Barkhausen AI

CONVENTION · BA-C-6

Version 1.0 · Stable

A taxonomy of AI-related crawlers

"AI crawler" names a function, not a piece of software, and it conflates several.

“Every leap begins as a crackle.”

DOCUMENT	BA-C-6
VERSION	1.0 (Stable)
EFFECTIVE	2026
SUBJECTS	Crawlers & access; Conventions & policy
LICENSE	CC-BY-4.0
AUTHOR	Barkhausen AI

ABSTRACT

"AI crawler" names a function, not a piece of software, and it conflates several. A single site is visited by crawlers that collect content for model training, that build retrieval indexes AI answers draw from, that fetch one page in real time to answer a specific prompt, that act autonomously on a user's behalf, and that maintain traditional search indexes AI features also consume. Each function has different consequences for a site, and a robots.txt rule aimed at one silently binds or misses the others. This convention defines five functional classes — training, retrieval, user-fetch, agent, and search — from vendor documentation and public observation, states how a published crawler registry or census must classify by function rather than operator, and sets out how robots.txt semantics interact with each class. It records where the scheme's boundaries are unstable, including the agent class, which user-agent strings systematically undercount.

CONTENTS

"AI crawler" names a function, not a piece of software, and it conflates several. The phrase is used as though it picked out one kind of visitor a site receives, when the crawlers it names are doing at least five different jobs, each with a different effect on the site that admits or refuses them. A rule written to address "AI" therefore addresses whichever of those jobs its author happened to have in mind, and silently binds, or silently misses, the rest. This convention defines the classes so that a decision about one can be made without accidentally being a decision about the others.

The classification is functional: it sorts crawlers by what they do with a page — as operators document it or public observation establishes it — not by which company runs them. An operator that runs a training crawler, a retrieval crawler, and a real-time fetcher runs three members of three classes, controlled by three tokens; grouping them under the operator's name obscures exactly the distinction a site owner, a researcher, or a census needs. The companion note on robots.txt and AI crawlers sets out the practical failure this prevents — the mis-block, in which a rule aimed at one class removes a site from a service governed by another — and the AI crawler registry applies the scheme to named tokens as a living record. This convention is the scheme itself.

Normative note. The key words MUST, MUST NOT, SHOULD, and MAY carry their conventional standards meaning as used in interoperability specifications. They govern how a registry or census classifies and discloses; they place no obligation on, and express no judgment of, any crawler operator. A classification is a factual statement about documented or observed function, checkable against the sources a compliant record must cite.

1 1. Why classification matters

The classes matter because their consequences diverge. Admitting a training crawler affects whether a site's content enters a model-training corpus; it has, on its own, no bearing on whether the site appears in any answer. Refusing a retrieval crawler affects whether the site can appear in a specific AI system's generated answers; it has no bearing on training collection. The two decisions point in opposite directions for a site that wants visibility in AI answers but not presence in training data, and a single broad rule cannot express that preference. Presence in a training corpus, further, does not imply that a model retains or reproduces the content; no causal claim is made by classifying a crawler as training-class.

Because each class is controlled by a distinct token, and the tokens are maintained separately and revised over time, a rule aimed at a name ("this company's bot") rather than a function reaches whatever that name currently resolves to, and a rule that was correct when written can become incorrect as a new token is introduced. Classification by function is what makes an access decision stable against those changes, and what makes an aggregate count — how many operators run a retrieval crawler, how many training crawlers honor a per-token opt-out — a well-defined quantity rather than a list of names.

2 2. Five functional classes

This convention defines five classes. The table summarizes them; the subsections that follow give the normative definition, the consequence of admitting or refusing each class, documented examples using real tokens, and the way robots.txt interacts with the class.

CLASS	FUNCTION	DOCUMENTED EXAMPLES (TOKEN)	WHAT AN ACCESS DECISION CONTROLS
Training	Collects content for a model-training corpus	GPTBot, ClaudeBot, CCBot, Applebot-Extended, Google-Extended	Presence in that operator's training data
Retrieval	Builds a retrieval/answer index queried at answer time	OAI-SearchBot, PerplexityBot, Claude-SearchBot	Eligibility to appear in that system's AI answers
User-fetch	Fetches one page in real time for a specific prompt	ChatGPT-User, Perplexity-User, Claude-User, Google-NotebookLM	Whether a user-cited page can be retrieved live
Agent	Navigates and acts autonomously on a user's behalf	Google-Agent	Whether user-directed agents may operate on the site
Search	Maintains a traditional search index AI features also use	Googlebot, bingbot, Applebot	Classic search inclusion and AI features drawing on it

Figure 1 lays the five classes out as lanes running from a single site to the outcome each governs, and marks whether a per-crawler robots.txt token exists to control it.

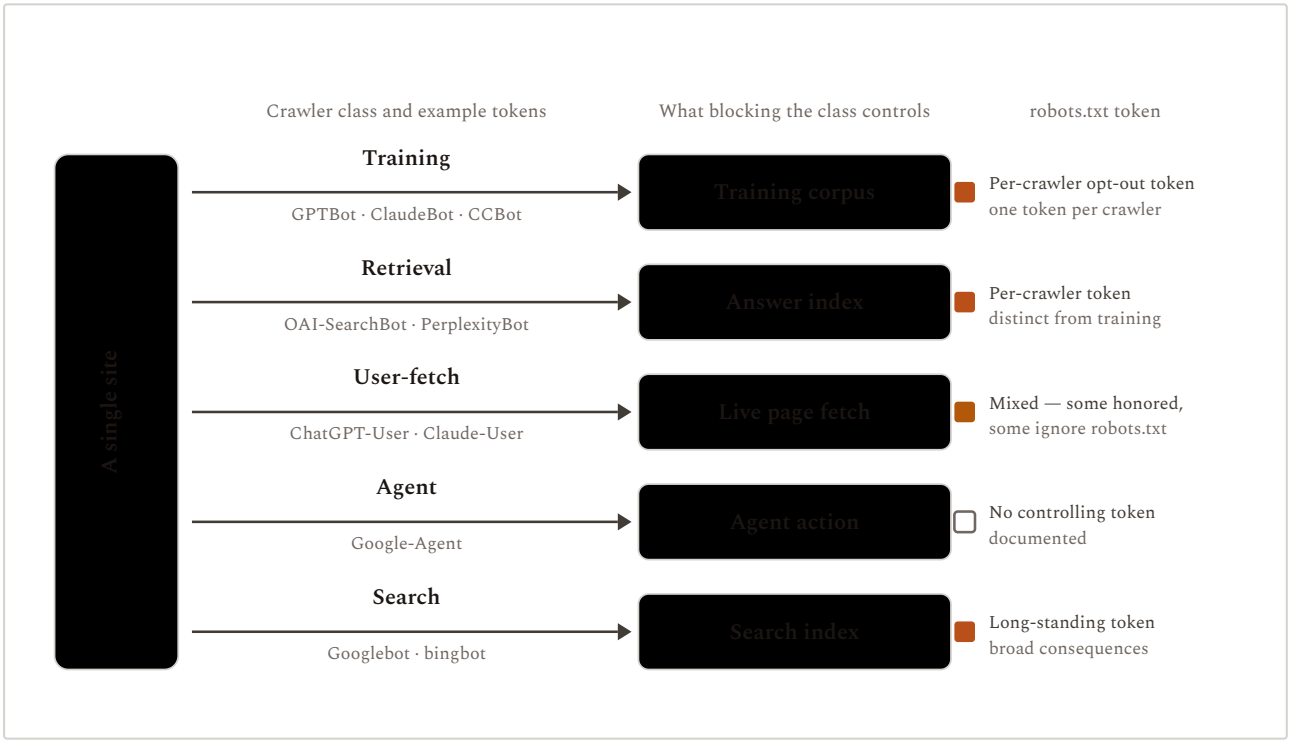

Figure 1. The five crawler classes drawn as lanes from a single site to the outcome each class's access decision controls, with the right column marking whether a per-crawler robots.txt token exists to govern that class. Built from the section 2 table: training, retrieval, and search classes are each controlled by their own token; the user-fetch class is mixed, with some tokens honored and some documented as generally ignoring robots.txt; and the agent class has no controlling token documented.

Barkhausen AI · BA-C-6

2.1 Training

A **training crawler** collects web content for inclusion in a corpus used to train models. Its access decision controls training presence and nothing else on its own: admitting it makes a site eligible for that operator's training collection, and refusing it, where the operator honors the refusal, keeps the site out of

that collection without affecting whether the site can be retrieved, indexed for answers, or searched. Documented examples include GPTBot, which OpenAI documents as crawling content that may be used to train its models [2]; ClaudeBot; Amazonbot; Applebot-Extended; and CCBot, whose resulting open dataset is a widely used input into many organizations' training pipelines and is unaffiliated with any single AI system [6]. Each is controlled by its own robots.txt token — `GPTBot`, `CCBot`, `Applebot-Extended` — so a training opt-out has to name the specific token it intends to reach.

2.2 Retrieval

A **retrieval crawler** builds a retrieval or answer index that an AI system queries at the moment it composes an answer. Its access decision has a visibility consequence that a training decision does not: refusing a retrieval crawler tends to remove a site from the pool of sources a specific AI system can cite or mention, so the site becomes ineligible to appear in that system's answers even though its training presence is unaffected. Documented examples include OAI-SearchBot, which OpenAI documents as surfacing sites in ChatGPT's search features [2]; PerplexityBot; and Claude-SearchBot. Each carries its own token — `OAI-SearchBot`, `PerplexityBot`, `Claude-SearchBot` — distinct from the same operator's training token, which is why a rule can grant retrieval access while refusing training collection, or the reverse, but only if it names the right token for each.

2.3 User-fetch

User-fetch traffic is a real-time fetch of a specific page, issued because a user asked an AI system about that page or a task that required it. It differs from retrieval crawling in cardinality and trigger: it fetches the pages a user's prompt implicates, when the prompt is made, rather than crawling the web in bulk to maintain an index. Its access decision controls whether a page a user points an assistant at can be retrieved live at all. Documented examples include ChatGPT-User, which OpenAI documents as visiting a page when a user asks ChatGPT about it [2]; Perplexity-User; Claude-User; and Google-NotebookLM, which Google documents as requesting individual URLs that NotebookLM users have supplied as sources [5]. Robots.txt interaction is not uniform across this class: some user-fetch tokens are honored, while Google documents that its user-triggered fetchers "generally ignore robots.txt rules" because the fetch was requested by a user [5] — a divergence the registry records by the absence of a controlling token.

2.4 Agent

Agent traffic is autonomous traffic from software acting on a user's behalf — navigating pages and taking actions on them, rather than reading them for a corpus or an index. Its access decision controls whether user-directed agents may operate on a site's pages and forms. As of July 2026 the only officially documented, user-agent–identifiable member of this class is Google-Agent, which Google documents as used "by agents hosted on Google infrastructure to navigate the web and perform actions upon user request," giving Project Mariner as the example [5]. Google-Agent is a user-triggered fetcher and, per the same documentation, generally ignores robots.txt; the registry accordingly records no controlling robots token for it. The agent class is where this taxonomy's boundaries are least stable, for reasons set out in section 4.3: most agent activity is not identifiable from user-agent strings at all, so the class as observed is a small and unrepresentative slice of the class as it operates.

2.5 Search

A **search crawler** maintains a traditional web-search index, of the kind that predates AI answers, which AI features now additionally draw on. Its access decision has the oldest and best-understood consequences —

inclusion in classic search results — and a newer one, since AI answer features grounded in a search index inherit that index's contents. Documented examples include Googlebot [4], bingbot, and Applebot. These tokens are long-standing, and a decision about them reaches further than a decision about any AI-specific token, which is why blocking a search crawler to limit AI exposure is a high-cost action: it removes the site from general search as well. The class is included here not because a search crawler is an AI crawler, but because an AI-answer visibility analysis is incomplete if it ignores the search index that answer features consume.

3 3. Classification requirements

A published crawler registry or census that uses this taxonomy MUST meet the following requirements. They are stated so that conformance can be checked from the published record alone.

- A record MUST classify each crawler by its documented or observed **function**, not by its **operator**. An operator that runs crawlers in several classes MUST appear as several rows, one per class, not as one row named for the operator.
- A record MUST state the **robots.txt token** that controls each crawler, and MUST record that this token MAY differ from the user-agent substring the crawler presents. Where a crawler honors no token, the record MUST say so explicitly rather than leaving the field blank.
- A record MUST distinguish "the operator documents no crawler" from "a crawler is documented or observed but its class is unclear." These are different states of knowledge — an absence of a source versus a source that underdetermines the classification — and collapsing them into a single "unknown" destroys information a reader needs.
- A record SHOULD record the **verification method** for a crawler where the operator documents one — published IP ranges, reverse-DNS conventions, or a request-authentication protocol — so that a reader can distinguish a token verifiable against the operator's infrastructure from one that can only be taken at face value.
- A record MUST **date** each classification. Vendor documentation churns: tokens are added, renamed, and re-scoped, and a class assignment is a statement about the documentation as of a stated date. A record that carries a "last reviewed" date, as the registry does, converts each classification into a checkable, perishable claim rather than a standing assertion.

A record MUST cite the source that supports each class assignment — vendor documentation for a documented crawler, and an explicit "by public observation" marker for a crawler classified without a vendor source. That distinction is itself load-bearing: a classification drawn from observation is weaker evidence than one drawn from documentation, and a reader MUST be able to tell which they are looking at.

4 4. Boundary cases and honest limits

A functional taxonomy is only as good as its behavior at the edges. Four boundary cases recur, and a record that does not handle them explicitly will misrepresent the field.

4.1 One user agent, more than one class

A single user-agent string can serve two classes, over time or across products: an operator can repurpose a token, or document one as performing more than one function, so the mapping from user agent to class is not permanently one-to-one. A record MUST treat the class as a property of the crawler at a stated date,

not as an immutable attribute of the user-agent string — one reason the dating requirement of section 3 is normative rather than advisory.

4.2 Product tokens and user-agent tokens

Some controls are not crawlers at all. Google documents Google-Extended as "a standalone product token" publishers use to manage whether Google-crawled content may be used for training future generations of its Gemini models and for grounding, and states it "doesn't have a separate HTTP request user agent string" — the crawling is done with existing Google user agents, and the token operates in a control capacity only [3]. A product token is therefore a robots.txt lever applied within other crawlers' fetches, not a distinct visitor a site sees in its logs. A record MUST represent such a token as what it is: it belongs in the taxonomy by the function it controls (training, here), but it MUST NOT be presented as a separately observable crawler, because no user agent by that name will appear. This is a specific instance of the general rule that the robots token and the user-agent substring are different fields that MAY diverge.

4.3 The agent class is undercounted by user-agent strings

The agent class cannot be measured the way the others can. Counting crawlers by user-agent string works, imperfectly, for the training, retrieval, user-fetch, and search classes, whose members present identifiable user agents. For the agent class it fails, and in one direction: it undercounts. As of July 2026, only one agent-class crawler is officially documented and user-agent–identifiable, Google-Agent [5]. Other agent products do not present a distinct agent user agent at all: agents that drive a browser — whether hosted by the operator or running locally — present as ordinary browser traffic, indistinguishable at the user-agent level from a human visitor, and any identification the operator offers runs through side channels other than the user-agent string. (OpenAI's documented ChatGPT-User token covers user-triggered page visits and GPT Actions [2]; it is not documented as an agent identifier.) A user-agent-based count of the agent class therefore reports a floor, not an estimate, and a record MUST state this plainly as a measurement limit rather than presenting its agent-class rows as a census of agent activity.

4.4 Undocumented operators

A taxonomy can only classify what is documented or observed, and several operators publish neither a crawler documentation page nor, in some cases, a stable user agent. A token can be widely reported by third parties while its operator publishes nothing about it — the Bytespider token attributed to ByteDance is observed but carries no operator documentation — and some operators publish no crawler user agent at all, crawling without an identifiable token. A record MUST classify these by public observation only, MUST mark them as such, and MUST NOT promote an observed token to the status of a documented one. Where an operator publishes nothing and presents no stable user agent, the honest entry is that its crawling is not identifiable from user-agent strings — the same limit as the agent class, from the opposite direction.

5 5. How robots.txt semantics interact with each class

Every class here interacts with robots.txt, but robots.txt is not what a site-level intuition often takes it to be. The governing specification states that its rules "are not a form of access authorization" [1]: a robots.txt entry is a request a crawler is asked to honor, not a barrier the file enforces. Whether any given crawler honors a published file is a decision made by whoever operates the crawler. A record that describes robots.txt as a control mechanism without stating this misrepresents it.

Given that, per-class control exists only to the extent that operators define per-class tokens. The training and retrieval classes are the best-provisioned: major operators define a distinct token for each, so a site can, in principle, request training exclusion and retrieval inclusion independently — the mis-block the robots note describes is a failure to name those tokens precisely, not an absence of tokens to name. The search class has long-standing, well-understood tokens. The user-fetch class is mixed: some members honor a token, while others, such as Google's user-triggered fetchers, are documented as generally ignoring robots.txt because the fetch was user-requested [5]. The agent class is at present the least controllable through robots.txt: its one documented member ignores robots.txt as a user-triggered fetcher, and its undocumented members present no token to address. A record MUST NOT imply that a robots.txt opt-out exists for a class whose operators have defined none; the correct statement is that per-token control is unavailable, and a reader should not infer from an empty token field that a control was declined rather than never offered.

6 Limitations

This taxonomy reflects vendor documentation and public observation as of its review date. Documentation churns — tokens are introduced, renamed, and re-scoped — so every classification is perishable, and the maintained record, not this convention, is the current mapping; the registry's "last reviewed" date states how current a given class assignment is.

The classes are likely to split. The agent class in particular is a single undifferentiated category today only because agent traffic is barely identifiable; as operators document distinct agent identifiers, or as authentication protocols make agent traffic separable from browser traffic, the class will differentiate, and this convention will grow a new version rather than silently reinterpret the current one. The retrieval and search classes may also blur further as answer features and search indexes converge.

Finally, classification is not verification of behavior. Sorting a crawler into a class is a statement about its documented or observed function, not a finding that the crawler behaves as documented in practice; whether a given crawler's behavior matches its class — whether a training crawler honors a training opt-out, whether a retrieval crawler respects its token — is a separate empirical question requiring independent measurement, outside the scope of a classification scheme. A taxonomy tells a reader what a crawler is for; it does not tell them what a crawler did.

REFERENCES

1. M. Koster, G. Illyes, H. Zeller, and L. Sassman, IETF. *RFC 9309: Robots Exclusion Protocol* (2022). https://www.rfc-editor.org/rfc/rfc9309.html Accessed 2026-07-09. [archived]
2. OpenAI. *Overview of OpenAI Crawlers*. https://developers.openai.com/api/docs/bots Accessed 2026-07-09. [archived]
3. Google, Crawling infrastructure documentation. *Google's common crawlers*. https://developers.google.com/crawling/docs/crawlers-fetchers/google-common-crawlers Accessed 2026-07-09. [archived]
4. Google, Google Search Central documentation. *Googlebot*. https://developers.google.com/search/docs/crawling-indexing/googlebot Accessed 2026-07-09. [archived]
5. Google, Crawling infrastructure documentation. *Google's user-triggered fetchers*. https://developers.google.com/crawling/docs/crawlers-fetchers/google-user-triggered-fetchers Accessed 2026-07-09. [archived]
6. Common Crawl Foundation. *CCBot*. https://commoncrawl.org/ccbot Accessed 2026-07-09. [archived]

HOW TO CITE

Barkhausen AI (2026). A taxonomy of AI-related crawlers. https://barkhausen.ai/conventions/crawler-taxonomy/

Barkhausen AI · Est. MMXXVI · *Every leap begins as a crackle.*