeds of the trainmax-standard pipeline. Std is sample standard deviation across seeds. The average per-skill std is ±0.78%.

Skill	Seed 1	Seed 2	Seed 3	Mean	Std
PeopleSearch	+13.0	+10.7	+11.8	+11.8	±1.2
WebSearch	+5.7	+6.2	+5.7	+5.9	±0.3
Calendar	+2.7	+1.9	+2.8	+2.5	±0.5
IntKnowledge	−1.9	−3.7	−2.0	−2.6	±1.0
UserManager	+6.5	+7.1	+5.5	+6.4	±0.8
DirectReports	+4.2	+3.8	+2.8	+3.6	±0.8
Email	+0.6	+1.3	−0.2	+0.6	±0.8
StatusReport	+1.9	+0.9	+0.1	+1.0	±0.9
WorkbackPlan	+0.1	+1.2	−0.4	+0.3	±0.8
Average	**+3.6**	**+3.3**	**+2.9**	**+3.3**	**±0.8**

Table 13: Per-skill training-set F1 at iter-0 (initial LLM description) vs. best across iterations on the production agent (trainmax-standard). Δtrain is the iterative refinement gain on the training subset. Skills ordered as in Table 1.

Skill	iter-0	best	Δtrain
IntKnowledge	58.1	88.9	**+30.8**↑
PeopleSearch	97.0	97.0	+0.0
Email	90.3	90.3	+0.0
UserManager	96.2	96.2	+0.0
WorkbackPlan	66.7	94.7	**+28.1**↑
DirectReports	95.5	95.5	+0.0
WebSearch	92.6	92.6	+0.0
Calendar	90.0	90.0	+0.0
StatusReport	100.0	100.0	+0.0

Table 14: Threshold sweep: mean Δval for tools below vs. above each iter-0 training F1 threshold τ. The 65% threshold from production is highly significant ($p = 0.0008$). At ToolBench scale, low iter-0 F1 predicts *larger* validation gains, not failure.

τ	$n_{<\tau}$	mean Δval$_{<\tau}$	$n_{\geq\tau}$	mean Δval$_{\geq\tau}$	p
55%	26	+6.4	22	+2.3	0.052
60%	28	+5.9	20	+2.5	0.093
65%	**33**	**+6.3**	**15**	**+0.6**	**0.0008**
70%	35	+5.9	13	+0.9	0.002
75%	38	+5.5	10	+0.9	0.005
80%	41	+5.2	7	+0.3	0.0003

Table 15: Closed-world results. Top: I1/I2/I3 under standard optimization (train_size=20, max10). Bottom: I2 training size ablation and single-shot comparison. Val F1 averaged over 100 tools.

	n	Orig	Init	Best	$\Delta_{0\to1}$	$\Delta_{0\to F}$
I1	100	90.6	89.4	91.5	−1.2	+0.9↑
I2	100	90.6	89.2	91.5	−1.4	+0.9↑
I3	100	82.8	82.0	83.5	−0.8	+0.6↑
I2 training size ablation						
train=5	100	90.7	89.5	91.1	−1.2	+0.4↑
train=10	100	90.7	89.3	**91.7**	−1.4	**+1.0**↑
train=20	100	90.6	89.2	91.5	−1.4	+0.9↑
I2 single-shot vs. iterative (train=20, max10)						
Single-shot	100	90.6	—	91.3	—	+0.6↑
Iterative	100	90.6	89.2	91.5	−1.4	+0.9↑