Episode Book

EPISODE #5869

Scraping AliExpress Specs: Selectors, Agents, or Neither?

PUBLISHED	11 October 2026 at 14:05 UTC
DURATION	22:52
LISTEN	myweirdprompts.com/episode/ali-express-spec-scraping-03238/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.3
SOURCES	10 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

Daniel has a home inventory system that tracks where things physically are, and he wants to add a snapshot function: save an AliExpress URL, pull the page, keep only the technical sheet. No reviews, no price. The obvious plan — define the divs, extract them — runs into a problem that has already happened: AliExpress product pages are client-side rendered, the inline params object is now an empty object, and a parser reading it returns nothing while logging a success. So what are the live options? A maintained Apify actor that returns a parsed specs array, or an agentic extraction layer like Firecrawl's JSON mode where the prompt names both the field and where on the page it lives. Both assume you can reach the page at all — and Alibaba's Baxia anti-bot returns HTTP 200 with a punishment header, so a dashboard counting 2xx responses reports a healthy pipeline while every row is empty. The extraction library is competing for second place.

TRANSCRIPT



Corn

The thing about buying small parts from AliExpress is that you don't lose the part. You lose the sheet that tells you what the part actually is.



Herman

The listing changes, the seller relists under a new ID, the whole page vanishes one day, and the only record of the thread pitch and the flow rate was in a table you scrolled past in April.



Corn

Daniel's been sitting on that problem for a while. He's got the home inventory system already, the one that records where things physically are, and what he wants to add is a snapshot function. Save the AliExpress link, pull the page, keep the part of it that reads like a technical sheet. No reviews. He doesn't even want the price.



Herman

That's the detail that makes this interesting.



Corn

He wants it targeted. Define what you extract up front, get only that back. And he's aware of the obvious failure mode there, which is that you define the divs, AliExpress redesigns the page, and your scraper is quietly extracting nothing.



Herman

So he's weighing two things. The traditional route, an Apify actor doing selector work, against an agentic extraction layer where a prompt tells the model what to pull. And then the part he flagged himself, which is that AliExpress is one of the most scraped sites on earth, so anti-bot is going to be a factor.



Corn

His ideal flow is worth keeping in view. He saves a URL against a catalog item, that triggers the runner, and if it works the PDF gets uploaded and appended to the item through his own internal API. So the extraction is one piece of a pipeline, not the whole thing.



Herman

Here's what I want to get to first, because it changes how you read his worry. He says he could define the divs he needs, but that risks breaking if AliExpress changes their front end.



Corn

That was the right fear. It's just late.



Herman

The AliExpress product page is client-side rendered now. The shell you get back, the raw HTML, is around seventy-seven kilobytes and contains no price and no SKU data. Everything is assembled in the browser after the fact.



Corn

And the piece that older tutorials lean on, the window dot run params object where the product data used to sit inline?



Herman

Empty object. It's still there in the page, it's just empty. Which is the nastiest version of this, because a parser that reads it doesn't throw an error. It returns nothing, cheerfully, and your job logs a success.



Corn

So it already broke. He was planning around a risk that has already materialized on the site he's targeting.



Herman

One of the stealth browser projects, running a fresh identity, got zero of twenty-four page loads on a flagged IP. The page either renders with Baxia's challenge in it or it doesn't render at all.



Corn

So the surface-level question, selectors or agent, is downstream of something bigger.



Herman

Considerably. And I think we should give him the real comparison anyway, because he needs it to make the call, and then we should show him why the comparison keeps collapsing. So if the old selector approach is dead on arrival, what are the two live options?



Corn

Option one. He mentioned Apify, which is the sensible place to look, because Apify's store has an AliExpress scraper that specifically returns what he wants.



Herman

The one to look at is by a builder called Goldmine, the logical underscore scrapers AliExpress scraper. It returns a specs array. Name and value pairs. That is the technical sheet, already parsed for you.



Corn

That's the exact field he described wanting.



Herman

There's an include description flag if he wants the description HTML as well, plus category path IDs, variants, shipping, store. And it handles proxies automatically, residential proxies in whatever country he's shipping to.



Corn

So the geography problem, the fact that the same product shows a different price and sometimes a different listing depending on where you're asking from, is handled inside the actor.



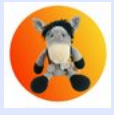
Herman

For price, yes, though he doesn't care about price. Pricing is pay per result. Three point nine nine tenths of a cent, roughly, per product on the free tier, dropping to two point nine nine on the paid tiers, so about two dollars ninety-nine per thousand products.



Corn

That's cheap enough that this is not a budget question. What's the reliability look like?



Herman

Four hundred and ninety-five total users, about eighty point nine percent of runs succeeded, four point nine eight out of five. And the FAQ on the actor page is unusually candid. It says AliExpress sometimes refuses product details to an IP address, the actor retries each product from several addresses, and a product it still cannot load keeps its listing fields with its detail fields empty.



Corn

Which is the same silent failure, just inside the black box.



Herman

Same shape. The run succeeds, the row exists, the specs field is empty.



Corn

The trade-off is that he doesn't own the maintenance. He inherits someone else's.



Herman

That's the honest version of the argument for this path. His fear about front-end changes is correct, but on this route that fear becomes the vendor's problem. The actor page says it's tested against live pages and fixed when the site changes. That's not a guarantee. It's a division of labor.



Corn

And the trigger pattern he described, run-on-save, maps cleanly. Apify fires webhooks on run events, and the only action available right now is a POST to whatever URL you specify.



Herman

So save the URL, POST to the actor, let the webhook POST back to his pipeline when the run finishes. That part is boring, which is a compliment.



Corn

Option two. The agentic layer, where the prompt does the extracting.



Herman

Firecrawl's JSON mode is the closest thing to what he described. You pass it a prompt, or a schema, or both, and it returns structured data from a single known URL. It's synchronous, it's one request, it doesn't need to navigate anything.



Corn

And the docs give the exact guidance someone building this needs. There's a line about adding location hints. The example they use is telling the model to pull flow rate in GPM from the specifications table.



Herman

That's the whole trick for a specs table. You're not just naming the field, you're naming where on the page it lives, because the model can actually read the page and find the table with that heading. Which is exactly what a CSS selector does, except it's resilient to the table moving.



Corn

There's a second piece of guidance that matters more than it sounds. Include null handling in the field descriptions, so the model doesn't guess missing values.



Herman

That's the one thing I'd tattoo on anyone building an extraction layer. A model asked for a flow rate when there isn't one will invent a plausible flow rate. You have to tell it, in the schema, that empty is an acceptable answer.



Corn

Does it handle the page attributes? The data IDs, the custom attributes?



Herman

No, and this is a v2 caveat worth knowing. HTML attributes aren't available in JSON extraction. The extraction runs on the markdown conversion of the page, so data attributes get stripped before the model ever sees them.



Corn

For a specs table that's fine. It's visible text, and visible text survives the markdown conversion.



Herman

Fine for this task, useless if he ever wanted to target something by attribute rather than by what it says.



Corn

And Firecrawl has an agent endpoint that goes further, autonomous navigation, no URL needed. Overkill here?



Herman

Their own docs say so. JSON mode on the scrape endpoint is cheaper and synchronous for one known URL. The agent is for when you don't know where the thing is.



Corn

Then there's the open-source option, Browser Use, where you describe the scraping task in plain language instead of writing selectors.



Herman

And this is where the picture gets complicated, because Browser Use's own CTO published a piece in September called The Bitter Lesson of Browser Agents. The argument is that pure LLM agents are slow, expensive, and unpredictable for high-frequency tasks.



Corn

That's a company arguing against the thing its own tool does.



Herman

It's a company following the evidence. Their answer is a product called Workflow Use. You record a workflow once, deterministically, and then it runs reliably a million times, and when the page changes underneath it, the model repairs the workflow rather than the workflow being rewritten by hand.



Corn

So record the reliable path, let the model patch it when it breaks.



Herman

That's the hybrid, and it directly answers his stated fear. He doesn't want to maintain selectors, and he doesn't want an agent burning tokens and time on every single run. Workflow Use is the middle.



Corn

The bitter lesson framing is honestly the most useful thing in this whole comparison. The industry's practitioners have already run the experiment he's proposing to run, and they concluded the same thing.



Herman

Both of those architectures assume you can actually reach the page.



Corn

That's where this gets interesting, and it's the part Daniel half-anticipated. Let's do the anti-bot layer properly, because I think it's the decisive thing and the rest is decoration.



Herman

AliExpress runs Alibaba's Baxia system. The public testing on it is fairly stark. Blocked requests come back with an HTTP two hundred status.



Corn

Two hundred. The success code.



Herman

Two hundred, with a bxpunish header and body markers. Things like rgv587 flag, x5secdata, fragments of the tmd string, FAIL SYS USER VALIDATE.



Corn

So if your dashboard counts two hundreds as successes, it's green while every row is empty.



Herman

Bright Data's line on it is the one I'd put on a wall. If a dashboard counts HTTP two-xx responses as successes, it reports a healthy pipeline while the rows are empty. That's not a scraping problem, that's a monitoring problem that hides a scraping problem.



Corn

How bad is the actual ceiling? If he builds this himself and just runs it, how far does he get?



Herman

There's public testing on the signed internal API, the mtop endpoint, the one that needs an MD5 signature built from a token, a timestamp, an app key, and the body. Even with a correct signature, from a single residential IP, one run got seven successes in sixty calls.



Corn

Seven.



Herman

Fifteen in sixty on a second run. The first block landed at request eight and request twelve respectively.



Corn

So you get maybe a dozen good pulls before Baxia decides you're a bot, and then it's over for that IP.



Herman

New sessions didn't clear it. Sixty-second waits didn't clear it. The blocked responses were exactly five hundred and forty-three bytes each, which is another tell. If you're logging response sizes, you can spot the block by the shape of it.



Corn

Right. So the choice of extraction library is competing for second place behind the choice of how you get the request to land at all.



Herman

By a wide margin. Bright Data's own conclusion is that you scrape the pages the site makes public and let your infrastructure handle IP addresses and fingerprints. Their three managed products all got through where self-run stealth browsers got zero out of twenty-four loads on a flagged IP.



Corn

And that's the honest answer to what he asked. He wanted a recommendation between two architectures. The recommendation is that he pick the unblocking layer first, then the extractor, and if he's picking a maintained actor, he's really picking a maintained unblocking layer with an extraction layer attached.



Herman

There's a smaller point in there worth pulling out, because he'd run into it. The framing of browserless.



Corn

He used that word.



Herman

The agentic tools that actually work against AliExpress all run a browser. Firecrawl, Browser Use, the Bright Data browser API. Browserless in the sense of no headless Chrome is not achievable against a client-side rendered page with Baxia in front of it. The browser is the thing doing the rendering.



Corn

The intelligence moved up a layer. The browser didn't go away.



Herman

And then there's the page economics, which I find funny. The mtop response for a product has twenty-seven top-level modules. The useful ones, the eight that carry data you'd want, are sixteen and a half kilobytes. About fifteen point nine percent of the payload.



Corn

What's the rest?



Herman

Shipping is the largest single module at thirty-four thousand bytes. Global data at twenty-three thousand. So you're paying bandwidth for shipping scaffolding you didn't ask for, on a page that renders to about three megabytes when it's fully built out.



Corn

So even a successful request is mostly freight.



Herman

And there's the surface-level scoring worth knowing. One anti-bot index rates AliExpress easy, one out of ten, at the homepage. But it flags that deep pages, profiles, listings, search, are usually harder, and product detail specifically is readily CAPTCHA-gated when the hits are rapid or the referer is missing.



Corn

Which is the whole design. Make the homepage look soft, make the deep data expensive.



Herman

So the architecture comparison, on its own terms, favors the hybrid. Targeted extraction with a maintained unblocking layer. But the anti-bot reality makes it less a preference and more the only thing that works.



Corn

There's the PDF piece too, which he kind of set aside and I don't think he should.



Herman

Neither Apify nor Firecrawl returns a PDF. They return JSON or markdown.



Corn

So the pipeline needs a render step. Markdown to PDF, somewhere in his stack, after extraction and before the append call.



Herman

Which is not hard, but it's a real component, and it's the one place where the tooling is his to own. The trigger maps onto Apify webhooks, or Firecrawl's agent-completed event if he goes that route. The extraction maps onto the actor or the JSON mode. The unblocking maps onto whoever's selling proxies. The render, the PDF, the append through his internal API, that's his.



Corn

And notably, nobody has built the thing he wants. No off-the-shelf AliExpress technical-sheet-to-PDF tool exists.



Herman

Right. The closest are general product scrapers that happen to include a specs array, and general LLM extraction endpoints. The snapshot-to-PDF-and-file-it pipeline is his to assemble.



Corn

There's one gap I want to flag honestly, because it changes what he should test first. Nobody's published whether the specs section specifically is served from that signed API or from the rendered DOM. The teardowns cover price, SKU, reviews. The specs block is not addressed.



Herman

Which matters. If specs come from the rendered DOM, he needs full rendering against Baxia every time. If specs come from the signed API, it's a lighter request, and his whole cost and rate-limit math shifts.



Corn

So the first thing he should do is open a product page with the network panel and see where the specs table actually comes from. That's an hour of work that determines the architecture more than the Apify versus agent question does.



Herman

And the specs table is the one part of the page that doesn't change much, which is exactly why it's worth keeping. The price and the promotion change hourly, the thread pitch doesn't. Which is also why caching a snapshot once per product is viable at all, and why his price-blindness is a feature rather than a limitation.



Corn

Every field he isn't scraping is a field he isn't fighting for. Price is the most volatile, most region-dependent, most aggressively protected thing on the page, and he's walked around it.



Herman

And that's the thing. The specs sheet is the one part of the page that doesn't change much, which is exactly why it's worth keeping.



Hilbert

You're both right about the sheet being the thing worth keeping. But you're both wrong about why.



Hilbert

You're scraping AliExpress because you don't trust AliExpress to still have the page. Fine. That's not a scraping problem. That's a "what happens when the catalogue goes out of print" problem, and I've watched that happen to a man whose whole system died with a single document.



Herman

Which man?



Hilbert

My uncle. Had a collection of small technical parts. Fittings, mostly, and a few hundred of them, boxed and labelled. And he had a system for it, which was index cards in a filing cabinet.



Corn

Cards, not a spreadsheet.



Hilbert

Cards. And the cards didn't say what the part was. They said the page number.



Corn

The page number of what?



Hilbert

Of the catalogue. One physical copy of an industrial supply catalogue, and it lived in his garage on a shelf above the workbench. The card gave you the part's drawer and the page where the specs were printed. That was the whole index. You found the card, you found the page, you had your thread pitch and your material and your pressure rating.



Corn

So the card was a pointer, not a record.



Hilbert

It worked beautifully. For eleven years. Then the catalogue went out of print, the supplier stopped sending new editions, and the copy in the garage was the only one he had. By the time he'd passed, that shelf was the only place those specs existed. The cards weren't wrong. They just pointed at nothing.



Herman

That's the actual argument for Daniel's PDF snapshot. Not convenience. Permanence.



Hilbert

The snapshot's the right instinct. Just not for the reason he gave. He thinks he's solving a scraper problem. He's solving a "the source will disappear" problem, and the source disappearing doesn't have to mean anti-bot. Sometimes it just means the catalogue stopped being printed.



Corn

The card's assumption was that the catalogue would always exist in a place he could reach. Same assumption as the URL.



Hilbert

Up to a point.



Herman

Can I ask what happened to the cards?



Hilbert

I have them.



Corn

You have them.



Hilbert

In a box. I've been meaning to digitise them for years. The problem is I can't find anyone who still owns that catalogue, so the page numbers don't mean anything to anyone but me, and I keep getting distracted by a dispute with my neighbour about a shared driveway.



Corn

The driveway.



Hilbert

He's been parking on my side since April. The cards are not going anywhere. The driveway is going somewhere.



Herman

The catalogue itself, did he buy it?



Hilbert

No. He won it.



Corn

He won a catalogue.



Hilbert

In a raffle. At a trade show, in a city I can't quite name. The prize was the catalogue, a leather-bound limited edition, and a lifetime supply of one specific size of O-ring, which he never used because it didn't fit any of his parts.



Herman

How many O-rings are we talking about?



Hilbert

I'd have to count. They're in the same box as the cards.



Corn

You have the O-rings.



Hilbert

I have the O-rings. And I've been reading up on AliExpress, because I'm going to list them. Which brings me to my question, which is about seller-side anti-bot measures, and I don't think either of you is prepared to answer it.



Corn

We are not.



Herman

We are not.



Hilbert

Then never mind.



Herman

The buyer-side scraping problem, and the seller-side listing problem, in the same house.



Hilbert

In the same box. That's the part nobody thinks about. The card is only as good as the thing it points at, and the thing it points at doesn't owe you anything. The catalogue goes out of print. The listing gets pulled. You keep the card and you keep the O-rings and you've got a very tidy record of something you can't look up anymore.



Corn

Which is why Daniel is right to want the sheet itself, and not a link to it.



Herman

Why the pointer-versus-record distinction is the thing I'm taking from this. The card and the URL are the same object. They're both promises that the underlying document will still be there.



Corn

An hour ago I'd have said the interesting question was Apify against the agent layer.



Herman

It's still worth doing the comparison. But the ceiling isn't the library. It's that Baxia gives you about a dozen good requests per IP before it stops answering.



Corn

Which means the choice is less "which tool" and more "whose unblocking layer do you trust, and what does it cost you per thousand." That's a procurement question wearing a technical costume.



Herman

It's a strange thing to realise about a personal project. He's building a small personal system, and the honest advice is to pay someone who's already solved the unblocking problem, rather than solve it himself for one user.



Corn

I'm not sure that's good news. The unblocking problem only gets solvable at scale, and scale means vendors, and vendors mean a handful of companies standing between you and the public web. His PDF snapshot is a way of not depending on AliExpress. But the thing that produces it depends on Bright Data or Apify or whoever, and those are fewer and fewer.



Herman

Which is the centralisation question, and it doesn't resolve cleanly. Every layer of the pipeline has somebody who owns it and somebody who rents access to it. His snapshot is a hedge, but it's a hedge purchased through a chain of other dependencies.



Corn

Still. The sheet is the thing worth keeping. He's right about that, and Hilbert's uncle is a decent argument for it.



Herman

The one concrete piece of advice I'd leave him with is the network panel question. Before he picks a stack, find out where the specs table comes from. If it's the signed API, the whole architecture is lighter and cheaper than we've been assuming, and the scraping comparison matters less than he thinks.



Corn

If it's the DOM, he's rendering through Baxia on every single product, and the unblocking layer is the project, not a component of it.



Herman

That's it.



Corn

We've kept you long enough on the driveway. Thanks to Hilbert Flumingtop, our producer, for being on the desk, and to Daniel for a prompt that turned out to be about archival more than it was about scraping. If you enjoyed this one, try episode eight, Building Your Own Whisper; episode two, Local STT For AMD GPU Owners; and episode twenty, Architectural AI. This has been My Weird Prompts. If you've enjoyed it, a review wherever you're listening helps people find the show. Send us your own prompt on Telegram at t dot me slash MWP listener bot. We'll be back soon.



Herman

Go check where your specs live.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Apify Actor `logical_scrapers/aliexpress-scraper` (Goldmine) [primary]**
   `specs` array, pricing, proxy handling, FAQ.
   https://apify.com/logical_scrapers/aliexpress-scraper

2. **Apify docs, Webhook integration [primary]**
   run-event triggers, POST-only action.
   https://docs.apify.com/platform/integrations/webhooks

3. **Firecrawl docs, JSON mode [primary]**
   Structured result (v2 API), schema/prompt extraction, HTML-attribute caveat, schema tips.
   https://firecrawl.mintlify.app/features/llm-extract

4. **Firecrawl docs, Agent (v2, Spark 2) [primary]**
   autonomous extraction, webhooks, pricing.
   https://docs.firecrawl.dev/features/agent

5. **Bright Data, How to Scrape AliExpress Data at Scale (2026; 36-min read)**
   Baxia markers, signed-API success rates, CSR shell, country discounts.
   https://brightdata.com/blog/web-data/how-to-scrape-aliexpress

6. **Gregor Zunic, The Bitter Lesson of Browser Agents, 2026-09-15**
   agent vs. deterministic workflow.
   https://browser-use.com/posts/bitter-lesson-browser-agents

7. **Scrapfly, Browser Use vs Playwright (v0.6.0, Aug 2025)**
natural-language vs selector scraping.
https://scrapfly.io/blog/posts/browser-use-vs-playwright

8. **Crawlora anti-bot index, aliexpress.com (homepage snapshot 2026-06-13)**
Alibaba slider, per-URL difficulty.
https://crawlora.net/anti-bot-index/site/aliexpress.com

9. **HN discussion of The Bitter Lesson of Browser Agents (2026-09-15).**
https://news.ycombinator.com/item?id=49713637

10. **HN, Show HN: Workflow Use**
Deterministic, self-healing browser automation (RPA 2.0) (2025-05-16).
https://news.ycombinator.com/item?id=44007065