On this page >

← Blog (https://runcycles.io/blog/)

April 3, 2026 · Albert Mavashev · 17 min read

risk-assessment | risk-scoring | action-control | RISK_POINTS | agents | production | governance

Copy link | Share | Share | PDF (https://runcycles.io/pdfs/ai-agent-risk-assessment-score-classify-enforce-tool-risk.pdf)

AI Agent Risk Assessment: How to Score, Classify, and Enforce Tool Risk Before Production

> **Part of:** AI Agent Risk & Blast Radius Reference ↗ — the full pillar covering action authority, risk scoring, blast-radius containment, and degradation paths.

A customer-onboarding agent sends 200 collections emails instead of welcome emails. Total model cost: $1.40. Business impact: $50K+ in lost pipeline ↗. A coding agent retries an ambiguous error 240 times. Total model cost: $4,200 ↗. Both agents passed every test. Both were under their dollar budgets. In both cases, the agent had no tool-level risk assessment.

These are not anomalies. They are what happens when teams deploy agents without classifying what those agents can do. Model risk — bias, hallucination, drift — gets attention. **Tool risk** — what happens when the agent acts on the world — does not.

This post is a practical guide to assessing agent risk at the tool level: how to inventory your agent's capabilities, classify each one by blast radius, assign risk scores, and convert those scores into enforceable runtime budgets.

Why AI Agents Need a Distinct Risk Assessment

Traditional AI risk assessments focus on model behavior: Is the output biased? Does it hallucinate? Is it robust to adversarial inputs? These matter, but they evaluate the *reasoning* layer. Agents add a second layer — the *action* layer — where the model's outputs become tool calls that change state in the real world.

The distinction is operational:

Risk Type	What Goes Wrong	How You Detect It	What You Lose
Model risk	Wrong answer, biased output, hallucination	Output evaluation, benchmarks, user reports	Trust, accuracy
Tool risk	Wrong action, excessive action, unauthorized action	After the action has already happened	Revenue, data, reputation, compliance standing

Model risk produces bad answers. Tool risk produces bad *consequences*. A bad answer can be corrected. A sent email, a triggered deploy, a deleted record, a processed payment — these persist after the agent stops.

For AI agents that qualify as high-risk AI systems, the EU AI Act's Article 9 ↗ requires a risk management system that identifies and mitigates foreseeable risks throughout the system's lifecycle. The NIST AI RMF's Map function ↗ requires understanding the AI system's context and potential impacts. Both require risk assessment *before* deployment — not incident review after.

A tool-level risk assessment is a practical way to operationalize these requirements for agents specifically.

The Three Risk Dimensions

Agent risk is not one-dimensional. Teams that assess only cost exposure ↗ miss the two dimensions where the worst incidents happen.

Dimension 1: Cost Risk

The agent spends more than expected. Runaway loops, retry storms ↗, expanding context windows, concurrent agent bursts. The failure mode is financial — unexpected invoices, blown budgets, margin erosion.

Unit of measurement: Dollars, tokens ↗, API calls. **Existing coverage:** Most teams assess this first. See How Much Do AI Agents Actually Cost? ↗ and AI Agent Cost Management ↗ for detailed analysis.

Dimension 2: Action Risk

The agent does something it should not have done, or does the right thing too many times. Wrong emails, accidental deploys, data deletion, ticket storms. The failure mode is operational — customer impact, compliance violations, production incidents.

Unit of measurement: Blast radius, reversibility, affected parties. **Existing coverage:** Covered in AI Agent Action Control ↗ and 5 Failures Only Action Controls Would Prevent ↗.

Dimension 3: Delegation Risk

The agent spawns sub-agents, delegates tasks, or hands off to other agents. Each delegation multiplies exposure — the parent's budget covers the child's mistakes. A three-level delegation chain with a 3x retry multiplier at each level creates 27x worst-case cost amplification.

Unit of measurement: Delegation depth, sub-agent count, inherited permissions. **Existing coverage:** Covered in Multi-Agent Budget Control ↗.

A complete risk assessment evaluates all three dimensions for every agent before it reaches production.

The Tool Risk Classification Framework

Every tool your agent can call has a risk profile. Classifying tools by tier is the foundation of risk assessment — it forces the question most teams skip: *which of these actions should the agent be allowed to take without a pre-execution check?*

Five Tiers

Tier	Class	Examples	Reversibility	Blast Radius	Enforceme Pattern
0	**Read-only**	Search, retrieve, summarize, vector lookup	No state change	None (side-effect risk only; sensitive-read risk may be budgeted separately)	Event ↗ (post-hoc accounting
1	**Write-local**	Save draft, write log, update cache, create temp file	Easy to reverse	Internal only	Event or reserve-commit depending on volume
2	**Write-external**	API call to third party, webhook trigger, external query	Possible but not guaranteed	Partner systems affected	Reserve-commit ↗ (always)
3	**Mutation**	DB write/update/delete, send email, post to Slack, create ticket	Difficult or impossible	Customer-facing	Reserve-commit wit caps
4	**Execution**	Deploy, payment processing, infrastructure change, permission grant	Irreversible in practice	Production users, financial, regulatory	Reserve-commit wit strict allowlist

The Classification Decision Tree

Edge cases are common. A "file write" could be Tier 1 (writing a temp log) or Tier 4 (overwriting a production config). Use these three questions to classify:

1. Does this action leave your system boundary? If yes → Tier 2 or higher. External actions involve systems you do not control. Reversal requires coordination with third parties, if it is possible at all.

2. Can a human undo this action in under 5 minutes? If yes → Tier 0 or 1. The action is low-risk because its consequences are contained and reversible. If no — if undoing requires a database restore, a customer apology, or a rollback deploy — the action is Tier 3 or higher.

3. Does this action affect someone who did not request it? If yes → Tier 3 or higher. Actions that affect end users, customers, or external parties carry reputational and regulatory risk regardless of their technical complexity. A Slack message to an internal channel is Tier 1. A Slack message to a customer-shared channel is Tier 3.

Classification Example: Customer Support Agent

A support agent with 12 tools, classified tool by tool:

Tool	What It Does	Leaves System?	Reversible <5min?	Affects Others?	
`search_knowledge_base`	Searches internal docs	No	N/A (read)	No	
`get_customer_record`	Reads CRM entry	No	N/A (read)	No	
`get_order_history`	Reads order data	No	N/A (read)	No	
`summarize_conversation`	Generates summary	No	N/A (read)	No	
`save_draft_response`	Saves draft to queue	No	Yes	No	
`add_internal_note`	Appends note to CRM	No	Yes	No	
`lookup_shipping_status`	Calls carrier API	Yes	N/A (read)	No	
`send_customer_email`	Sends email to customer	Yes	No	Yes	
`create_support_ticket`	Creates ticket in Jira	Yes	Difficult	Yes (triggers notifications)	
`update_crm_record`	Modifies customer data	No	Difficult	Yes (downstream systems)	
`issue_refund`	Processes financial refund	Yes	No	Yes	

Tool	What It Does	Leaves System?	Reversible <5min?	Affects Others?	
`escalate_to_billing`	Triggers billing workflow	Yes	No	Yes	

Four Tier 0 tools (read-only), two Tier 1 (reversible local writes), one Tier 2 (external read), three Tier 3 (mutations with customer impact), two Tier 4 (financial/irreversible).

This classification alone tells you something important: **half the agent's tools are Tier 0–1 (safe), but a third are Tier 3–4 (dangerous).** The risk profile is not evenly distributed — the dangerous tools are a minority, but they carry the majority of the blast radius.

A note on reads in regulated environments. The tier system isolates side-effect risk — actions that change state in the world. But in regulated or exfiltration-prone environments, sensitive read tools may warrant non-zero risk points or a separate data-access budget. An agent that reads 10,000 customer records in a single run has not changed any state, but it has created a data-exposure surface that OWASP ASI03 (identity and privilege abuse) ↗ and NIST's Map function ↗ both flag as a risk. If your agent handles PII, financial, or health data, consider assigning 1–2 risk points to sensitive reads or tracking them under a separate `DATA_ACCESS` budget that limits read volume independently of action risk.

Assigning Risk Scores: The RISK_POINTS Methodology

Classification tells you *which* tools are dangerous. Risk scoring tells you *how much* danger to allow per run. RISK_POINTS ↗ are budgets denominated in blast radius, not dollars — the mechanism that turns a risk assessment into an enforceable limit.

Step 1: Assign Base Points Per Tier

Start with a baseline that reflects relative risk. The specific numbers are less important than the ratios between tiers.

Tier	Base RISK_POINTS ↗	Rationale
0	0	No side effects — reads should be free
1	1	Low impact, easily reversible
2	5	External dependency, some coordination to reverse
3	20	Customer-facing impact, difficult to reverse
4	50	Irreversible, financial or production impact

These are starting points. The existing examples ↗ in the Cycles documentation use similar scales — 1–2 for low-impact internal actions, 20 for emails, 50 for deploys — but your numbers should reflect your organization's risk tolerance, not a universal standard.

Step 2: Apply Context Multipliers

Base points assume a generic context. Four factors adjust them for your specific deployment:

Audience size. An email to an internal team (10 people) is different from an email to a customer segment (10,000 people). The same action, different blast radius.

Audience	Multiplier
Internal only	1x
Single external party	1.5x
Customer segment (<100)	2x
Broad external (100+)	3x

Data sensitivity. A CRM note about a meeting is different from a CRM note containing PII or financial data.

Sensitivity	Multiplier
Public / non-sensitive	1x
Internal / business confidential	1.5x
PII or customer data	2x
Financial, health, or regulated data	3x

Regulatory exposure. Actions in regulated contexts carry higher risk because the consequences include compliance violations, not just operational impact.

Regulatory Context	Multiplier
No specific regulation	1x
General data protection (GDPR)	1.5x
Industry-specific (HIPAA, PCI-DSS, SOX)	2x
Multiple overlapping regulations	3x

Reputational exposure. Some actions are disproportionately damaging not because of their technical impact but because of who sees them and how easily the consequences spread. A wrong email to an internal team is a correction. The same wrong email to a customer is a lost deal. The same wrong email screenshotted on social media is a news cycle. The technical action is identical — the reputational blast radius is not.

Visibility	Multiplier
Internal only (team channels, internal tools)	1x
Single external party (one customer, one vendor)	1.5x
Public-facing or shared channel (customer portal, community Slack)	2x
Press, analysts, social media, or regulatory bodies	3x

This multiplier captures a risk dimension the other three miss. Audience size counts heads. Data sensitivity classifies information. Regulatory exposure measures legal consequence. Reputational exposure measures **trust consequence** — how much brand credibility is at stake if this action goes wrong, and how easily the damage amplifies beyond the initial blast radius.

Consider: a support agent that sends a wrong email to one customer (1.5x) versus a support agent that posts a wrong response in a public community forum (2x) versus a support agent whose hallucinated response gets screenshotted in a viral post about AI failures (3x). Same underlying action, same data sensitivity, same regulatory context — but the reputational exposure is the difference between a support ticket and a PR crisis.

Formula: `Final RISK_POINTS = Base × max(Audience, Sensitivity, Regulatory, Reputational)`

Use the highest multiplier, not the product — stacking all four would inflate scores beyond useful range. The point is to shift tools that are particularly dangerous in your context above the baseline.

Step 3: Score the Support Agent

Applying the methodology to the 12-tool support agent, assuming: B2B SaaS (500+ customer accounts), customer PII in CRM (2x data sensitivity), GDPR-applicable (1.5x regulatory), customer emails are forwardable/screenshotable (2x reputational for customer-facing actions):

Tool	Tier	Base	Multiplier	Final RISK_POINTS	Rationa
`search_knowledge_base`	0	0	—	**0**	Read-c
`get_customer_record`	0	0	—	**0**	Read-c access modifie
`get_order_history`	0	0	—	**0**	Read-c
`summarize_conversation`	0	0	—	**0**	Read-c
`save_draft_response`	1	1	1x	**1**	Interna mutatic externa
`add_internal_note`	1	1	2x (PII)	**2**	CRM n contair data
`lookup_shipping_status`	2	5	1x	**5**	Externa state c
`send_customer_email`	3	20	2x (reputational: customer-facing, forwardable)	**40**	Custom can screen — trust
`create_support_ticket`	3	20	2x (reputational: visible in customer portal)	**40**	Ticket custom trigger notifica
`update_crm_record`	3	20	2x (PII mutation)	**40**	Modifie data, a downst system

Tool	Tier	Base	Multiplier	Final RISK_POINTS	Rationa
`issue_refund`	4	50	3x (sensitivity: financial data)	**150**	Financi commi irrever sensiti multipli domina
`escalate_to_billing`	4	50	3x (sensitivity: triggers financial workflow)	**150**	Financi custom billing commu

Two things to notice in these scores. First, the Tier 4 tools score highest not because of a single overwhelming multiplier but because financial data sensitivity (3x) outweighs the reputational exposure (2x) — the `max()` function selects the highest. Second, the Tier 3 tools are where reputational exposure is the dominant multiplier: `create_support_ticket` scores 40 (2x reputational, because the ticket is visible in the customer portal) rather than 30 (which 1.5x audience alone would give). The multiplier framework surfaces the risk dimension that matters most for each tool, rather than applying a uniform factor.

Step 4: Set the Per-Run Budget

The per-run budget defines how much action surface you are willing to expose in a single agent execution. It should reflect what a *normal, successful run* looks like — with headroom for variation, but not enough to cause the worst-case scenario.

For the support agent, a typical resolution involves:

- 4–6 knowledge base searches (0 points each)
- 1–2 CRM reads (0 points each)
- 1 internal note (2 points)
- 1 customer email (40 points)
- Occasionally: 1 ticket creation (40 points) or 1 CRM update (40 points)

Normal run: ~42–82 points. Set the per-run budget at **250 points** — enough for a complex resolution (email + ticket + CRM update = 120 points) with ample headroom, and enough for a single refund plus one email (150 + 40 = 190), but not enough for a refund plus a billing escalation.

What this prevents:

- Agent cannot send more than 6 emails in a single run (6 × 40 = 240 points) — and in practice, other actions consume budget first
- Agent cannot issue a refund *and* send more than 2 emails (150 + 80 = 230 points, leaving only 20 for notes)
- Agent can search and read without limit (0 points each)
- Agent cannot issue 2 refunds in a single run (2 × 150 = 300 > 250)
- Agent cannot escalate to billing *and* issue a refund (150 + 150 = 300 > 250)

The budget makes risk trade-offs explicit. The agent can reason freely but must prioritize its consequential actions.

The Risk Assessment Worksheet

This is the artifact your team fills out before deploying an agent. It documents the classification, scoring, and budget decisions — and becomes audit evidence for Article 9 compliance ↗.

```yaml
# AI Agent Risk Assessment Worksheet
# Fill out one per agent before production deployment

agent:
  name: support-copilot
  owner: customer-experience-team
  description: "Resolves Tier 1 support tickets via chat and email"
  deployment_date: 2026-04-15
  assessment_date: 2026-04-03
  assessor: jane.doe@example.com

tools:
  - name: search_knowledge_base
    tier: 0
    risk_points: 0
    enforcement: event
    rationale: "Read-only, internal docs, no state change"

  - name: get_customer_record
    tier: 0
    risk_points: 0
    enforcement: event
    rationale: "Read-only CRM access, PII viewed but not modified"

  - name: save_draft_response
    tier: 1
    risk_points: 1
    enforcement: event
    rationale: "Internal draft queue, easily deleted"

  - name: add_internal_note
    tier: 1
    risk_points: 2
    enforcement: reserve-commit
    rationale: "CRM note, may contain PII — 2x multiplier"

  - name: lookup_shipping_status
    tier: 2
    risk_points: 5
    enforcement: reserve-commit
    rationale: "External carrier API, read-only"

  - name: send_customer_email
    tier: 3
    risk_points: 40
    enforcement: reserve-commit
    rationale: "Customer-facing, forwardable/screenshotable — 2x reputational"

  - name: create_support_ticket
    tier: 3
    risk_points: 40
    enforcement: reserve-commit
```

```yaml
    rationale: "Visible in customer portal — 2x reputational"

  - name: update_crm_record
    tier: 3
    risk_points: 40
    enforcement: reserve-commit
    rationale: "PII mutation, affects downstream systems — 2x data sensitivity"

  - name: issue_refund
    tier: 4
    risk_points: 150
    enforcement: reserve-commit
    rationale: "Financial data, irreversible — 3x sensitivity (financial)"

  - name: escalate_to_billing
    tier: 4
    risk_points: 150
    enforcement: reserve-commit
    rationale: "Financial workflow, customer-visible — 3x sensitivity (financial)"

budget:
  per_run: 250
  unit: RISK_POINTS
  rationale: >
    Normal resolution: 42-82 points.
    Complex resolution (email + ticket + CRM update): 120 points.
    Budget of 250 allows complex resolutions and a single refund + email,
    but prevents double-refund or refund + billing escalation.

  degradation_thresholds:
    - consumed_percent: 50
      action: "Disable Tier 4 tools (issue_refund, escalate_to_billing) — remaining budget cannot
    - consumed_percent: 80
      action: "Disable Tier 3 tools (send_customer_email, create_support_ticket, update_crm_record
    - consumed_percent: 100
      action: "DENY — read-only mode, summarize and stop"

  cost_budget:
    per_run: 5.00
    unit: USD
    rationale: "Typical run costs $0.30-$1.50. $5 cap prevents retry loops."

context:
  audience: "B2B SaaS customers (500+ accounts)"
  data_sensitivity: "PII (customer names, emails, order history)"
  regulatory: "GDPR"
  reputational: "Customer-facing emails and portal tickets are forwardable/screenshotable; wrong d
  delegation: "None — single agent, no sub-agent spawning"

review_schedule: "Quarterly, or after any incident involving this agent"
```

This worksheet is not a Cycles configuration file — it is a risk assessment document. The next section shows how it translates into enforceable controls.

From Assessment to Enforcement

A risk assessment without enforcement is documentation. It satisfies a checkbox but does not prevent incidents. The assessment becomes operational when it translates into runtime budgets.

Step 1: Create the Budget

The worksheet's `budget` section translates into Cycles budgets at two levels. The **app-level budget** sets a ceiling for the agent across all runs. The **per-run budget** is created dynamically for each execution and enforces the per-run limit from the worksheet.

App-level budget — created once via the Admin API (port 7979):

```bash
# Create an app-level risk-point ceiling for the support agent
curl -s -X POST http://localhost:7979/v1/admin/budgets \
  -H "Content-Type: application/json" \
  -H "X-Cycles-API-Key: $CYCLES_API_KEY" \
  -d '{
    "scope": "tenant:acme-corp/app:support-copilot",
    "unit": "RISK_POINTS",
    "allocated": { "amount": 10000, "unit": "RISK_POINTS" }
  }'
```

Per-run budget — created dynamically when each run starts. Each execution gets a unique workflow-scoped ID, so the workflow scope acts as the run boundary:

```bash
# Create a per-run risk-point budget (250 points per the worksheet)
curl -s -X POST http://localhost:7979/v1/admin/budgets \
  -H "Content-Type: application/json" \
  -H "X-Cycles-API-Key: $CYCLES_API_KEY" \
  -d '{
    "scope": "tenant:acme-corp/app:support-copilot/workflow:run-abc-123",
    "unit": "RISK_POINTS",
    "allocated": { "amount": 250, "unit": "RISK_POINTS" }
  }'
```

The same pattern applies to cost budgets, using `USD_MICROCENTS` (1 USD = 100,000,000 microcents):

```bash
# Create a per-run cost budget ($5.00 = 500,000,000 microcents)
curl -s -X POST http://localhost:7979/v1/admin/budgets \
  -H "Content-Type: application/json" \
  -H "X-Cycles-API-Key: $CYCLES_API_KEY" \
  -d '{
    "scope": "tenant:acme-corp/app:support-copilot/workflow:run-abc-123",
    "unit": "USD_MICROCENTS",
    "allocated": { "amount": 500000000, "unit": "USD_MICROCENTS" }
  }'
```

Both risk-point and cost budgets enforce independently. The agent can be under its dollar budget but over its risk-point budget — or vice versa. This is the dual-control model: cost authority and action authority ↗ operating in parallel. Budgets at both levels enforce hierarchically — a run cannot exceed its own 250-point limit, and all runs together cannot exceed the app-level ceiling. See common budget patterns ↗ for more examples.

Step 2: Validate with Shadow Mode

Before turning enforcement on, run the assessment against real traffic. Shadow mode ↗ evaluates every action against the budget but never denies. The output shows:

- How many runs would have been denied (budget too tight)
- Which tools consume the most risk points (scoring calibration)
- How actual usage compares to your "normal run" estimate
- Whether the degradation thresholds trigger at the right points

Run shadow mode for at least one week of production traffic. If more than 5% of runs would have been denied, the budget is too tight — either raise the per-run limit or re-examine the tool scores. If zero runs would have been denied, the budget may be too loose — it would not have caught the incidents you designed it to prevent.

Step 3: Calibrate and Adjust

Risk assessments are living documents. Scores should change when:

- **A new tool is added.** Classify it, score it, update the worksheet before deployment.
- **An incident occurs.** If a tool caused damage, re-evaluate its tier and multiplier. The $50K email incident would justify raising `send_customer_email` from 40 to 60 points.
- **Usage patterns shift.** If shadow mode shows agents consistently using 230 of 250 points on normal runs, the budget is too tight for safe operation — raise it or optimize the workflow.
- **Context changes.** Entering a new market with different regulatory requirements, expanding the customer base, or adding delegation capabilities all change the multipliers.

Document every change in the worksheet with a date and rationale. The change history is itself audit evidence — it demonstrates the "continuous, iterative" risk management that Article 9 requires of high-risk AI systems ↗.

Risk Assessment and Regulatory Compliance

The worksheet is not just an engineering artifact. It maps directly to regulatory requirements:

Requirement	What the Worksheet Provides
EU AI Act Article 9 ↗ — Risk management system	Tool inventory, risk classification, mitigation (budgets), ongoing review schedule
NIST AI RMF — Map ↗	Identification of risk surfaces (tools), impact assessment (tiers), context factors
NIST AI RMF — Measure ↗	Baseline risk scores, shadow mode metrics, calibration data
ISO 42001 — Risk assessment ↗	Documented risk identification, scoring methodology, proportionate controls
OWASP ASI02 — Tool misuse ↗	Per-tool classification and enforcement pattern

The worksheet is evidence you assessed the risk. The shadow mode report is evidence you validated the assessment. The enforcement log is evidence you acted on it. Together, they form a complete risk management chain — identification, assessment, mitigation, monitoring — documented through normal operation rather than separate compliance processes.

Summary: The Five-Step Assessment Process

1. **Inventory** every tool your agent can call.
2. **Classify** each tool by tier (0–4) using the three-question decision tree.
3. **Score** each tool with base RISK_POINTS and context multipliers.
4. **Budget** the per-run limit based on what a normal, successful run looks like.
5. **Validate** with shadow mode, then enforce and calibrate continuously.

The assessment can usually be done in an afternoon for a typical agent with 10–20 tools. The enforcement is permanent.

Further Reading

- Action Authority: Controlling What Agents Do ↗ — RISK_POINTS concept and toolset-scoped budgets
- AI Agent Action Control: Hard Limits on Side Effects ↗ — the five-tier action taxonomy and progressive capability narrowing
- 5 Failures Only Action Controls Would Prevent ↗ — incident math for action failures
- Shadow Mode ↗ — validating enforcement without blocking production
- The AI Agent Governance Framework ↗ — mapping regulatory requirements to runtime controls
- Understanding Units in Cycles ↗ — USD_MICROCENTS ↗, TOKENS, CREDITS ↗, and RISK_POINTS
- Degradation Paths ↗ — DENY, downgrade, disable, or defer

Related how-to guides

- Assigning RISK_POINTS to agent tools ↗
- Webhook integrations ↗
- Integrating with OpenAI ↗

MORE FROM THE BLOG

Python AI Agent Control: Cost, Risk, and Audit by Layer
May 6, 2026
(https://runcycles.io/blog/python-ai-agent-control-cost-risk-audit-layers)

Agent Memory Writes Are Actions, Too
May 16, 2026
(https://runcycles.io/blog/agent-memory-writes-are-actions-too)

AI Agent Approval Queues Need Runtime Authority
May 11, 2026
(https://runcycles.io/blog/ai-agent-approval-queues-need-runtime-authority)

← Back to all posts (https://runcycles.io/blog/)