# Learning from Delivery Traces: A Research Agenda for Continually Improving Data Platform Agents

*Literature review and conceptual analysis*


## Abstract

Data-platform agents operate in environments where schemas, APIs, organizational policies, and local practices change continuously. Static prompts and fixed tool descriptions therefore decay, while unconstrained learning from successful interactions risks reproducing obsolete shortcuts and unsafe habits. This research agenda examines delivery traces as a source of situated knowledge for continually improving analytics agents. SiriusBI supplies an upstream view in which multi-round clarification and data-conditioned generation adapt business questions to domain context. SiriusDeliver supplies a downstream view in which hierarchical skills, artifact lifecycle control, and trace-driven skill evolution support warehouse delivery at production scale. The agenda connects these systems with research on retrieval-augmented generation, reasoning and acting, learned tool use, production technical debt, readiness testing, data-quality verification, data cascades, documentation, human-AI interaction, and limits of intrinsic self-correction. It proposes a trace-to-skill pipeline with selection, abstraction, specification, testing, approval, deployment, monitoring, and retirement stages. It also defines an evaluation program for learning benefit, safety, drift, and organizational distribution. The central claim is that traces should be treated as governed evidence, not as demonstrations that automatically deserve imitation. Continual improvement becomes credible only when learned procedures remain scoped, testable, reversible, and attributable.


## 1. Why Static Agents Decay

A data warehouse is a moving target. Tables are renamed, owners change, engines add features, governance rules tighten, cost profiles shift, and teams develop new conventions. A procedure that worked last quarter may now select a deprecated source or violate a retention rule. A general language model can reason about a textual description of these changes, but descriptions themselves become stale. Continual adaptation is therefore not an optional enhancement; it is part of operating the system.

The obvious source of adaptation data is the trace. A delivery trace may contain the initiating request, context retrieval, selected tools, generated code, validation results, platform errors, human edits, retries, and the final outcome. It captures not just what artifact was delivered but how obstacles were resolved. Thousands of traces can reveal repeated procedures that deserve to become reusable skills.

The same record is treacherous. A successful trace may contain an unnecessary workaround, a credential that was too broad, an undocumented manual intervention, or a silent quality defect discovered weeks later. Historical frequency can reflect organizational habit rather than good practice. Trace learning must therefore distinguish evidence of behavior from evidence of correctness.

## 2. The Target Systems Define a Learning Continuum

SiriusBI addresses adaptation at the question-to-query boundary. Jiang et al. (2024) describe an industrial BI system that combines coordinated modules, multi-round dialogue, and a data-conditioned choice between one-step fine-tuning and a two-step semantic intermediate representation. Its dialogue mechanism includes semantic completion, knowledge-guided clarification, and proactive querying. The official report describes deployments across finance, advertising, and cloud contexts, more than 93 percent SQL-generation accuracy, and analyst query-time reductions from minutes to seconds.

The data-conditioned strategy can be read as a precursor to continual adaptation. It acknowledges that domains differ in available data and adaptation cost. A well-established domain may justify fine-tuning; a new domain may be better served by a structured intermediate route. But model adaptation alone does not keep metric catalogs, schemas, and policies current. The system must also retrieve changing context and preserve corrections made through interaction.

SiriusDeliver makes learning from operations explicit. Xie et al. (2026) describe a warehouse delivery agent with hierarchical skill orchestration, artifact lifecycle controls before and after execution, and a trace-driven skill evolution mechanism. The production report covers 18,240 sessions and 3,600 monthly active users over two months across six teams and four task types. It reports 87.2 percent end-to-end success, 73.5 percent autonomous submission, and an A/B test in which median delivery time fell from 228 to 23 minutes and engineer effort from 95 to 11 minutes while final success remained comparable.

These results establish a setting in which trace learning could have high leverage. Repeated platform operations create many opportunities to reuse procedural knowledge. They also increase the cost of a bad learned skill because the procedure can be invoked at scale. A research agenda must address both sides of that leverage.

## 3. What Is a Skill?

A skill should be more than a retrieved transcript or a long prompt. It is a versioned procedure with a declared purpose, input and output types, preconditions, allowed tools, side-effect class, invariants, expected observations, recovery branches, and an owner. It may contain language-model reasoning steps, deterministic code, or both. Its specification should be understandable without the original conversation.

This definition connects to tool-use research. ReAct interleaves reasoning with actions so observations can update subsequent decisions (Yao et al., 2023). Toolformer investigates how models can learn when to call APIs, which arguments to provide, and how to incorporate results (Schick et al., 2023). A warehouse skill needs these capabilities, but it must also restrict them. The important question is not only whether an agent can call a scheduler API. It is whether this skill may call it in the production environment, for which task type, under which identity, with which resource and retry limits.

Skills should express exception semantics. If a source partition is missing, the procedure might wait, choose an approved fallback, request a backfill, or escalate. Each branch carries assumptions and consequences. A trace may show one branch; the skill specification must state the boundary within which that branch generalizes.

Finally, a skill should return evidence. It does not merely say "delivery succeeded." It returns artifact identifiers, validation outcomes, execution status, policy decisions, and unresolved warnings. Evidence makes composition possible: a parent skill can authorize its next step based on typed outcomes rather than trusting generated narrative.

## 4. A Trace-to-Skill Pipeline

The proposed pipeline begins with trace selection. Candidate traces should be sampled from successes, failures, escalations, recoveries, and later incidents. Selection based only on immediate success creates survivorship bias. Traces must be linked to delayed outcomes such as data-quality alerts, rollbacks, and downstream complaints before they are labeled exemplary.

The second stage is sanitization and segmentation. Sensitive values, credentials, and personal data are removed or replaced with governed references. The trace is segmented into decisions, tool actions, observations, and human interventions. Hidden model reasoning is neither required nor a reliable training target. Observable events and concise decision rationales provide a stronger audit surface.

The third stage is abstraction. Concrete table names, dates, and team-specific settings are replaced with typed variables where generalization is justified. The system identifies the stable pattern: for example, "create a daily partitioned transformation with dependency checks and a canary validation." Abstraction should be conservative. Two superficially similar traces may encode different business or regulatory obligations.

The fourth stage is specification. Preconditions, postconditions, invariants, failure branches, permitted effects, and escalation rules are made explicit. Retrieval-augmented generation can supply current platform documentation and policy while the skill is synthesized (Lewis et al., 2020). The retrieved sources and versions become part of the candidate skill's provenance.

The fifth stage is testing. Unit tests evaluate deterministic components. Simulated tool interfaces test branches and side effects. Historical replay tests representative episodes. Adversarial cases probe missing metadata, schema drift, permission boundaries, partial failures, prompt injection, and conflicting instructions. Automated data-quality verification can supply executable postconditions over resulting datasets (Schelter et al., 2018).

The sixth stage is approval and staged deployment. Low-risk skills with strong tests may receive automated policy approval. High-impact skills require review by platform and data owners. Deployment begins in shadow or sandbox mode, then canary scope, then broader availability. Every invocation records the skill version and evidence returned.

The final stages are monitoring and retirement. Drift indicators include rising exception rates, increased human edits, changed APIs, failed invariants, and performance differences across teams. A skill should have expiry conditions and an owner. Retirement prevents the library from accumulating procedures that remain callable long after their assumptions disappear.

## 5. The Quality of the Learning Signal

Outcome labels are multi-dimensional. A platform submission can succeed technically while producing poor data. A user can accept an artifact because reviewing it is burdensome. A quick delivery can impose maintenance work on another team. Trace learning should therefore combine immediate execution, semantic review, data quality, human effort, downstream incidents, and longevity.

Data-cascade research is a warning against narrow labels. Sambasivan et al. (2021) show how undervalued data work and upstream problems can compound in high-stakes AI systems. A trace corpus assembled only from agent-visible events may omit the steward who repaired source data or the analyst who later reconciled a metric. The research agenda should study whose labor becomes visible in traces and whose is externalized.

Production-readiness research suggests a way to operationalize broader quality. Breck et al. (2017) translate reliability concerns into specific tests and monitoring needs. A skill-readiness score could require representative replay, failure-branch coverage, ownership, permissions review, rollback, observability, and postcondition checks. The score should gate deployment, not conceal a failed critical criterion through averaging.

Technical debt also belongs in the learning signal. Sculley et al. (2015) describe how data dependencies, glue code, and feedback loops generate hidden maintenance costs. A learned skill that depends on five undocumented conventions may save time today and create fragility tomorrow. Candidate evaluation should estimate dependency complexity, duplication, and future maintenance obligations.

## 6. Memory Layers and Their Governance

Continual agents need several memories. Episodic memory stores selected past runs. Semantic memory stores current schemas, policies, and metric definitions. Procedural memory stores tested skills. Outcome memory stores delayed incidents and evaluations. Mixing these layers produces errors: an old episode can be mistaken for current policy, and a successful procedure can be applied outside its scope.

Retrieval-augmented generation provides a general mechanism for accessing non-parametric memory that can be updated and inspected (Lewis et al., 2020). Enterprise retrieval should add access control, freshness, provenance, and conflict resolution. The most similar document is not necessarily authoritative. Retrieval ranking should consider ownership, validity intervals, environment, and approval status.

Documentation frameworks can improve memory objects. Datasheets call for explicit records of data motivation, composition, collection, use, and limitations (Gebru et al., 2021). An analogous skill sheet should state training traces, intended tasks, unsupported contexts, tests, known failures, risk class, maintainer, and expiry. This makes procedural memory governable rather than magical.

Memory updates should be reviewable diffs. If a platform error leads the agent to propose a new retry rule, reviewers should see which traces motivated it, what specification changed, which tests were added, and what deployment scope is requested. This is more defensible than silently appending the latest conversation to a vector index.

## 7. Human Correction as Structured Evidence

Human edits are valuable but ambiguous signals. An engineer may repair a model mistake, apply a personal preference, or use a temporary workaround. The system should ask for lightweight correction labels when impact justifies the interruption: semantic correction, policy correction, platform update, style preference, or exceptional workaround. These labels improve later abstraction.

Human-AI interaction guidance emphasizes supporting efficient correction, communicating system status, and preserving user control (Amershi et al., 2019). A trace-learning interface can meet these goals by showing the proposed lesson: "The system intends to learn that task type A requires dependency flag B in environment C." The engineer can approve, narrow, or reject the generalization without authoring a full skill.

Credit and responsibility also matter. Skills may encode substantial domain knowledge contributed by analysts and platform engineers. Provenance should identify contributors and owners, while governance should prevent a widely reused skill from becoming nobody's responsibility. Continual improvement is an organizational process mediated by an agent, not autonomous model self-development.

## 8. External Feedback, Not Ritual Self-Correction

A common agent pattern asks the model to critique and revise its own output. This can help expose alternatives, but it is not reliable evidence of correction. Huang et al. (2024) find that large language models struggle to improve reasoning through intrinsic self-correction without informative external feedback. In data engineering, a second answer generated from the same context may repeat the same semantic error with greater confidence.

Trace learning should prioritize feedback from parsers, compilers, query plans, policy engines, execution logs, data tests, differential queries, and authorized humans. Each signal covers a distinct failure class. Parser success says nothing about business meaning; data-quality success says little about access policy. The skill specification should declare which external signals support each postcondition.

SiriusDeliver's before-and-after artifact control is particularly relevant because it can acquire execution feedback during the episode (Xie et al., 2026). Future research should test which observations actually

improve repair, when the agent misdiagnoses a signal, and whether repeated repair loops converge or merely optimize for the nearest automated check.

## 9. Evaluation Program for Continual Improvement

The first research question is learning benefit: do trace-derived skills increase end-to-end success, reduce effort, or improve quality compared with retrieval of documentation and fixed expert-authored skills? A controlled study should compare these conditions on future, temporally separated tasks. Random splits are inadequate because they allow near-duplicate procedures from the same platform era into training and test sets.

The second question is generalization boundary. Performance should be stratified by team, task type, schema family, platform version, and policy regime. Researchers should measure beneficial transfer, harmless abstention, and harmful overgeneralization. A skill that works well only for its source team can still be valuable if its scope is explicit.

The third question is safety. Measures include unauthorized action attempts, escaped defects, rollback, sensitive-data exposure, unsafe retry, and misleading evidence. Near misses are important. A policy engine that blocks an unsafe learned action demonstrates containment, but frequent blocks indicate a poor skill or scope.

The fourth question is maintenance economics. Trace learning consumes review, testing, monitoring, and retirement labor. Studies should report net human effort over months, not only time saved per successful delivery. They should measure skill-library growth, duplication, stale-skill removal, and incident-response burden.

The fifth question is distribution. Does learning improve performance equally across teams, or does it primarily benefit groups whose work is already well represented in traces? Does the system route hard data labor to less visible roles? These questions require qualitative studies alongside telemetry.

## 10. Concrete Research Hypotheses

Several hypotheses follow. First, typed trace segmentation will produce safer transferable skills than end-to-end imitation of full transcripts. Second, delayed outcome linkage will reduce promotion of procedures associated with later incidents. Third, explicit expiry and ownership will reduce stale-skill invocations. Fourth, risk-tiered approval will preserve most efficiency gains while sharply reducing harmful actions. Fifth, cross-layer traces connecting SiriusBI-style intent clarification with SiriusDeliver-style deployment will reduce semantic drift between exploratory queries and production tables.

Another hypothesis concerns routing. SiriusBI selects generation strategies according to data conditions (Jiang et al., 2024). A broader router could choose among direct tool use, retrieval, a tested skill, a planning agent, or human escalation. Evaluating routing regret would reveal whether the system selects the right level of autonomy, not merely whether any one method performs well.

Finally, skill evolution may work best as proposal generation. The agent detects recurring patterns and drafts specifications and tests; deterministic tooling and accountable reviewers control promotion. This division uses language models for abstraction and synthesis while keeping institutional authority visible.

## Conclusion

Delivery traces are a rich record of how data work actually proceeds, including the exceptions that static manuals omit. SiriusBI shows how an analytics system can adapt interaction and generation to domain conditions. SiriusDeliver shows how a production delivery agent can coordinate skills, control artifacts across execution, and evolve procedures from traces. Together they motivate continual data-platform agents whose memory spans intent, operations, and outcomes.

The research challenge is not simply to learn faster. It is to decide what deserves to be learned, where it applies, how it is tested, who authorizes it, and when it should be forgotten. Traces become trustworthy learning material only when they are connected to delayed outcomes, sanitized, abstracted conservatively, converted into explicit skill contracts, and governed through deployment and retirement. That discipline can turn experience into institutional capability without turning yesterday's accidents into tomorrow's automation.

## References

Jiang, J., Xie, H., Yang, J., Shen, S., Wang, Z., Zheng, Y., ... & Jiang, J. (2024). Siriusbi: A comprehensive llm-powered solution for data analytics in business intelligence. *arXiv preprint arXiv:2411.06102*.

Xie, H., Zhou, X., Yang, J., Shen, S., Wang, Z., Zheng, Y., ... & Jiang, J. (2026). SiriusDeliver: Automating Data Warehouse Delivery at Tencent. *arXiv preprint arXiv:2608.09185*.

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., et al. (2019). Guidelines for human-AI interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (pp. 1-13). https://doi.org/10.1145/3290605.3300233

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data* (pp. 1123-1132). https://doi.org/10.1109/BigData.2017.8258038

Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daume III, H., & Crawford, K. (2021). Datasheets for datasets. *Communications of the ACM, 64*(12), 86-92. https://doi.org/10.1145/3458723

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2024). Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*. https://openreview.net/forum?id=IkmD3fKBPQ

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems, 33*.

Sambasivan, N., Kapania, S., Highfill, H., Akrong, D., Paritosh, P., & Aroyo, L. M. (2021). Everyone wants to do the model work, not the data work: Data cascades in high-stakes AI. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1-15). https://doi.org/10.1145/3411764.3445518

Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F., & Grafberger, A. (2018). Automating large-scale data quality verification. *Proceedings of the VLDB Endowment, 11*(12), 1781-1794. https://doi.org/10.14778/3229863.3229867

Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., et al. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems, 36*. https://doi.org/10.52202/075280-2997

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., et al. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems, 28*, 2503-2511.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X