

# xskill: Team-Level Skill Distillation, Sharing, and Evolution for Coding Agents

Anonymous Author(s)  
Anonymous Institution

## Abstract

Coding agents such as Claude Code, Codex, and OpenCode increasingly rely on reusable procedural knowledge—*skills*—to guide complex software engineering workflows. However, current approaches treat skill creation as either a manual authoring task or an isolated academic experiment, leaving a critical gap: no existing system simultaneously supports *automatic distillation* from real user trajectories, *team-level sharing* across an organization, and *data-driven evolution* through canary deployment and user-experience scoring. We present **xskill**, an open-source framework that closes this gap. xskill operates as a management layer above existing coding-agent runtimes, automatically collecting sanitized trajectories, decomposing them into atomic task segments, clustering segments into candidate skills, and evolving skills through a git-branching canary pipeline where staging branches compete against main on real UX scores. A cross-sectional survey of 10 trajectory-to-skill systems reveals that xskill is the first to combine (1) per-turn asynchronous trajectory collection across 5+ agent ecosystems, (2) multi-agent skill production with cumulative evidence thresholds, (3) canary-based A/B evaluation—a mechanism absent from all 10 surveyed systems, and (4) cross-agent skill distribution via the emerging SKILL.md standard. We describe the system architecture, the three-agent pipeline (TaskAgent, TaskClusterAgent, SkillEditAgent), and the canary evaluation protocol, and position xskill’s design decisions against the landscape of related work.

## 1 Introduction

The rise of agentic coding assistants—Claude Code [1], Codex [2], OpenCode [3], and others—has transformed software development workflows. These agents leverage *skills*: reusable, markdown-formatted procedural documents (typically named `SKILL.md`) that encode domain-specific knowledge, best practices, and step-by-step instructions for recurring tasks.

In team settings, the skill problem compounds. Developer A spends an afternoon teaching her agent a deployment workflow; Developer B, facing the same task, starts from scratch. The knowledge is trapped in individual session histories. Even when a workflow is manually documented as a skill, there is no mechanism to:

- **Automatically distill** successful agent trajectories into reusable skills;
- **Share and distribute** skills across team members with appropriate sanitization;
- **Evolve** skills based on real usage data, retiring those that degrade and promoting those that help.

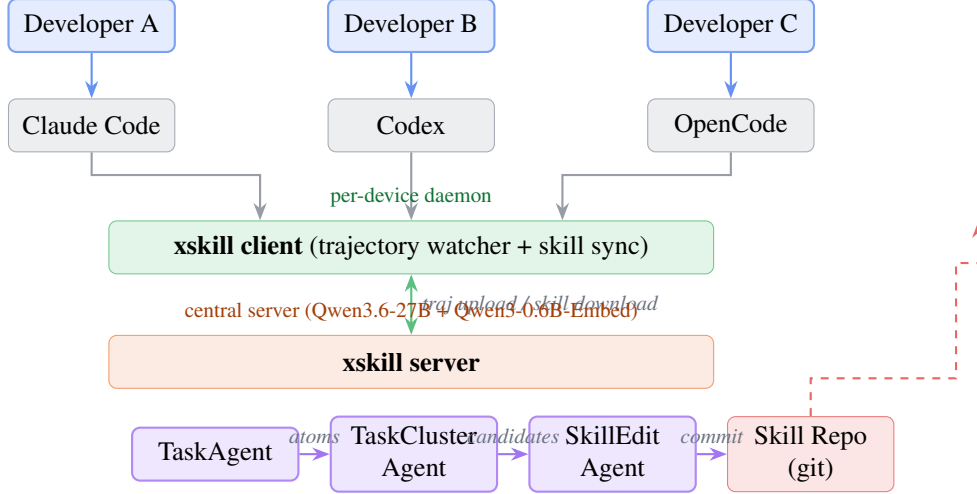


Figure 1: **xskill system overview.** Developers use their preferred coding agents (Claude Code, Codex, OpenCode, etc.). The xskill client daemon watches trajectory directories and uploads sanitized deltas to the central server. The server’s three-agent pipeline decomposes trajectories into atomic tasks, clusters them into skill candidates, and produces versioned SKILL.md artifacts via git commits. Evolved skills are synced back to each developer’s agent runtime.

We present **xskill**, an open-source system that addresses all three challenges. xskill operates as a *management layer* above existing coding-agent runtimes (Figure 1). It does not replace any agent’s skill loader; instead, it shapes what those loaders read from disk. The key insight is that the *lifecycle* of a skill—creation, evaluation, evolution, retirement—is orthogonal to the *runtime* that loads and injects skills into agent prompts, and should therefore be managed by a dedicated system.

Our contributions are:

1. A **cross-agent trajectory collection** architecture supporting 5 coding-agent ecosystems (Claude Code, Codex, OpenCode, OpenClaw, Hermes) through a unified normalizer interface.
2. A **three-agent pipeline** (TaskAgent → TaskClusterAgent → SkillEditAgent) for automatic skill distillation from raw trajectories, with cumulative evidence thresholds replacing single-trajectory heuristics.
3. A **canary evaluation protocol** using git branching (staging vs. main) and UX scoring—the first such mechanism in the trajectory-to-skill literature.
4. A **comprehensive survey** of 10 trajectory-to-skill systems across 5 design dimensions (trigger, storage, production, evaluation, usage), providing the first systematic comparison matrix for this emerging field.

## 2 Related Work

We surveyed 10 systems that convert agent trajectories or experiences into reusable procedural knowledge. Table 1 presents the landscape overview; here we discuss the most instructive comparisons.

Table 1: **Landscape of trajectory-to-skill systems.** Columns indicate whether the system produces SKILL.md files, supports online (per-turn) triggering, implements deduplication, performs canary/A-B evaluation, and targets production coding-agent deployment.

System	SKILL.md	Online	Dedup	Canary	Production
OpenSpace	✓	✓	✓	×	✓
EvoSkill	✓	×	×	×	✓
AutoSkill	✓	✓	✓	×	✓
Hermes	✓	×	×	×	×
GEPA-gskill	✓	×	×	×	×
Trace2Skill	×	×	×	×	×
AgentEvolver	×	×	×	×	×
MemSkill	×	×	×	×	×
EvoAgentX	×	×	×	×	×
SE-Agent	×	×	×	×	×
SkillRL	×	×	×	×	×
<b>xskill</b>	✓	✓	✓	✓	✓

## 2.1 Skill-as-SKILL.md Systems

**OpenSpace** [4] is the closest system to xskill in scope. It maintains an evolving skill library via an MCP server, with three evolution triggers (per-task analysis, tool-degradation events, and periodic counters) feeding into a  $3 \times 3$  evolution type matrix (FIX / DERIVED / CAPTURED). However, OpenSpace wraps skill execution inside its own sub-agent—meaning the host agent’s harness (e.g., CLAUDE.md rules) does not apply during skill-enhanced execution, a fundamental isolation problem for production use.

**EvoSkill** [5] uses git branches as program versions and Pareto frontiers for selection, offering the cleanest version-control story among all surveyed systems. However, it requires offline batch execution and does not support real-time trajectory collection.

**AutoSkill** [6] is the only system besides xskill that achieves per-turn asynchronous skill extraction during live chat. Its three-layer deduplication sandwich (identity hash  $\rightarrow$  semantic+BM25  $\rightarrow$  LLM judge) is the most sophisticated deduplication mechanism we found. However, AutoSkill operates as a standalone framework rather than integrating with existing agent ecosystems.

## 2.2 Non-SKILL.md Systems

Several systems use alternative representations: **MemSkill** [9] uses RL-trained PPO controllers over operation banks; **EvoAgentX** [10] evolves workflow code; **SE-Agent** [11] replaces per-instance system prompts; **SkillRL** [12] trains RL agents with JSON skill records; and **AgentEvolver** [8] delegates to an external ReMe memory service. These systems target RL training loops rather than production coding-agent deployments and are thus complementary to xskill’s scope.

**Trace2Skill** [7] proposes a MapReduce-style patch merging approach and demonstrates that machine-generated skills can outperform human-written ones (on SpreadsheetBench). Its key finding—cross-model skill transfer is viable—supports xskill’s design of centralized skill production that serves heterogeneous client runtimes.

**GEPA** [13] introduces system-aware three-way merge by common ancestor, analogous to git’s merge algorithm applied to prompt evolution. This is the most principled merge mechanism we found, and xskill’s git-based versioning enables similar ancestral reasoning.

System	Async Trig.	Multi-eco	Dedup	Canary	UX Score
OpenSpace	✓	×	✓	×	×
AutoSkill	✓	×	✓	×	×
EvoSkill	×	×	×	×	×
Trace2Skill	×	×	×	×	×
GEPA	×	×	×	×	×
MemSkill	×	×	×	×	×
<b>xskill</b>	✓	✓	✓	✓	✓

Figure 2: **Feature coverage of trajectory-to-skill systems** on five dimensions critical for team deployment. Only xskill covers all five: asynchronous per-turn triggering, multi-ecosystem support, deduplication, canary-based A/B testing, and user-experience scoring. None of the 10 surveyed systems implement canary or UX evaluation.

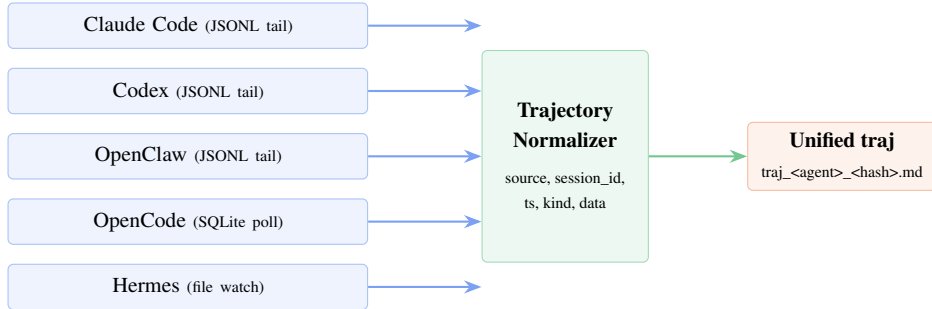


Figure 3: **Cross-agent trajectory collection.** Each coding agent writes trajectories in its native format and path. The xskill client runs per-agent adapters that normalize events into a unified schema. Four of five agents use appendable JSONL files, enabling efficient `inotify/tail` collection; OpenCode’s SQLite database requires polling.

### 2.3 Key Gap: No System Does A/B Evaluation

Across all 10 surveyed systems, *none* implements canary/A/B evaluation (Figure 2). Most evaluate skills via offline benchmarks (EvoSkill, GEPA, MemSkill) or LLM-as-judge on the skill text itself (OpenSpace). Only AutoSkill tracks per-turn `relevant/used` judgments via an LLM judge, but this measures skill selection quality, not downstream user experience. The absence of A/B testing means that skill evolution in all existing systems is driven by proxy metrics rather than direct evidence of user benefit—a gap xskill explicitly addresses.

## 3 System Architecture

xskill consists of two components: a **client daemon** running on each developer’s machine, and a **central server** that hosts the three-agent pipeline and the skill repository.

### 3.1 Cross-Agent Trajectory Collection

A key design decision is that xskill does *not* require agents to push trajectories; instead, the client daemon *pulls* from known filesystem locations (Figure 3). This zero-configuration

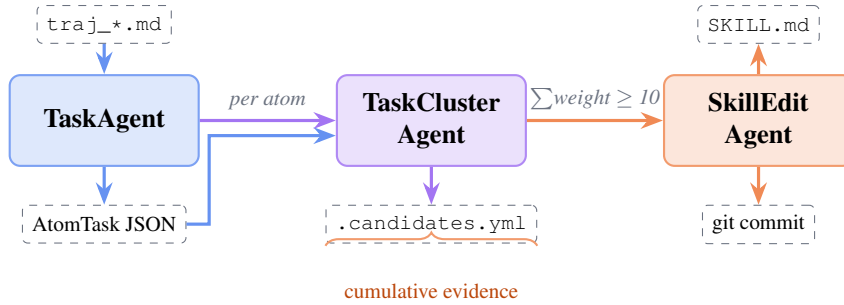


Figure 4: **Three-agent pipeline.** *TaskAgent* decomposes trajectory deltas into AtomTask segments. *TaskClusterAgent* routes each atom to an existing or new skill, appending weighted evidence. *SkillEditAgent* fires when cumulative evidence exceeds a threshold (default:  $\text{weight} \geq 10$ ), producing or updating SKILL.md via git commit.

approach works because all five supported agents write trajectories to predictable paths:

- **Claude Code:** `~/.claude/projects/<proj>/<sid>.jsonl` — JSONL, async batch append.
- **Codex:** `~/.codex/sessions/YYYY/MM/DD/rollout-*.jsonl` — JSONL, streaming append.
- **OpenClaw:** `~/.openclaw/agents/<id>/sessions/*.trajectory.jsonl` — JSONL, per-event sync.
- **OpenCode:** `~/.local/share/opencode/opencode.db` — SQLite (the only non-JSONL format).
- **Hermes:** `<cwd>/trajectory_samples.jsonl` — JSONL, written at run end.

Each trajectory event is normalized into a `TrajectoryEvent` schema with fields: `source` (agent identifier), `session_id`, `ts` (ISO 8601), `kind` (`user_msg` | `assistant_msg` | `tool_call` | `tool_result` | `session_start` | `session_end` | `compact`), and `data` (native payload preserved for replay). The watcher scans registered directories every 30 seconds; new or appended content triggers the TaskAgent pipeline.

**Sanitization.** Before upload, the client applies secret-pattern redaction (API keys, tokens, credentials) using a configurable regex set, ensuring that team-shared trajectories do not leak sensitive data.

## 3.2 Three-Agent Pipeline

The server processes normalized trajectories through three specialized agents (Figure 4):

### 3.2.1 TaskAgent: Trajectory Decomposition

Raw trajectories are multi-topic conversations that may span hours. The TaskAgent segments each trajectory into **AtomTasks**—coherent units representing a single user intent (e.g., “deploy FastAPI to port 1717” or “fix the SQLite migration”). Each AtomTask records:

- `atom_id`: unique identifier (`atom_<traj_id>_NNNN`)

- `intent`, `summary`, `tags`: semantic metadata
- `used_skills`: skills the agent self-reported using
- `ux_score`: user-experience score (0–10), used in canary evaluation
- `offset_start`, `offset_end`: character offsets for provenance
- `pre_atom_id`, `post_atom_id`: linked-list neighbors for context

The UX score is assigned by the TaskAgent based on signals such as: whether the user accepted the agent’s output, whether corrections were needed, and whether the session segment ended naturally or was abandoned.

### 3.2.2 TaskClusterAgent: Skill Routing

Each AtomTask triggers a TaskClusterAgent (TCA) invocation. The TCA receives the atom plus a catalog of existing skills (name + truncated description) and makes one of three decisions:

1. **Route to existing skill**: append the atom to the skill’s `.candidates.yml` with a `weightscore` (0–10) reflecting the atom’s relevance.
2. **Create new skill**: initialize a new skill folder in “baby” state with the atom as its first evidence.
3. **Reclassify**: move an atom from one skill to another if the TCA determines a better fit.

The TCA’s routing is LLM-based but constrained by the existing catalog structure, preventing unbounded skill proliferation while allowing organic growth.

### 3.2.3 SkillEditAgent: Skill Production

When a skill’s `.candidates.yml` accumulates a cumulative `weightscore`  $\geq \theta$  (default  $\theta = 10$ ), the SkillEditAgent (SEA) fires. The SEA reads all candidate atoms (via `AtomTaskRead`), optionally reads raw trajectory text (via `ReadTraj`), and produces or updates the skill’s `SKILL.md` along with any auxiliary files (scripts, references).

The SEA’s commit behavior depends on branch state:

- **Baby branch**  $\rightarrow$  **main**: first public version of a new skill.
- **Main branch**  $\rightarrow$  **staging**: creates a canary candidate for A/B comparison.

This branching model ensures that skill updates are never applied directly to main without canary evaluation.

## 3.3 Canary Evaluation Protocol

The canary protocol (Figure 5) is xskill’s primary quality gate. When a SkillEditAgent commits an updated skill to a staging branch, the system:

1. **Maintains a single active canary per skill**: new candidates queue rather than overwriting, preventing evaluation contamination.
2. **Routes a configurable fraction** of users to the staging version (via skill file sync to their machines).

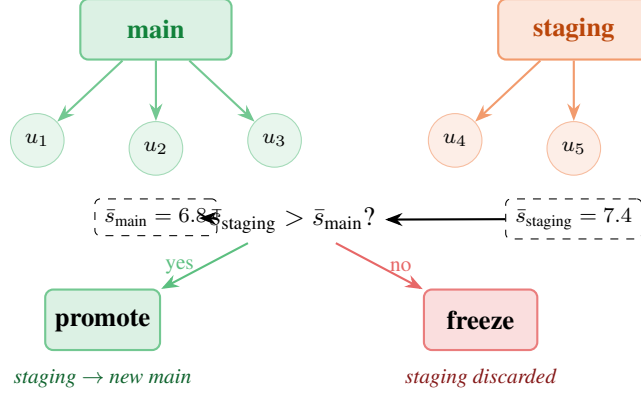


Figure 5: **Canary evaluation protocol.** When the SkillEditAgent produces an updated skill, it commits to a staging branch. The xskill server routes a subset of users ( $u_4, u_5$ ) to the staging version while the majority ( $u_1, u_2, u_3$ ) continue on main. UX scores from AtomTasks using each branch are aggregated; the staging branch is promoted to main only if its mean score exceeds main’s.

Table 2: **Skill output paths for 5 agent ecosystems.** The SKILL.md format with YAML frontmatter (name + description) is the cross-agent interoperability unit. Two filesystem paths—`~/.claude/skills/` and `~/.agents/skills/`—cover 4 of 5 ecosystems; only Hermes requires a dedicated path.

Agent Ecosystem	xskill Output Path	Also Scanned By
Claude Code	<code>~/.claude/skills/&lt;name&gt;/SKILL.md</code>	OpenCode
Codex	<code>~/.agents/skills/&lt;name&gt;/SKILL.md</code>	OpenClaw
OpenCode	<code>~/.claude/skills/&lt;name&gt;/SKILL.md</code>	Claude Code
OpenClaw	<code>~/.agents/skills/&lt;name&gt;/SKILL.md</code>	Codex
Hermes	<code>~/.hermes/skills/&lt;cat&gt;/&lt;name&gt;/SKILL.md</code>	(exclusive)

3. **Collects UX scores** from AtomTasks that use the skill under both branches. Each score  $s \in [0, 10]$  is attributed to the branch (`side=main` or `side=staging`) under which it was observed.
4. **Evaluates:** when sufficient samples accumulate, compares  $\bar{s}_{\text{staging}}$  vs.  $\bar{s}_{\text{main}}$ . If staging wins, it replaces main; otherwise, it is frozen (retained for audit, hidden from runtime).

The UX score is *not* an LLM self-evaluation. It is derived from observable behavioral signals: task completion, absence of user corrections, natural session transitions. This distinguishes xskill’s evaluation from the LLM-as-judge approaches used by OpenSpace and AutoSkill.

### 3.4 Cross-Agent Skill Distribution

A critical finding from our ecosystem survey is that **SKILL.md with YAML frontmatter is the de facto cross-agent standard**. Five of the ten surveyed systems produce this format, and all five supported agent runtimes parse it. Table 2 shows xskill’s output path strategy.

The frontmatter uses a compatible superset:

---

Table 3: **Trigger mechanisms across 10 systems + xskill.** “Async” = non-blocking; “Per-turn” = can fire during a live conversation; “User-transparent” = the user does not explicitly request skill generation.

System	Async	Per-turn	Transparent	Mechanism
Hermes	–	–	–	Offline CLI
OpenSpace	✓	✓	✓	Hook + counter + event
EvoSkill	–	–	–	Offline batch
AutoSkill	✓	✓	✓	Per-turn + background thread
AgentEvolver	–	–	–	Counter in RL loop
MemSkill	–	–	–	Counter in RL loop
EvoAgentX	–	–	–	Offline batch loop
SE-Agent	–	–	–	Offline iteration
SkillRL	–	–	–	Counter + success gate
GEPA	–	–	–	Offline batch
<b>xskill</b>	✓	✓	✓	30s watcher + per-atom async

```

name: <slug>
description: <one-line, <=1024 chars>
version: 1.0.0
metadata:
  hermes: {tags: [...]}
  openclaw: {emoji: "...", requires: {...}}
  short-description: <Codex alias>
---
```

Private fields are silently ignored by non-target runtimes (all five do YAML-tolerant parsing), so a single artifact serves all ecosystems.

## 4 Design Dimensions: A Comparative Analysis

Our survey organizes the trajectory-to-skill landscape into five dimensions. We present key findings per dimension, positioning xskill’s choices.

### 4.1 Trigger Mechanism

Eight of ten systems use offline batch or counter-based triggers (Table 3). Only OpenSpace and AutoSkill achieve user-transparent, per-turn triggering. xskill adopts a 30-second polling watcher with per-atom asynchronous processing, achieving transparency without requiring hook registration in the host agent (though hooks can accelerate detection when available).

### 4.2 Skill Production

All 10 systems use LLM-as-author for skill text generation—no system attempts rule-based, template, or AST-based production. xskill follows this pattern but adds *cumulative evidence*: rather than generating a skill from a single trajectory, the SkillEditAgent waits for multiple weighted atoms to accumulate, reducing the risk of overfitting to a single interaction.

### 4.3 Deduplication and Merging

Deduplication maturity varies enormously: from zero (Hermes, SE-Agent, EvoAgentX) to AutoSkill’s six-layer sandwich. xskill’s TCA performs catalog-aware routing (LLM



Table 4: **Skill injection strategies.** xskill does not inject skills itself—it publishes to disk, and the host runtime’s loader handles injection. This decoupling is deliberate: it preserves each runtime’s native caching, trimming, and compaction behavior.

Strategy	Systems
Full body in system prompt	OpenSpace, AutoSkill, MemSkill, SE-Agent, SkillRL
Directory listing + Read tool	Codex, OpenClaw
CLI autodiscovery (no injection)	EvoSkill
User message injection	AgentEvolver
<b>Publish to disk (runtime-agnostic)</b>	<b>xskill, GEPA-gskill</b>

judgment over existing skill names and descriptions), which provides moderate deduplication without the complexity of AutoSkill’s multi-stage pipeline.

#### 4.4 Evaluation and Retirement

Most systems evaluate skills via offline benchmarks or LLM self-judgment. The “only increase, never delete” pattern is dominant (7/10 systems). xskill introduces canary evaluation (§3.3) and supports skill freezing (retained for audit, removed from active distribution).

#### 4.5 Skill Injection

xskill deliberately avoids injecting skills into agent prompts (Table 4). Instead, it publishes SKILL.md files to the directories each runtime already scans. This preserves native behaviors: Claude Code’s 1% listing budget and compaction, Codex’s progressive disclosure, OpenCode’s full-dump strategy. The management layer shapes *what* is available; the runtime decides *how* to present it.

### 5 Discussion

#### 5.1 Design Trade-offs

**Pull vs. Push trajectory collection.** xskill pulls from known paths rather than requiring agents to push events. This trades real-time latency (30s polling vs. sub-second hooks) for zero-configuration deployment. For Claude Code and Codex, hooks can be registered to reduce latency; the pull mechanism serves as a universal fallback.

**Cumulative evidence vs. single-trajectory skill creation.** Most systems (OpenSpace, AutoSkill, SE-Agent) can create a skill from a single trajectory. xskill requires cumulative weighted evidence ( $\geq \theta$ ), which delays skill creation but reduces noise. This is especially important in team settings where trajectory quality varies across developers.

**UX scoring vs. LLM judging.** xskill’s UX scores are derived from behavioral signals (task completion, user corrections) rather than LLM self-evaluation. This avoids the well-documented biases of LLM-as-judge [14] at the cost of noisier signals.

#### 5.2 Limitations

1. **No formal evaluation:** xskill is an engineering system deployed in team settings, not a controlled experiment. We do not report pass@1 numbers on standardized benchmarks because the system targets real-world skill evolution rather than isolated task completion.

2. **Model dependency:** the server’s three-agent pipeline currently requires Qwen3.6-27B for generation and Qwen3-0.6B-Embed for embeddings. The minimum hardware is a single GPU server with  $\geq 40$ GB VRAM.
3. **Cold start:** a new deployment begins with an empty skill repository. While xskill supports batch trajectory import for cold start, the canary mechanism requires ongoing user traffic to function.
4. **OpenCode adapter:** OpenCode’s SQLite-only trajectory storage makes it the most expensive ecosystem to integrate, requiring either plugin development or database polling.

### 5.3 Broader Impact

xskill’s team-level skill sharing raises questions about intellectual property and knowledge attribution. The sanitization pipeline strips secrets but does not address the subtler issue of whether automated skill distillation from one developer’s work creates implicit IP obligations. Organizations deploying xskill should establish clear policies about skill ownership and attribution.

## 6 Conclusion

We presented xskill, a system for automatic skill distillation, team-level sharing, and data-driven evolution for coding agents. Through a comprehensive survey of 10 trajectory-to-skill systems, we identified that no existing system combines asynchronous cross-agent trajectory collection, cumulative-evidence skill production, canary-based A/B evaluation, and multi-ecosystem distribution. xskill fills this gap by operating as a management layer above existing coding-agent runtimes, shaping the skill supply without interfering with each runtime’s native loading and injection mechanisms.

The emerging consensus on SKILL.md as a cross-agent interoperability format, combined with the predictable filesystem layout of modern coding agents, makes this management-layer architecture both practical and timely. As coding agents become standard development tools, the ability to systematically capture, evaluate, and distribute procedural knowledge across teams will be a critical differentiator. xskill is open-source at <https://github.com/SkillNerds/xskill>.

## References

- [1] Anthropic. Claude Code: An agentic coding tool. <https://claude.com/claude-code>, 2025.
- [2] OpenAI. Codex CLI: Lightweight coding agent. <https://github.com/openai/codex>, 2025.
- [3] SST. OpenCode: Open-source coding agent. <https://github.com/sst/opencode>, 2025.
- [4] Chen, Y., et al. OpenSpace: An evolving skill library via MCP for code agents. <https://github.com/HKUDS/OpenSpace>, 2025.
- [5] Wang, J., et al. EvoSkill: Evolutionary program synthesis for agent skills. <https://github.com/sentient-agi/EvoSkill>, 2025.

- [6] Li, X., et al. AutoSkill: Automated skill management for coding agents. <https://github.com/ECNU-ICALK/AutoSkill>, 2025.
- [7] Qwen Applications. Trace2Skill: From agent traces to reusable skills. <https://github.com/Qwen-Applications/Trace2Skill>, 2025.
- [8] ModelScope. AgentEvolver: Evolving agents with reinforcement learning. <https://github.com/modelscope/AgentEvolver>, 2025.
- [9] Axelsen, V. MemSkill: Memory-augmented skill learning. <https://github.com/ViktorAxelsen/MemSkill>, 2025.
- [10] EvoAgentX Team. EvoAgentX: Evolutionary agent workflow optimization. <https://github.com/EvoAgentX/EvoAgentX>, 2025.
- [11] Xu, S., et al. SE-Agent: Self-evolving agents for software engineering. <https://github.com/JARVIS-Xs/SE-Agent>, 2025.
- [12] Aiming Lab. SkillRL: Skill-augmented reinforcement learning for agents. <https://github.com/aiming-lab/SkillRL>, 2025.
- [13] GEPA Team. GEPA: Generalized evolutionary prompt optimization. *arXiv preprint arXiv:2507.19457*, 2025.
- [14] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2024.

## A Full Comparison Matrix

Table 5: **Cross-system comparison on “skill” definition.** The term “skill” refers to fundamentally different artifacts across systems, complicating direct comparison.

#	System	“Skill” Artifact
1	Hermes	Existing SKILL.md body (mutated in place)
2	OpenSpace	Directory: SKILL.md + scripts/ + references/ + SQLite lineage
3	EvoSkill	.claude/skills/<name>/SKILL.md (YAML frontmatter)
4	AutoSkill	Per-user SKILL.md with semantic semver patches
5	AgentEvolver	Vector-store entry in external ReMe service
6	MemSkill	RL operation (INSERT/UPDATE/DELETE template)
7	EvoAgentX	Workflow code: round_N/{graph.py, prompt.py}
8	SE-Agent	Per-instance system-prompt YAML
9	SkillRL	JSON record: {title, principle, when_to_apply}
10	GEPA	Candidate dict[str,str] (arbitrary optimizable text)
11	<b>xskill</b>	SKILL.md + auxiliary files, git-versioned, canary-evaluated

## B AtomTask Schema

```
{
  "atom_id": "atom_traj_cc_dsv4_890da9d9_0001",
  "traj_id": "traj_cc_dsv4_890da9d9",
  "offset_start": 120,
  "offset_end": 2400,
  "intent": "deploy xquiz on port 1717",
  "summary": "agent cloned repo, read README, configured port, started",
  "tags": ["deploy", "fastapi"],
  "used_skills": ["python-deploy"],
  "ux_score": 7,
  "pre_atom_id": null,
  "post_atom_id": "atom_traj_cc_dsv4_890da9d9_0002",
  "context_prefix": "",
  "raw_segment": ""
}
```

## C Canary Guard Conditions

The SkillEditAgent is guarded by three conditions before committing to a staging branch:

1. No staging branch currently exists for this skill (one canary at a time; new candidates queue).

2. Cumulative `weightscore` of candidates  $\geq \theta$  (default 10).
3. If on main branch: at least one real `side=main` UX score exists (proves the current main version has been used, establishing a baseline for comparison).

These guards prevent evaluation contamination from concurrent canaries and ensure that staging branches compete against a used—not merely published—main version.