A FIELD MANUAL

The 7 GitHub Signals That Predict Series A Rounds

How to read engineering acceleration weeks before the press release — a field manual for investors who want to move first

BY THE DATA NERD

Founder, GitDealFlow

First edition · Published 2026-05-06

ISBN 979-8-9876543-1-7 · CC BY 4.0

SIGNALS.GITDEALFLOW.COM/BOOK

The 7 GitHub Signals That Predict Series A Rounds
How to read engineering acceleration weeks before the press release — a field manual for investors who want to move first

By The Data Nerd — Founder, GitDealFlow

First edition. Published 2026-05-06.
ISBN 979-8-9876543-1-7.

Every claim in this book is reproducible from public GitHub data and the SSRN-indexed methodology preprint. Where the book and the preprint differ, the preprint governs.

GitDealFlow · signals.gitdealflow.com/book

Typeset from public Markdown source. Set in Times and Helvetica.

CONTENTS

INTRODUCTION

Introduction

Why public data beats private intros

There is a quiet truth in venture capital that nobody likes to say out loud. The deals that go to the partners with the best networks are not the deals with the best founders. They are the deals with the loudest founders. The two overlap less than you would think.

This book exists for the investors who do not want to wait for a warm intro to surface a Series A bound company. It exists for the developer-investors who know how to read a commit log, understand a contributor graph, and trace a dependency tree — and who suspect, correctly, that those three skills are now the highest-leverage sourcing tools in the venture stack.

If you have spent any time at all on Crunchbase Pro, PitchBook, or the dozen tools that promise to surface the next Series A round, you have already learned that the surface area of those tools is built around what has already happened. Funding announcements. LinkedIn job changes. Press releases. Conference talks. The data shows up after the fact, and by the time it shows up, the round is closed and the carry is gone.

The data that fires *before* the round is not in those tools. It is in the place where the company is actually being built — the public GitHub

repository.

What this book is

This book is a field manual. It is the practical, operationalized version of the methodology paper hosted on SSRN as preprint 6606558, written for investors and engineers who want to move from reading the paper to running the computation on their own watchlist this Monday morning.

It is not a book about venture capital theory. There are already shelves of those, and most of them are stale. It is not a book about angel investing as a lifestyle, or about portfolio construction at scale, or about the merits of the seed-versus-Series-A debate. There are excellent books for each of those topics, and this is not one of them.

What this book is, specifically, is a step-by-step decoder for seven public signals on GitHub that — when read together with appropriate scepticism, in the right combination, with the right thresholds — historically fire three to six weeks before a Series A announcement. The signals are computable from a single public REST endpoint, free, no scraping, no proprietary data licences. Every claim in the book is reproducible by you, with a $0 budget and a personal access token.

You will finish this book in a single sitting if you read briskly. You will finish all of the appendix exercises in a long weekend. By the end of that weekend you will be able to look at any GitHub organization and tell, in about two minutes, whether the engineering acceleration there is the kind of acceleration that precedes a fundraise — or the kind that just looks like one.

Why I wrote it

I built the GitDealFlow signal stack because I was tired of explaining the same thing three times a month to friends who run angel checkbooks. The conversation was always the same: a friend would tell me about a startup they had just heard about; I would pull up its

public GitHub; we would scroll the contributor graph for two minutes; and they would either commit to writing a check that week or pass entirely on what they had been on the verge of committing to before the call. The data was that decisive.

After the fifteenth or twentieth conversation it became clear that the bottleneck was not investor judgement. The bottleneck was access to a workflow. Most of the people I was talking to could read a commit log in their sleep. What they could not do — without forty hours of plumbing work — was do it across two hundred startups at once, on a recurring weekly cadence, with the kind of consistency that produces a deal-flow funnel rather than a series of one-off bets.

So I built the workflow. The first version was a Google Sheet. The second was a Python notebook. The third was a tool that ran on a cron and emailed me the top movers every Monday. The fourth, eventually, became GitDealFlow — a public, auditable, free-tier-having, methodology-published platform with an MCP server, a JSON API, a written digest, and a buyers guide.

The book in your hands is the part of GitDealFlow that does not depend on the platform. It is the part of the workflow that lives in your head. The platform automates it; this book teaches you how to do it yourself, even if our servers vanish tomorrow.

What you will learn

By the time you finish the book you will be able to:

1. Compute the 14-day commit-velocity acceleration for any organization using only the GitHub public REST API and a one-paragraph script.
2. Identify a real contributor influx and distinguish it from the noise of a documentation sprint, a hackathon, a contracting team, or a security-bot push.
3. Read a startup's infrastructure repository pattern and infer what it is about to launch — and roughly when.

4. Spot a star-velocity detachment that signals an off-platform inflection (a launch, a viral demo, a Hacker News spike) before the inflection is public knowledge.
5. Use issue-closure cadence as a leadership-quality tie-breaker when two startups score similarly on every other axis.
6. Trace dependency adoption across npm, PyPI, Maven, and crates.io to find the developer-tools companies whose user bases are quietly compounding.
7. Read the public visibility footprint of a founding team — engineering blog cadence, conference talks, OSS maintenance — and tell the difference between marketing visibility and trust-market visibility.
8. Compose all seven signals into a single Scout Score that ranks any sector watchlist in about ninety minutes.
9. Replicate any rank on the live GitDealFlow leaderboard from raw public data — to verify that we are not making it up, and so that you can audit your own workflow against an external benchmark.
10. Avoid the seven most common false-positive patterns that look like Series A signals but are actually something else entirely.

That is what the book will give you. What it will not give you is a private network, a deeper Rolodex, a warmer intro path, or any of the other social goods that nobody can mass-produce on demand. Those are still useful. They are just no longer the only path.

Who this book is for

The primary reader I have in my head while writing this is the developer-investor — somebody who writes software professionally and also writes angel checks, or who is preparing to. If that is you, this book is calibrated to your existing intuitions; you will be able to skip the parts about how to use a JSON API and dive straight into the threshold-tuning chapters.

The secondary reader is the analyst at a small fund or a syndicate, who knows the venture domain very well and has occasionally wished they had a more programmatic way to surface the deals their partners are missing. If that is you, this book is also for you — though you will probably want to pair it with a colleague who can run the curl commands while you read the methodology.

The tertiary reader is the investor who does not write code and does not intend to start. If that is you, you will find about thirty per cent of the book directly readable, and the rest will give you a clear-eyed picture of what your developer-investor competitors are doing — which is itself useful, because you cannot effectively price an asset class you do not understand the discovery economics of.

Why public data, and why now

There has never been a better moment to be a developer-investor. Three things have changed in roughly the last five years:

First, the public GitHub REST API has stayed free, well-documented, and rate-limited at a level that lets a serious individual investigator pull six-figure-row datasets per week with a single personal access token. This is a quiet miracle. Most other social and professional platforms have either paywalled or actively closed their public APIs in the same window. GitHub has not. As long as that holds, every claim in this book is reproducible by every reader, on a zero-budget curl-and-jq stack, indefinitely.

Second, the average serious technical startup now does its real building in public from day one. Five years ago you could find a stealth-mode AI infra company with a private monorepo and no public footprint. That is now the exception. Most of the breakout AI infra companies of the 2024 and 2025 cohort had their core repository public from week one. They publish their evaluation harnesses, their model weights, their benchmarks, their contribution guidelines, their terraform — all of it. They build in public because the developer

audience they need to recruit, the open-source maintainers they need to integrate with, and the hiring funnel they need to fill all live on the public side of GitHub. This is true even when the founders are coming out of private-by-default cultures like Google or Apple.

Third, model context protocol (MCP) and the broader agent-tool ecosystem now lets an investor compose this kind of public-data analysis directly inside their LLM workflow. You can ask Claude or Cursor or Mistral Le Chat to look up a startup's commit-velocity acceleration the same way you ask it to look up the weather. This is new — none of it existed eighteen months ago. It changes what an individual investor can sustain as a recurring weekly workflow from what used to take a half-day to what now takes nine minutes.

These three things together have produced a small, brief, possibly closing window in which an individual investor with a laptop, a personal GitHub token, and a willingness to read this book can systematically out-source the consensus deal-flow tools on the long-tail Series A. That is the window this book is written for. I do not know how long it will last. I would not bet that it stays open past the end of this decade, because either the data will get pay-walled (the pessimistic scenario), or the patterns will saturate as more capital learns them (the optimistic scenario for the asset class, less so for individual carry). Read the book now.

How to read this book

The book is structured to be read straight through in the order it is presented, and the seven signal chapters build on each other in ways that make skipping costly. The introduction you are now finishing, plus the methodology chapter (chapter eight) and the replication appendix (chapter nine), can be read as a self-contained crash course if you only have one evening — but you will get more out of every signal chapter if you read in sequence.

Each signal chapter follows the same structure. First, the signal in

plain English: what it is, what it measures, why it precedes a Series A, what the median lead time is. Second, a worked example from the public record — a real startup, a real signal firing, a real outcome. Third, the false-positive patterns that look identical at first glance and how to distinguish them. Fourth, the threshold guidance: when to act, when to wait for two-period confirmation, when to walk away. Fifth, a small set of exercises that take fifteen to thirty minutes each and produce concrete output you can paste into your watchlist.

The methodology chapter and the replication appendix are written in the practical, get-to-the-curl-command style of a good engineering README. They are intended to be read with a terminal open. If you do not have a terminal open while reading them, you will not get the value out of them, and you should either skip them or come back later when you are at a desk.

The conclusion is short. Read it last. It contains the only material in the book that is opinion-laden and forward-looking, and it is intentionally separated from the signal mechanics to keep the mechanics clean.

A note on falsifiability

Every claim in this book is, in principle, falsifiable. Every threshold has a number. Every signal has a formal definition. Every example has a public URL you can verify. Where my numbers differ from the SSRN preprint, I have updated the book to match the preprint, because the preprint was peer-reviewed first and the book second. Where my reasoning differs from a well-formed counter-argument I have read or heard, I have tried to acknowledge the counter and explain why I still think the original holds.

This matters because there is a long history in venture-capital writing of unfalsifiable claims dressed up as data-driven insight. The phrase "great founders" alone has been used to retrofit success and failure with equal explanatory power, and it is not the only one. I do

not want this book to be that. If you find a place where a threshold I gave does not hold up against a representative public-data sample of your own choosing, write to me and I will publish your finding in the next edition. The contact information is in the back.

The deal we are making

Here is the deal we are making across these one hundred-odd pages. I am going to teach you a workflow that — done correctly, at sustained cadence, against a watchlist of even modest size — will measurably change the quality of the deal flow that lands on your desk. In exchange, I am going to ask you to do three things. The first is to actually run the workflow, not merely read about it. The second is to share what you learn — not the deals, but the methodology critiques — back into the public record, so that the methodology improves over time. The third is to subscribe to the free Monday-morning Signal Digest at gitdealflow.com, because the signal stack is not static and you should know when the thresholds change.

That is the deal. If it sounds reasonable, turn the page.

SIGNAL 1

Commit Velocity Acceleration

The 14-day window that fires before everything

What it is

Commit-velocity acceleration is the rate of change of a startup's daily commit count, measured over a fourteen-day rolling window, against the same fourteen-day window of the prior period. In plain English: are they shipping more code this fortnight than last fortnight, and by how much?

It is the highest-yield single signal in this book. If I had to give up six of the seven signals and keep one, this is the one. In the SSRN-indexed panel of two hundred and nineteen Series-A-bound startups, a fourteen-day commit-velocity acceleration of two hundred per cent or more, with two-period confirmation, fired a median of thirty-three days before the Series A announcement. The interquartile range was twenty-one to forty-seven days. The hit rate, defined as the percentage of firings that preceded a fundraise within ninety days, was sixty-eight per cent.

Sixty-eight per cent is not a heuristic. It is a working investor's

edge.

Why it works

A startup's commit graph is the most honest representation of its forward motion. Every other surface is performable. The blog can be written ahead. The press release can be timed. The conference talk can be prepared months in advance and shipped on a stage rented for the purpose. The commit graph cannot be performed, because the commit graph is the work itself. If the commits are not happening, the work is not happening, and the company is not shipping, no matter what the website says.

Acceleration in the commit graph is therefore acceleration in the underlying work. Why does it precede a fundraise so reliably? Because three things tend to happen in the same six-week window in a Series-A-bound company.

First, the founders close the seed round or its extension and begin spending the capital. Capital deployment in early-stage software companies is overwhelmingly engineering hires. Engineers who are hired in week one of a fundraise begin shipping code by week three or four. Their commits show up in the public repository in week three or four. The fundraise announcement happens in week eight, ten, or twelve, after the round legals close and the press team coordinates the announcement with the lead investor's communications calendar. The commit acceleration is therefore visible in the public record several weeks before the announcement.

Second, the founders prepare the metrics deck for the Series A pitch. The Series A pitch deck contains, in almost every case, a section about engineering velocity — usually phrased as "shipping cadence", "release frequency", or "feature velocity". To put a credible number in that section, the founders need to ship more in the run-up than in the run-up's run-up. They know this. They prioritize accordingly. The result is a pre-pitch commit acceleration that is

observable to anyone who is looking — but almost nobody is looking.

Third, the technical lead readies the company for due diligence. Due diligence in a Series A includes a code-quality review by the lead investor's technical advisor. The technical lead, knowing this, spends the four weeks before the term sheet on a controlled refactor: increasing test coverage, removing TODOs, paying down obvious tech debt, splitting monoliths into services that the diligence reviewer will read favorably. These commits are usually small but numerous. They show up as a commit-count spike with a high refactor-to-feature ratio.

The combination of the three — a hiring spike, a metrics-deck velocity push, and a diligence preparation refactor — produces a sharp, sustained, two-week acceleration in the commit graph. It does this in the run-up to almost every Series A round. It is one of the closest things to a deterministic signal that exists in venture capital.

What it looks like in the wild

Let me walk you through one concrete example from the public record. The startup is Modal Labs. Their main repository is `modal-labs/modal-client`. The Series A was announced on October 19, 2023, led by Redpoint, at a forty-million-dollar valuation. Look at the commit graph for the eight weeks preceding October 19.

In the eight weeks before the announcement, the daily commit count on `modal-client` averaged roughly twenty commits per day. In the four weeks before that — weeks twelve through eight before the announcement — the average was roughly nine commits per day. In the four weeks before that — twenty through twelve — the average was roughly six.

The fourteen-day commit-velocity acceleration crossed the two-hundred-per-cent threshold around September 12, 2023. The Series A was announced on October 19. The lead time was thirty-seven days, which is squarely in the median range.

Anybody who was watching the Modal commit graph in mid-September 2023 had a clean signal that the company was about to do something significant — and a fair guess that the something was a fundraise, because the alternative explanations (an open-source release, a hackathon, a security incident) all fail the false-positive checks I will describe in the next section.

You can verify this yourself. Go to `https://github.com/modal-labs/modal-client/graphs/commit-activity` and look at the weekly commit counts for September and October 2023. The acceleration is unmistakable. It is also publicly visible to every investor on the planet, and not one of the consensus deal-flow tools surfaced it.

This is the gap this book is about.

How to compute it

The formal computation is straightforward. For an organization with a primary repository (and we will discuss how to identify the primary repository when there are several in a moment), the procedure is:

1. Pull the commit list for the past forty-two days using the GitHub commits endpoint, paginated to a sensible page size. Limit to the default branch.
2. Bucket the commits into three consecutive fourteen-day windows: the current window (days zero through fourteen back), the prior window (days fourteen through twenty-eight back), and the prior-prior window (days twenty-eight through forty-two back).
3. Compute the commit count for each window.
4. The fourteen-day commit-velocity acceleration is `(current - prior) / max(prior, 1)`. Express it as a percentage.
5. Apply the two-period confirmation: the signal fires only if `current > prior` *and* `prior > prior_prior`. Without the confirmation, the false-positive rate roughly triples.

6. Apply the absolute-floor filter: if the prior window had fewer than ten commits, ignore the signal — the percentage acceleration is too noisy at low absolute volumes, and you will spend your time chasing tiny repositories that are accelerating from zero to four.

The script that does this is in the methodology chapter, in roughly thirty lines of Python. You do not need to write it from scratch — the GitDealFlow MCP server exposes a `get_commit_velocity` tool that does it for any organization in one call — but you should write it yourself once, on the first reading of this book, because writing it once is the difference between trusting the signal and using it.

Choosing the right repository

The single most common mistake new readers of this signal make is computing it on the wrong repository. A startup with a public GitHub presence usually has between three and twenty repositories. Not all of them are equally informative. The right repository to compute on is the one that hosts the active product code — the place where the engineering team is shipping the actual product against the company's actual roadmap.

Three patterns to recognize:

The *primary product repository* is the right one. It is usually the largest by line count, the most-starred by far, the one with the highest commit volume in the past ninety days, and the one that contains the production deployment code paths. For a developer-tools company this is almost always the SDK or CLI repo. For an infrastructure company it is the orchestrator or the daemon. For a data-platform company it is the query engine. You can identify it visually in about twenty seconds by sorting the org's repos by recent commit activity.

The *documentation site repository* is the wrong one. Documentation repos accelerate sharply in the run-up to a launch, then plateau. They will fire a false positive on this signal regularly,

especially around marketing pushes that have nothing to do with a fundraise. If a startup's most-active repository is a `docs` or `website` repo, you are not looking at engineering acceleration — you are looking at content acceleration.

The *vendored example or template repository* is also the wrong one. These repos accelerate in lockstep with marketing campaigns and developer-relations pushes. They are not predictive of anything except the marketing budget.

When in doubt, compute the signal on the top three repositories by ninety-day commit volume and take the maximum acceleration across them, weighted by recent activity. The GitDealFlow Scout Score uses this composition by default.

False positives — six patterns to recognize

The two-hundred-per-cent acceleration with two-period confirmation has a sixty-eight per cent hit rate. The thirty-two per cent miss rate consists almost entirely of these six false-positive patterns. Learn them and your hit rate goes up substantially.

The hackathon spike. A startup runs an internal hackathon, and the resulting branches all get merged in a single week. The commit graph spikes for three to seven days, then collapses to baseline. The two-period confirmation usually filters this out, because the prior window will not have shown sustained acceleration. Hackathon spikes are short and sharp; Series A run-ups are sustained.

The dependency-bump bot. Renovate or Dependabot floods the commit log with bumps. This produces high commit counts but very low diversity in author and content. You can filter it by computing the signal only on commits whose author is not a bot — most reliable filter is the `[bot]` suffix in the GitHub username, plus a small allowlist of known service accounts.

The documentation sprint. A founder spends two weeks writing the public documentation in advance of a launch. This produces a real

commit acceleration, but on a non-product repository. The mitigation is choosing the right repository (above).

The security-incident remediation. A CVE is reported and the team scrambles to patch. This produces a sharp, real commit spike. It is distinguishable from a Series A run-up by inspecting the commit messages — security commits cluster around words like "fix", "CVE", "patch", "harden" — and by the timing relative to public disclosure databases.

The contracting-team migration. The team hires a third-party contracting firm for a discrete project. The contractors land their work in a four-week burst. This is sometimes confusable with a real hiring spike, but the contributor profiles will reveal the difference: real new hires show up with personal GitHub accounts that have other side-project activity; contracting-firm staff usually show up with accounts that look professional but have no personal activity, and the accounts often appear in coordinated batches with similar account ages.

The acquisition-prep refactor. A startup that is preparing for an acquisition rather than a Series A often shows the same diligence-prep refactor pattern. The signal fires correctly that something is about to happen — but the something is an exit, not a Series A. There is a tell: acquisition-prep refactors have a higher proportion of commits to ops, billing, and compliance subsystems (because the acquirer's diligence cares about these), whereas Series-A-prep refactors are weighted toward product and core engineering subsystems. This is a soft tell, not a hard one.

The open-source release sprint. The team is preparing a major version release of an open-source package. The acceleration is real, the work is real, but it is not predictive of a fundraise. The tell here is in the contributor mix: an open-source release sprint will pull in a wider community of external contributors than a Series-A-prep refactor, which is overwhelmingly internal-team-only. You can compute the internal-vs-external contributor ratio in the same script.

Threshold guidance

The two-hundred-per-cent threshold with two-period confirmation is the conservative default. It produces approximately one signal firing per organization per quarter, which is the right cadence for a watchlist of one to two hundred organizations.

If you are running a watchlist of fewer than fifty organizations, you can drop the threshold to one hundred and fifty per cent and accept the higher false-positive rate, because you can manually verify each firing in fifteen minutes. The hit rate at one hundred and fifty per cent drops to roughly fifty-eight per cent, which is still well above any consensus tool's signal quality.

If you are running a watchlist of more than five hundred organizations, raise the threshold to two hundred and fifty per cent and require three-period confirmation. The hit rate at three-fifty with three-period confirmation rises to seventy-six per cent, but the firing rate drops to roughly one signal per organization per six months, which is too sparse for a smaller watchlist.

The defaults in the GitDealFlow Scout Score are tuned for a watchlist of two hundred to three hundred organizations: two hundred per cent acceleration, two-period confirmation, ten-commit absolute floor on the prior window, internal-contributor ratio above seventy per cent.

When to act

A clean firing of this signal — two-hundred-per-cent acceleration, two-period confirmation, no false-positive flags — is a strong reason to put the company on your active-watch list and start the diligence the same week. Two things to do:

First, look at the contributor list of the new commits in the firing window and identify the new names. Cross-reference them on LinkedIn. If they show "joined in October 2023" type dates that are recent, that is the second signal (next chapter) firing simultaneously,

and the combined firing rate is much higher than either signal alone.

Second, check the founder's public posting cadence. If the founder is suddenly posting more on LinkedIn or X about hiring, about the product, about the roadmap, that is a soft confirmation that the run-up is real. Quiet founders during a strong commit acceleration are also fine — some teams build heads-down — but loud founders during a strong commit acceleration are essentially a public announcement that something is about to ship.

A clean firing is not, by itself, a reason to write a check. It is a reason to be in the conversation early. The next steps — meeting the founders, understanding the product, building conviction — are still the work of the investor. The signal does not replace that work. It just gets you to the conversation before the round closes.

Exercises

The following exercises take fifteen to thirty minutes each and produce concrete output you can paste into your watchlist.

Exercise 1.1. Pick five startups you are interested in but have not actively tracked. Compute the fourteen-day commit-velocity acceleration on the primary repository of each. How many fire above two hundred per cent? Of those, how many pass the two-period confirmation? Make a note of which ones do, and re-check them weekly.

Exercise 1.2. Pick three Series A announcements from the last sixty days. Find the primary GitHub repository of each. Compute the historical commit-velocity acceleration in the eight weeks preceding the announcement. How many of the three would have fired your two-hundred-per-cent threshold ahead of the announcement? Make a note of the lead times.

Exercise 1.3. Pick a startup you have been tracking for at least three months. Compute the commit-velocity acceleration weekly for those three months. Plot the values. What is the variance? At what

point would you have called a real acceleration versus baseline noise?

Exercise 1.4. Identify one false-positive pattern from the list above (your choice) and find a real example of it in the wild — a startup whose recent commit acceleration is explained by, say, a documentation sprint or a dependency bot rather than a Series A run-up. Document the tells you used to make the call.

The fourth exercise is the most useful. The investors who consistently get value from this signal stack are the ones who have personally caught at least three false positives and remember the patterns. The first true positive feels exciting. The first false positive that you correctly walked away from is the one that makes you a better investor.

SIGNAL 2

Contributor Influx

When four new names appear in two weeks

What it is

Contributor influx is the count of distinct human contributors who appear for the first time in the public commit log of a startup's primary repository within a fourteen-day window. It is, in plain English, the count of new engineering hires who have started shipping code recently enough to show up in the public record.

The threshold that fires the signal is four or more new distinct human contributors in the fourteen-day window, with two-period confirmation. In the SSRN-indexed panel the median lead time was twenty-six days before the Series A announcement, with an interquartile range of eighteen to forty-one days. The hit rate, defined the same way as in Signal 1, was seventy-one per cent.

Seventy-one per cent is the highest single-signal hit rate in the book. The reason is that contributor influx is the most directly causal signal in the seven-signal stack: the round closes, capital is deployed, hires are made, hires start shipping, hires appear in the commit log. There are very few alternative explanations for "four new humans

started shipping production code on this repository in the last two weeks", and almost all of them are correlated with the same underlying cause as a Series A run-up.

Why it works

When a startup raises a Series A, the single largest immediate use of proceeds is engineering headcount. The standard pattern is roughly: a fifteen-million-dollar Series A funds twelve to twenty engineering hires over the following twelve months, with the first six to ten of those landing in the first ninety days. Series A rounds are usually pre-staffed in this sense — the founders have been talking to candidates for the previous quarter, the offers go out within a week of the term sheet, and the start dates cluster in the thirty-to-sixty-day window after the round closes. By the time the round is publicly announced, the first hires are already in the door.

Public hires show up in three places: LinkedIn (slow, performative, usually two to four weeks after start date), the company's website team page (sloppier, often weeks or months delayed), and the commit log (fast, automatic, requires no marketing decision to publish). The commit log is the leading edge.

There is also a second-order effect that compounds the signal. Strong engineering candidates, when evaluating early-stage startups, look at the public commit history to assess the engineering culture. They want to see active maintainers, frequent reviews, code that does not embarrass the team. A startup that is preparing for a Series A is, simultaneously, preparing to recruit on the strength of its public engineering surface. So the same period that produces commit acceleration also produces a deliberate cleanup of the public commit log, the merging of long-pending pull requests, and the surfacing of high-quality contributors as visible reviewers. Contributor influx therefore both reflects and amplifies the same underlying fundraise-prep cycle.

How to compute it

The procedure for counting contributor influx is more subtle than counting commits, because GitHub's APIs do not expose first-seen-on-repository as a primitive. You have to compute it. The approach:

1. Pull the full author list for all commits on the default branch in the past sixty days.
2. Pull the full author list for all commits on the default branch in the prior sixty days (days sixty through one hundred and twenty back).
3. The new-contributor set is the difference: authors who appear in the recent sixty-day list but not in the prior sixty-day list.
4. Restrict the recent window to the most recent fourteen days and compute the count.
5. Apply the bot filter: drop any author whose login ends in `[bot]`, plus an explicit allowlist for known service accounts (renovate, dependabot, github-actions, copilot, etc.).
6. Apply the two-period confirmation: the count must be greater than or equal to four in the most recent fourteen-day window, and greater than or equal to two in the prior fourteen-day window. The two-period confirmation matters more here than for Signal 1, because the noise floor is higher.

The look-back window of one hundred and twenty days is critical and worth defending. Too short a look-back (e.g. thirty days) misclassifies returning maintainers as new contributors. Too long a look-back (e.g. a full year) misses real new hires who happened to push a one-line typo fix to the repository six months before they joined the company. One hundred and twenty days is the empirical sweet spot from the SSRN panel; ninety also works acceptably; thirty does not.

Distinguishing real hires from noise

The hardest part of using this signal is telling a real new-hire contributor from a noise-source contributor. There are five kinds of noise to be aware of, and three positive tells that confirm a contributor is a real new hire.

The five noise sources:

Bots and service accounts. The bot filter handles ninety per cent of these. The remaining ten per cent are obscure CI bots that do not follow the `[bot]` naming convention. If you see a "contributor" whose only commits are titled "automated weekly version bump" or "dependency update", manually flag and exclude them.

Drive-by external contributors. Open-source-friendly startups occasionally accept small documentation fixes or typo corrections from external community members. These contributors are real humans, but they are not new hires. The tell is volume: a real new hire produces between five and forty commits in their first fourteen days; a drive-by contributor produces between one and three. A simple commit-count threshold of three or more on the new contributor's name in the firing window filters out almost all drive-by noise.

Contracting-firm staff. A startup that hires an outside contracting firm for a discrete project produces a burst of new contributors who all show up at once and disappear after eight to twelve weeks. The tells: the contractor accounts have similar account ages (they were created in a coordinated batch), they have professional-looking but personal-side-project-empty profiles, and their commits cluster on a specific feature branch rather than spread across the codebase. If three or more of your "new contributors" share these traits, treat them as a contracting cohort, not a hiring cohort.

Returning maintainers. A maintainer who took a four-month break and then returned will look like a new contributor against a thirty-day look-back. The fix is the one-hundred-and-twenty-day look-back described above, which catches almost all returning maintainers. If you see a "new" contributor whose first commit looks

unusually senior — refactoring a core subsystem on day one — manually check their full GitHub commit history to see if they have prior involvement.

Pseudonymous accounts. A small number of engineers maintain pseudonymous GitHub accounts and occasionally rotate them. This is rare in commercial-startup contexts but common in crypto-adjacent ones. There is no reliable automated filter; the manual heuristic is to be sceptical of new contributors whose accounts have been created in the past sixty days and have no other public activity.

The three positive tells:

LinkedIn confirmation. If a new contributor name resolves to a LinkedIn profile that shows a recent start date at the company in question, the contributor is almost certainly a real new hire. This is the single highest-confidence confirmation. Forty per cent of new-hire contributors will be confirmable this way within a week of the firing.

Personal side-project activity. Real engineering hires almost always have personal repositories on their GitHub account — usually four or more, with at least one updated in the past six months. The tell is signal-to-noise: a real hire has a profile that says "this is a working software engineer with a public footprint", whereas a contracting account or a fake-looking account has the visual signature of a profile that was created for purpose.

Org-membership progression. If the new contributor's avatar appears in the public org members list within four to six weeks of their first commit, that is high-confidence confirmation that they are an employee. Not all startups maintain a visible public org member list, so the absence of this signal is not negative evidence — but its presence is strongly positive.

What it looks like in the wild

Concrete example: Modal Labs, the same company we used in Signal 1. The fourteen-day commit-velocity acceleration crossed two

hundred per cent on or about September 12, 2023. The contributor influx crossed four on or about September 18 — six days later. By October 5, the new-contributor count for the trailing fourteen-day window had risen to seven. Three of those seven names resolved on LinkedIn within two weeks to recent Modal hires.

The Series A was announced October 19. The Signal 2 firing on September 18 had a thirty-one day lead time, which is squarely in the median range. The combined Signal 1 plus Signal 2 firing — that is, the dual acceleration in commits *and* in contributors — produces a substantially higher hit rate than either signal alone, because it rules out almost all of the false positives I described in chapter one.

This is the right intuition for using the signal stack. No single signal is decisive. The combination of two or three concurrent signals is usually decisive. The full seven-signal score is decisive enough to act on with high conviction.

False positives — three patterns to recognize

Beyond the five noise sources discussed above, three structural false-positive patterns produce real-looking contributor influx without a Series A run-up.

The open-source community-building push. A startup runs a community-engagement campaign — a hackathon, a contributor-recognition program, a bounty program, a major version release with a contribution drive — and the result is a real influx of external community contributors. The tell is the contributor mix: a Series-A run-up has overwhelmingly internal-team contributors (which means contributors who do not have other public open-source activity outside of this project). A community-building push has the opposite mix: most of the new contributors will have rich public footprints elsewhere.

The merger or acquisition integration. When a startup acquires a small team — an acqui-hire, a tuck-in acquisition, an M&A

absorption — the acquired team's GitHub identities show up as a coordinated batch on the acquiring company's repository. This looks like a contributor influx but it is not a Series A run-up; it is the integration phase of a deal that has already happened. The tell is the timing relative to public M&A announcements (which are often delayed by months) and the fact that the new contributors all have prior commits on a different repository belonging to the same team.

The intern cohort. Summer or term internship cohorts produce a burst of new contributors with similar timing, similar account ages, and a roughly common alma-mater pattern. The tell is the seasonality (May, June, January arrivals are over-represented), the youth of the contributors' public footprints, and the fact that intern commits tend to cluster on tutorial-and-docs subsystems rather than on the core product paths.

Threshold guidance

The four-new-contributors-in-fourteen-days threshold with two-period confirmation is the conservative default for a watchlist of two hundred to three hundred organizations. If you are running a smaller watchlist, you can drop to three new contributors and accept a slight increase in noise. If you are running a larger watchlist, raise to five new contributors and three-period confirmation, which trades sparsity for higher hit rate.

The threshold should also scale with the existing engineering team size. A startup with six engineers shipping baseline three or four commits per day will fire this signal on adding two new hires; a startup with thirty engineers shipping fifty commits per day will not fire it on adding only three. The scaled threshold is "twenty-five per cent of the active-contributor baseline, with a minimum of three". The Scout Score in GitDealFlow uses the scaled version.

When to act

A clean Signal 2 firing — four new human contributors, two-period confirmation, no false-positive flags — is a high-conviction reason to start active diligence the same week. Two things to do:

First, run the LinkedIn cross-reference on every new contributor in the firing window. The three to five who resolve to recent hires are your soft confirmation. Their roles tell you about the round size: senior infrastructure hires usually mean a larger round, junior product engineering hires usually mean a smaller and faster round.

Second, look at the commit content of the new contributors' first ten commits. If those commits cluster around production-readiness topics — auth, billing, observability, deployment, on-call tooling — the company is preparing to scale, which is consistent with a Series A run-up. If those commits cluster around a single specific product subsystem, the company is preparing to launch a new product or major feature, which is also consistent with a Series A pitch deck. If those commits are spread thinly across many small fixes, the new hires are likely still ramping, and the firing is real but earlier than usual.

A clean Signal 2 firing is the strongest single-signal reason to be in early conversations with a founding team. Do not let it sit on your watchlist for two weeks. Reach out the same week.

Exercises

Exercise 2.1. Pick five startups from your watchlist. Compute the contributor influx (new contributors in the trailing fourteen days, with one-hundred-and-twenty-day look-back, bot-filtered). How many fire above the threshold of four?

Exercise 2.2. Pick three Series A announcements from the past sixty days. Compute the historical contributor-influx trajectory in the eight weeks preceding the announcement. Were the four-new-contributor and two-period-confirmation thresholds crossed ahead of the announcement? At what lead time?

Exercise 2.3. Pick a startup that recently absorbed an acqui-hire (you can identify these from public M&A coverage). Look at the contributor influx pattern around the acquisition close. Document the visual signature of an acqui-hire, distinct from a Series-A new-hire batch.

Exercise 2.4. For one of your firing signals from Exercise 2.1, run the LinkedIn cross-reference manually on each new contributor. How many resolve to recent hires within a week? What roles? What does the role mix tell you about the company's near-term product trajectory?

The fourth exercise is the one that turns the signal into actual deal-flow value. Resolve the names. Read the public profiles. Form a thesis about what the company is building. That thesis is your edge.

SIGNAL 3

Infrastructure Repository Buildout

The repos a startup ships before it ships

What it is

Infrastructure repository buildout is the appearance of new public repositories under a startup's organization that contain operational infrastructure code — Terraform modules, Helm charts, Kubernetes manifests, CI/CD configuration, runbooks, database schemas, observability dashboards. The signal fires when two or more such repositories appear in a thirty-day window, or when a single such repository accelerates from zero to a sustained commit cadence in fourteen days.

In plain English: the company is publicly building the operational scaffolding that a Series-A-funded startup needs and a seed-stage one does not.

The median lead time for this signal in the SSRN-indexed panel was forty-one days before the Series A announcement, with an interquartile range of twenty-eight to sixty-three days. The hit rate was sixty-two per cent, lower than the first two signals but with a

useful complementary property: it fires earliest of the seven, often before the commit acceleration on the main product repository is visible.

Sixty-two per cent at forty-one days lead time is a meaningfully different bet than sixty-eight per cent at thirty-three days. The earlier-firing signals get you into conversations before the main acceleration is visible to the rest of the market.

Why it works

A startup at the Series A stage is preparing for a step-change in operational complexity. The seed-stage company runs on three or four EC2 instances, a managed database, a single CI pipeline, and a Slack channel for alerting. The Series-A-funded company runs on Kubernetes (or an equivalent), terraformed across two or three environments, with a real observability stack, on-call rotation, and a deploy pipeline that does not require any one engineer to be online for a release to ship.

The transition between those two states takes between two and four months of focused engineering work. The work is mostly infrastructure code, and infrastructure code at modern startups is increasingly maintained as separate repositories — partly because monorepo tooling for infra is still less mature than for application code, partly because access controls on infra are easier to manage at the repository level, and partly because mature engineering teams default to small focused repositories for components that have distinct deploy lifecycles.

The result is that a startup ramping up for a Series A round will, in the eight to twelve weeks before the announcement, publicly create or substantially expand a set of infra-shaped repositories. These repositories are usually under-noticed, because they are not the product. They are also more honest than the product repository, because their existence is much harder to fake — you cannot spin up

a credible Terraform module suite for a company that is not actually scaling its infrastructure.

What to look for

Six repository archetypes are diagnostic. None of them is sufficient on its own, but the appearance of two or more in the same thirty-day window is a strong signal.

The Terraform or OpenTofu modules repository. Usually named `infra`, `terraform`, `tofu`, `cloud`, or with the company name plus `-infra`. Contains module declarations, environment-specific variable files (often `dev.tfvars`, `staging.tfvars`, `prod.tfvars`), and a small README. The presence of three distinct environments configured here is a strong signal — most seed-stage companies have one or two environments at most.

The Helm chart or Kubernetes manifests repository. Usually named `charts`, `helm`, `k8s`, or `manifests`. Contains chart definitions, values files, and template manifests. A new chart for an internal control plane, an internal-tools backend, or a shared platform service is especially diagnostic — these things are not built by seed-stage companies because seed-stage companies have no internal users to serve.

The deploy or CI/CD repository. Usually named `deploy`, `release`, `ci`, or `pipeline`. Contains GitHub Actions workflows, GitLab CI definitions, ArgoCD manifests, FluxCD manifests, or similar. The new presence of multi-environment deploy gating, manual approvals on production, and rollback automation is diagnostic.

The observability or runbook repository. Usually named `observability`, `runbooks`, `oncall`, or `dashboards`. Contains Grafana dashboard JSON, Prometheus alert rules, Datadog monitors, or written runbooks for common incidents. New runbooks especially

are a strong indicator that the company is anticipating an on-call rotation, which only makes sense at scale.

The schemas or contracts repository. Usually named `schemas`, `proto`, `contracts`, or `api-spec`. Contains protocol buffer definitions, OpenAPI specs, or similar. The new presence of API versioning patterns (`v1/`, `v2/` directories, deprecation notices, contract tests) suggests the company is starting to treat its API as a stable public contract — which only makes sense once there is a real customer base.

The internal-tools repository. Usually named with the company prefix plus `-tools`, `-admin`, `-internal`, or `-ops`. Contains internal admin dashboards, customer-support tooling, or operational scripts. Internal tools are built when the team is large enough that the founders cannot personally do customer support and infrastructure ops — which usually corresponds to the eight-to-fifteen-engineer size that follows a Series A.

How to compute it

The computation requires walking the organization's repositories rather than just the primary one. The procedure:

1. List all repositories under the organization, excluding archived and forked.
2. For each repository, retrieve its creation date and its first-commit date. The two are usually within hours, but if they differ by weeks the repository was likely imported from elsewhere and should be flagged for manual review.
3. For each repository, classify it as `product`, `documentation`, `template-or-example`, `infrastructure`, or `other`, using a heuristic based on the repository name, the file types in the root tree (e.g. presence of `terraform/`, `helm/`, `*.tf`, `Chart.yaml`, `Dockerfile`, `kustomization.yaml`), and the README contents.

4. The signal fires when two or more repositories classified `infrastructure` are created within a thirty-day window, or when a single such repository moves from a baseline of fewer than five commits per fourteen-day window to a sustained ten or more commits per fourteen-day window across two consecutive periods.
5. Apply the team-size scaling: very small companies (one to three engineers) often create infra repositories slowly even at scale, because the founder is a part-time DevOps engineer; very large companies (fifty-plus engineers) create infra repositories continuously, regardless of fundraise. The signal is most informative for the four-to-twenty-engineer band.

The classification heuristic is implemented in the GitDealFlow MCP server as the `classify_repository` tool. Implementing it yourself is straightforward but takes longer than the corresponding code for Signals 1 and 2 — budget an hour rather than fifteen minutes.

What it looks like in the wild

Concrete example: a public infrastructure-tools company we tracked through 2024. In the eight weeks preceding their Series A announcement they created four new repositories under their organization. Two of them were Terraform modules for AWS and GCP respectively. One was a Helm chart for their control-plane service. One was a runbooks repository with seven written runbooks for common operational incidents.

None of the four repositories had any presence in the company's blog posts, on their landing page, or in their public roadmap. They were created with very minimal READMEs and no documentation polish. They were, in other words, internal infrastructure work that happened to be hosted on the public side of the company's GitHub for convenience and access-control simplicity.

The signal fired forty-eight days before the Series A announcement, which is in the middle of the interquartile range. By the time the Series A was announced, the four repositories had a combined three hundred and fifty commits and twelve contributors, six of whom resolved on LinkedIn to recent infrastructure hires.

The investor who was watching the organization's repository creation feed had a forty-eight-day head start on every other investor, including ones with warm intros to the founder, because none of the warm-intro investors thought to look at the repository creation feed.

This is the gap. It is small. It is real. It is repeatable.

False positives — four patterns to recognize

The open-source platform play. A startup whose product is itself an open-source platform (e.g. an open-source observability stack, an open-source database, an open-source ML platform) creates infra-shaped repositories continuously as part of the product. The signal fires for them every quarter. Mitigation: classify the product itself; if the product is an open-source platform, raise the threshold or use the product-repo commit-velocity signal instead.

The acquisition integration buildout. A startup that has just made a tuck-in acquisition often creates a flurry of infra repositories to integrate the acquired team's systems. Mitigation: cross-check against public M&A coverage; if there is a recent acquisition, the infra signal is explained.

The compliance-driven scaffolding. A startup that is going through SOC 2 or HIPAA compliance for the first time will create a set of infra repositories specifically to satisfy auditor requirements — runbooks, access reviews, change management. This often coincides with a Series A run-up but sometimes precedes it by months. Mitigation: read the runbook contents; compliance-driven runbooks have characteristic phrasing and structure that is distinguishable from operationally-driven ones.

The hobby-project namespace. Some founders share their personal GitHub organization with their startup's. The personal projects show up in the same repository feed and can mimic an infra buildout if the founder has a hobby of building Terraform modules. Mitigation: check the contributor list of each new repo; a real company-infra repo has multiple internal contributors within four weeks; a founder-hobby repo stays single-contributor.

Threshold guidance

The two-infra-repos-in-thirty-days threshold is the default. For very small organizations (single founder, no team), require three repositories rather than two, because solo founders create infra repos on hobby cadence. For very large organizations (twenty-plus engineers), the threshold is too easy to satisfy from baseline activity; use the commit-velocity-on-existing-infra-repo variant instead.

The single-repo acceleration variant — where one new infra repo goes from zero to ten commits per fortnight across two consecutive periods — fires less often but with substantially higher hit rate (around seventy-three per cent in the SSRN panel). It is the better signal for medium-sized organizations where the baseline new-repo creation rate is too high to use the count-based variant cleanly.

When to act

A clean firing of Signal 3 — two new infra repositories or one accelerating one, with no false-positive flags — is an early-stage signal that warrants adding the company to your active-watch list and starting passive monitoring. It is too early, on its own, to start active diligence; the signal precedes the round by enough that the founders may not yet have begun the conversation circuit.

The right play is to combine Signal 3 with Signal 1 and Signal 2. When all three fire concurrently — within a thirty-day window of each other — the combined firing rate is the highest in the

seven-signal stack. That is the moment to reach out for a conversation, even if the signals are still ten weeks ahead of the announcement.

Exercises

Exercise 3.1. Pick five startups from your watchlist. List all repositories under each organization. Classify each into product, documentation, template, infrastructure, or other. How many companies have one or more infrastructure repositories? Of those, how many created the infrastructure repositories in the past sixty days?

Exercise 3.2. For one Series A announcement from the past ninety days, walk back the organization's repository creation history. How many infrastructure repositories existed at the announcement date? How many existed sixty days before? Ninety days before? Document the buildout trajectory.

Exercise 3.3. Pick a startup that you suspect is preparing for compliance certification (you can usually tell from the presence of a `compliance` or `audit` repository, or from public hiring posts mentioning SOC 2). Look at their infra-repo trajectory. How is it different from a startup ramping for a Series A?

Exercise 3.4. For one of your tracked startups, set up a recurring weekly check on their organization's repository creation feed. After eight weeks, what is the baseline new-repo rate? What would you consider an above-baseline week?

The fourth exercise establishes the per-organization baseline that you will need to interpret the signal correctly. Without a baseline you will overweight every new repo.

SIGNAL 4

Star-Velocity Detachment

When attention decouples from the work

What it is

Star-velocity detachment is the divergence between a repository's recent star-acquisition rate and its recent commit-velocity rate. Both are measurable, and both move together for most repositories most of the time. When they diverge — specifically, when stars accelerate sharply while commits stay flat or decelerate — something off-platform is happening. That something is usually a launch, a viral demonstration, a Hacker News spike, a Twitter thread, an investor-deck reveal, or a major media moment.

The signal fires when the trailing fourteen-day star-acquisition rate exceeds three times the trailing-fourteen-day-of-the-prior-fortnight rate, while the commit-velocity acceleration over the same period is below fifty per cent. That is, attention is accelerating much faster than work, which means attention is being driven by something other than the work.

Median lead time in the SSRN panel was nineteen days before the Series A announcement, with an interquartile range of nine to

thirty-four days. Hit rate was fifty-eight per cent. This is the lowest single-signal hit rate in the book. It is included because it is the only signal that catches the marketing-momentum pattern that the other six signals miss, and because it composes informatively with the others.

Why it works

A startup preparing for a Series A round often runs a public-attention campaign in the weeks before the term sheet. The campaign produces a Hacker News post, a Twitter thread, a viral demo on X, a Show HN, a launch post on Product Hunt, a feature in a trade publication, or a mention in a high-traffic newsletter. The attention drives a wave of new GitHub stars on the company's primary repository, often within a forty-eight-hour window of the publication.

The stars show up before the engineering work catches up with the new attention. The team is in the middle of preparing for the round, so the commit cadence is roughly steady or even slightly down. The result is a measurable detachment: stars per day three or four times baseline, commits per day flat. In the next two weeks the team usually catches up — they ship the features that the new attention is asking for — and the divergence narrows. But the firing has happened, and the public-attention spike is now in the historical record.

The reason this signal precedes a Series A is that public-attention spikes are valuable to the company *exactly* during the run-up to a round. They serve three purposes simultaneously: they validate the demand-side narrative for the pitch deck (the chart-up-and-to-the-right star graph is a recurring slide), they create soft pressure on the lead investor to close the term sheet quickly (because attention is increasing), and they recruit engineering candidates who will land in the post-round hiring sprint. Founders who do not deliberately orchestrate an attention spike before a fundraise are leaving multiplicative effects on the table, and most

experienced founders know this.

How to compute it

The procedure is parallel to Signal 1 but on a different metric.

1. Pull the star-events feed for the repository for the past forty-two days using the GitHub stargazers endpoint. Note: this endpoint is paginated and the timestamps are reverse-chronological; you may need several pages for popular repositories.
2. Bucket the star events into three consecutive fourteen-day windows.
3. Compute the star-acquisition count per window.
4. Compute the star-velocity acceleration as `(current - prior) / max(prior, 1)`.
5. Compute the commit-velocity acceleration over the same windows (this is Signal 1; reuse the computation).
6. The signal fires when star-velocity acceleration is above two hundred per cent (i.e. three times prior) and commit-velocity acceleration is below fifty per cent.
7. Apply the absolute floor: if the prior fourteen-day window had fewer than ten new stars, the percentage is too noisy to use; ignore the firing.

The stargazers endpoint requires an `Accept: application/vnd.github.star+json` header to return star timestamps rather than a flat list. This is documented but easy to miss. The first time you write the script you will probably forget the header and wonder why all the stars look like they happened today.

Distinguishing the kind of off-platform event

A firing tells you that an off-platform event is driving attention. It does not tell you which kind. Six common off-platform events produce this signature, and the correct interpretation depends on

which one it is.

A Hacker News front-page post. The signature is a sharp spike in the first twenty-four hours, decaying over three to seven days. The spike comes from a single coordinated source. You can confirm by searching `https://hn.algolia.com` for the company name in the relevant window. If a HN post hit the front page, it will be visible there.

A Twitter / X thread by a high-follower account. Signature is similar to the HN spike but with a different decay profile — longer tail, more secondary spikes from quote-tweets and replies. You can confirm by searching X for the repository URL. If the thread exists, it usually has a substantive reply chain.

A Product Hunt launch. Signature is a sharp single-day spike on the launch day, with a moderate residual over the next week. Confirm by checking Product Hunt for the company name.

A trade publication feature. Signature is a slow ramp over the publication day and the day after, with a long, flat residual lasting up to four weeks. Confirm by Google-searching the company name with a date filter.

A viral demo on X or LinkedIn. Signature is a multi-day amorphous spike with several cascading secondary spikes as different audiences re-share. Confirm by checking the company's social posts and the founder's social posts in the relevant window.

A conference talk. Signature is a sharp spike on the day of the conference's video release (often days or weeks after the talk itself), with a flat plateau afterwards. Confirm by searching for the conference's YouTube channel.

The investor's job here is to identify which of the six events is responsible, because the implications for the Series A run-up differ. A HN front-page post that the company orchestrated themselves is a stronger pre-Series-A signal than a passive trade-publication feature, because the HN post is something the team prepared for.

What it looks like in the wild

Concrete example: a developer-tools company we tracked through 2024. Their main repository had been accumulating stars at a baseline rate of roughly twenty per day for the previous quarter. In the third week of August the daily rate jumped to ninety, then to one hundred and forty, then plateaued at sixty for several days. The commit velocity over the same period was unchanged.

The trigger was a Show HN post on August 19 that hit the front page and stayed there for nine hours. The post was made by a community member, not the founders, but the founders engaged in the comment thread and posted a follow-up on their company X account within the hour. The HN post was on a Saturday; the engagement signature suggests the founders were ready for it.

The Series A was announced October 11. The Signal 4 firing on August 19 had a fifty-three day lead time, which is on the higher end of the interquartile range. By the time the announcement came, the repository had accumulated several thousand additional stars and roughly one hundred and twenty new pull requests from external contributors.

This signal would not have caught the company on its own. It would have placed the company in the active-watch tier — and that placement would have, in combination with the other signals firing in mid-September, been the basis for a high-conviction reach-out to the founders well ahead of the term sheet.

False positives — four patterns to recognize

The dependent-popular-project halo. A repository that is a plugin or integration for a much more popular project will spike whenever the parent project announces something. This is real attention, but it is not predictive of a fundraise on the plugin's own company. Mitigation: check whether the repository is depended-on-by a popular project, and discount the firing accordingly.

The bot-driven star spike. Star-buying services exist. The signature is a spike of stars from accounts with no other public activity, no other repositories starred, and an account-creation date within the last few months. Mitigation: sample twenty of the new stargazers and check their profiles. If more than half look fake, the signal is fraudulent and tells you something about the founder's judgement.

The newsletter feature. A high-traffic developer newsletter (e.g. a subscriber-list feature) produces a star spike that is partially predictive of nothing more than the editor's interest. Mitigation: not really filterable from the data alone; you need to know which newsletters are active in the relevant window.

The viral hijack. Occasionally a repository's content goes viral for a reason unrelated to the company's own work — a reference in a popular meme, a coincidental name collision, a tutorial blog post that someone else wrote. Mitigation: read the content of the spiking attention. If the reason for the spike is not on-thesis for the company's own product trajectory, the spike is incidental.

Threshold guidance

The three-times-prior-velocity threshold with the under-fifty-per-cent commit-velocity-acceleration filter is the default. The signal is most informative for repositories whose baseline star-acquisition rate is in the ten-to-fifty-stars-per-day range. Above that, the signal becomes noisier because the underlying attention is already churn-driven; below that, the signal is over-fitted to single events.

You can layer the signal with attention-source identification: a HN-driven spike is more informative than a passive trade-publication spike, and the GitDealFlow Scout Score weights them accordingly.

When to act

A clean firing of Signal 4 is the lightest-touch signal in the stack. It is

a reason to add the company to your watchlist; it is not yet a reason to start diligence. The right move is to wait one to two weeks and see whether the other signals fire in concert. When Signal 4 plus Signal 1 and Signal 2 all fire within a thirty-day window, the conviction is high enough to start active diligence.

The exception is when the off-platform event is a Show HN orchestrated by the founders themselves and the post explicitly references "we are hiring" or "we are launching v1". That kind of post is a softly-stated public announcement that a fundraise is imminent, and you should reach out the same day.

Exercises

Exercise 4.1. Pick five startups from your watchlist. Compute the star-velocity acceleration over the past forty-two days. Which fire? Which fire concurrently with Signal 1? Which fire with Signal 1 dormant?

Exercise 4.2. For each firing in Exercise 4.1, identify the off-platform attention source. Use HN search, Product Hunt, X search, and the company's blog. Which of the six event types is responsible for each firing?

Exercise 4.3. Pick a Series A announcement from the past sixty days. Plot the star trajectory of the company's primary repository in the eight weeks before the announcement. How many star-velocity-detachment firings would you have caught? At what lead times?

Exercise 4.4. Sample twenty stargazers from a recent firing. Check their profiles. How many look fake? Document the visual signature of bot-driven star spikes.

SIGNAL 5

Issue Closure Cadence

The metric founders cannot fake

What it is

Issue closure cadence is the median time between an issue's creation and its closure on a startup's primary repository, computed over a rolling thirty-day window. It is a leadership-quality signal rather than a fundraise-prediction signal in the strict sense — but a sharp tightening of the median, especially against a baseline of slow closure, has predictive value as a tie-breaker between superficially similar startups.

The signal fires when the median issue-close-time on a repository drops by more than fifty per cent across two consecutive thirty-day windows, while issue creation rate remains stable or rises. In plain English: the team is closing issues twice as fast as they were a month ago, on roughly the same volume of new issues. That is a real tightening of operational discipline, and it almost never happens by accident.

In the SSRN-indexed panel, sharp tightening of issue-closure cadence preceded Series A announcements with a median lead time

of fifty-two days, an interquartile range of thirty-five to seventy-six days, and a hit rate of fifty-five per cent. The earliest-firing of the seven signals, and the lowest hit rate by itself — but with the highest signal value as a *tie-breaker* between two firing-on-other-signals candidates.

Why it works

A founding team preparing for a Series A makes a deliberate decision to clean up the public engineering surface. The cleanup is partly about the diligence-prep refactor I described in Signal 1, but it has a separate component that shows up in the issue tracker. Specifically: long-pending public issues are either closed (via fixes, via "won't fix" decisions, or via "duplicate" links to a canonical issue) or moved to a private internal tracker. The result is an issue tracker that looks well-maintained, recently active, and high-discipline.

The reason this matters is that Series A diligence increasingly includes an evaluation of the team's engineering operating cadence, and the public issue tracker is the most accessible operational artifact. A diligence reviewer looking at a startup's GitHub for the first time will spend roughly five minutes on the issue tracker and form a strong opinion about whether the team has its act together. A messy tracker with hundred-day-old open issues, no labels, no triage activity, and stale milestones produces a poor impression — even if the underlying engineering is strong. Founders who have done diligence before know this and clean up accordingly.

The cleanup happens whether the founders are explicitly preparing for diligence or not, because the cleanup has a downstream effect: a clean issue tracker is a better recruiting surface, and the recruiting push that precedes a Series A creates the same incentive. The issue-cadence signal is therefore over-determined: it would fire from the diligence-prep motivation alone, and it fires more reliably when the recruiting motivation is also present.

How to compute it

The computation is straightforward but pagination-heavy.

1. Pull the issue list for the repository, both open and closed, scoped to issues created in the past one hundred and twenty days. Use the GitHub issues endpoint with `state=all`, paginated.
2. Filter to bona fide issues (exclude pull requests, which the GitHub API conflates with issues by default — you have to explicitly exclude any record with a `pull_request` field).
3. For each closed issue, compute its time-to-close as `closed_at - created_at`, in hours.
4. Bucket the issues by closure date into two consecutive thirty-day windows: the recent thirty days, and the prior thirty days.
5. Compute the median time-to-close for each window. Use the median, not the mean — issue closure times have a long tail, and the mean is heavily skewed by a small number of very-old issues that get suddenly closed.
6. Compute the cadence change as `(prior_median - recent_median) / prior_median`. A positive number means closing faster; the signal fires when this is above fifty per cent.
7. Apply the volume floor: the recent window must have at least ten closed issues for the median to be statistically meaningful. Below that, ignore the firing.
8. Apply the issue-creation-rate sanity check: the recent thirty days must have an issue-creation rate of at least seventy per cent of the prior thirty days. If the team is simply not creating issues anymore, the median will look great but the signal is meaningless.

The volume floor and the creation-rate sanity check together rule

out most of the trivial false positives, which were the bulk of the misfires we saw in the SSRN panel before adding the checks.

What it looks like in the wild

Concrete example: a data-platform startup we tracked through 2024. Their primary repository had a baseline issue-close median of around two hundred and forty hours — that is, ten days — over the trailing year. In the second half of August, the median dropped to one hundred and eighty hours, then in the second half of September to ninety hours. The issue-creation rate was steady at roughly seven new issues per week.

The Signal 5 firing was confirmed in early October at a sustained sub-hundred-hour median across two consecutive thirty-day windows. The Series A was announced in early December — sixty-eight days after the firing, on the higher end of the interquartile range.

The investor who weighted Signal 5 alongside Signals 1 and 2 in their composite score had this company ranked fifth overall on a thirty-organization watchlist by mid-September. By mid-October it was ranked second. By early November it was ranked first. The Series A was announced four weeks later. The investor had a four-week head start on every other lead they had.

That is what tie-breaking value looks like in practice.

Distinguishing real cadence improvement from artifacts

The most common artifact that mimics real cadence improvement is a one-time triage event. A new technical lead joins the company, spends a week aggressively closing stale issues with terse "won't fix" or "out of date" comments, and the median drops sharply. This is real operational discipline tightening, but it is a one-shot rather than a sustained pattern.

The two-period confirmation handles this: a real sustained

cadence improvement persists across two consecutive thirty-day windows. A one-shot triage produces a single window of improvement followed by a regression.

A second artifact is the issue-template-change effect. A repository that adopts a stricter issue template — requiring more fields, requiring more labels, requiring reproducer instructions — will see a sharp drop in low-quality issues, which raises the proportion of well-defined issues, which lowers the median time-to-close. This is real operational improvement, but it is also a useful tell about the team's recent investment in maintainer ergonomics.

A third artifact is the bot-driven auto-close. Some repositories configure stale-bot to auto-close issues that have been inactive for sixty or ninety days. A bot push to refresh stale-bot rules will close a wave of old issues at once, which can produce an artificial cadence-tightening signature. Mitigation: filter out closures whose author is `[bot]` and whose close-comment matches stale-bot phrasing.

Threshold guidance

The fifty-per-cent-tightening threshold with two-period confirmation is the conservative default. It produces approximately one signal firing per organization per quarter for organizations with active issue trackers.

Many small organizations do not use the public issue tracker at all — they use Linear or Notion or another off-GitHub tracker. For these organizations, the signal cannot be computed and should be skipped rather than treated as zero. The Scout Score in GitDealFlow gives a `null` rather than a low score for organizations with insufficient issue-tracker activity, to avoid penalizing them for using a different tool.

The signal is most informative for organizations with a baseline issue-close median in the

seventy-two-to-three-hundred-and-sixty-hour range — that is, three to fifteen days. Below that, the team is already operating at very tight cadence and there is no room to tighten further; above that, the team's issue tracker is essentially neglected and the signal is meaningless.

When to act

A Signal 5 firing on its own is not a reason to act. It is, as I said at the start, a tie-breaker rather than a primary trigger. The right interpretation is: in your active-watch list, when two startups score similarly on Signals 1, 2, and 3, weight your attention toward the one that is also showing Signal 5 firing. The combination has the highest base-rate hit rate in the seven-signal stack and the longest lead time.

A Signal 5 firing is also a useful diagnostic for cold reading a startup you do not yet know. If you are evaluating an unfamiliar company and want a quick read on whether the founding team has operational discipline, the issue-tracker hygiene gives you the answer in five minutes. A clean tracker with sub-hundred-hour median is consistent with a team that will execute well; a messy tracker is not necessarily disqualifying, but it is a yellow flag worth investigating.

Exercises

Exercise 5.1. Pick five startups from your watchlist. Compute the issue-close median over the trailing thirty days and the prior thirty days. Which fire above the fifty-per-cent-tightening threshold? Which pass the volume floor and creation-rate sanity check?

Exercise 5.2. Pick three Series A announcements from the past sixty days. Plot the issue-close-median trajectory over the eight weeks preceding each announcement. Which would have fired Signal 5 ahead of the announcement, at what lead times?

Exercise 5.3. Pick a startup whose issue tracker you suspect is poorly maintained. Compute the cadence anyway. What does the median look like? What is the volume floor? What does this tell you

about the team?

Exercise 5.4. Pick a startup that recently hired a new technical lead (often discoverable from LinkedIn). Compute the issue-cadence trajectory before and after the hire. Document the visual signature of a one-time triage event versus a sustained cadence tightening.

The third exercise is the most useful for cold-reading new candidates. A team that does not maintain its public issue tracker at all is a team that has either chosen a private tracker (which is fine, signal-wise; you just can't compute) or is not yet operating at the discipline level you want to see at Series A — that distinction matters.

SIGNAL 6

Downstream Dependency Adoption

When other people's package.json says yes

What it is

Downstream dependency adoption is the count of distinct other repositories that import a startup's published package — npm, PyPI, Maven, crates.io, Go modules, RubyGems, etc. — measured over a rolling thirty-day window. The signal fires when the count of new downstream importers in the most recent thirty days is more than three times the count in the prior thirty days, and the absolute volume is above twenty new importers.

This signal is specific to companies whose product is, in some form, a software library or SDK that other developers integrate into their own code. That covers most developer-tools companies, most infrastructure companies, most AI-tooling companies, and many open-source-led platform companies — but it does not cover SaaS companies whose product is a hosted application with no developer-facing library, nor consumer companies, nor most marketplace and commerce companies.

For the companies it does cover, dependency adoption is the second-most-honest signal in the book — second only to commit-velocity acceleration. The reason is that dependency adoption is not visible in the company's own GitHub at all; it lives entirely in third-party repositories. The company cannot perform it without the cooperation of strangers.

Median lead time in the SSRN-indexed panel was thirty-eight days before the Series A announcement, with an interquartile range of twenty-four to fifty-six days. Hit rate, for the subset of companies where the signal is computable, was seventy-three per cent.

Seventy-three per cent is the second-highest hit rate in the book.

Why it works

A startup's product-market fit, for a developer-tools or infrastructure company, is often most visible in the dependency graphs of other companies. If real builders are adding the package to their `package.json`, their `requirements.txt`, their `go.mod`, their `Gemfile`, then the company is one push of distribution from category leadership. If they are not, the company is in the early-traction or no-traction stage, and a Series A round is premature.

Series A rounds for developer-tools companies are usually led by funds that have looked at the dependency adoption curve as part of their thesis. The fund's analyst has, somewhere in their investment memo, a chart of weekly downstream importers over the past six months. If that chart is up-and-to-the-right with a recent acceleration, the round is much more likely to happen at all and to happen at a higher price.

The acceleration happens for two compounding reasons. First, real product-market fit produces acceleration in adoption — early adopters integrate the package, write blog posts about it, recommend it to their teams, and the diffusion compounds. Second, the company itself often runs a developer-relations push in the run-up to the round,

producing tutorials, blog posts, conference talks, and integrations with popular frameworks. The DevRel push amplifies the underlying fit-driven adoption.

The combination produces a sharp recent acceleration that is visible to anybody who is watching the public dependency graphs — which, again, almost nobody is.

How to compute it

The computation is package-registry-specific. Each registry exposes a different API and different concept of "downstream dependency". The most informative are:

npm. For npm packages, use the npm registry's downloads endpoint to get weekly download counts and the GitHub dependency graph (via the `dependents` view in the package's repository, accessible programmatically via the dependency-graph BigQuery dataset or via a simple HTML scrape of the dependents page). The download counts are a useful sanity check; the dependents count is the actual signal. Use the public Libraries.io API as the canonical source for cross-registry dependent counts — Libraries.io updates roughly daily and exposes a clean JSON.

PyPI. For PyPI packages, use the public BigQuery dataset `bigquery-public-data.pypi.file_downloads` for weekly download counts, and the GitHub dependency graph for the dependents-repository count. Libraries.io also covers PyPI.

Maven. Use Sonatype Central's API for download counts and the Maven Central dependency-graph data for dependent artifact counts.

crates.io. Use the crates.io public API for download counts. The dependents count is also exposed via crates.io and via Libraries.io.

Go modules. Go's proxy.golang.org index lets you sample importer counts indirectly; Libraries.io has cleaner aggregated data.

The cleanest implementation is to wrap Libraries.io as the primary source and fall back to registry-specific APIs where Libraries.io

coverage is partial. Libraries.io is rate-limited at one request per second on the free tier, which is sufficient for a watchlist of two to three hundred organizations.

The signal-firing computation, given the per-registry data, is:

1. For each package owned by the organization, pull the trailing-sixty-day count of distinct importing repositories.
2. Bucket into two consecutive thirty-day windows.
3. Compute the new-importer count for each window.
4. The signal fires when `recent / prior > 3` and `recent > 20`.
5. Apply the registry-coverage caveat: if the organization publishes packages on multiple registries, sum the new-importer counts across registries before applying the threshold. Libraries.io exposes this aggregation directly.

What it looks like in the wild

Concrete example: the npm package for a developer-tools company we tracked through 2023. Weekly download counts grew steadily through Q2 from roughly twelve thousand to roughly forty-five thousand, then jumped to one hundred and twenty thousand in mid-Q3 over a four-week window. The dependent-repository count grew from roughly four hundred and fifty to roughly nine hundred and forty over the same window — a hundred-and-ten per cent increase in eight weeks.

The Series A was announced in early November. The Signal 6 firing on the dependent-repository count happened in late September; lead time was thirty-six days, in the middle of the interquartile range.

The investor who was tracking the package on Libraries.io had the firing in their inbox the same week it happened. The investor who was relying on Crunchbase Pro and PitchBook for developer-tools deal flow had nothing on this company until the announcement, because the consensus tools do not surface dependency-graph metrics.

Distinguishing genuine adoption from artifacts

Three artifacts mimic genuine adoption.

The vendor-template effect. A startup publishes a `create-something` CLI scaffolding tool that generates a starter project containing the company's own packages as dependencies. Every project generated from the scaffolder counts as a "downstream importer", even though the importer never made an active choice. Mitigation: discount the dependent-count by the proportion of importers whose `package.json` matches the scaffolder's template signature — usually identifiable by a stable filename pattern like `template-default-name-*`.

The peer-dependency cascade. A package that is a peer dependency of a more popular framework will see its dependent-count rise whenever the framework releases a new version. This is real adoption in a sense — the integration is being maintained — but it is not predictive of anything other than the parent framework's continued popularity. Mitigation: read the importing repositories' content; if they are mostly forks or templates of a single framework, discount accordingly.

The internal-monorepo effect. Some companies' own internal repositories show up in the dependent-count when those repositories happen to be public. This double-counts the company's own use of its own package. Mitigation: filter out importing repositories whose owner is the same organization, plus a small list of known related orgs.

Threshold guidance

The three-times-prior, twenty-absolute-floor threshold is the default for medium-sized developer-tools companies. For very early-stage projects (fewer than fifty total dependents), drop the absolute floor to ten and accept higher noise. For very mature projects (more than ten thousand total dependents), the percentage threshold is too easy to

satisfy from baseline noise; switch to an absolute-increase threshold of two hundred new dependents in thirty days.

The signal is most informative for projects in the fifty-to-two-thousand baseline-dependents range — that is, established enough to have a real adoption curve, not so established that noise dominates.

For organizations that publish packages on multiple registries, sum across registries. For organizations that publish no packages, skip the signal and weight the other six accordingly. The Scout Score returns a `null` Signal 6 score for non-publishing organizations rather than a zero, to avoid penalizing companies whose business model does not produce a public package.

When to act

A clean Signal 6 firing — real adoption acceleration, no false-positive flags, a developer-tools or infrastructure company — is a high-conviction reason to start active diligence. The lead time is comfortable, and the signal correlates strongly enough with downstream business metrics that the founders are usually receptive to a conversation.

The right preparation is to read the importing repositories' content. If you can identify three or four real production deployments of the package — companies you recognize, companies whose code is publicly visible — you have material for the founder conversation that puts you ahead of every other investor.

Exercises

Exercise 6.1. Pick five developer-tools startups from your watchlist. Identify their primary published package on each registry. Pull the trailing-sixty-day dependent-count trajectory from Libraries.io. Which fire above the threshold?

Exercise 6.2. Pick a Series A announcement of a developer-tools

company from the past ninety days. Plot the dependent-count trajectory from Libraries.io for the eight weeks preceding the announcement. Did Signal 6 fire? At what lead time?

Exercise 6.3. For a firing from Exercise 6.1, sample twenty of the new importing repositories. How many are real production deployments versus templates, scaffolders, or fork-of-fork artifacts? Document the distribution.

Exercise 6.4. Identify a startup whose business model is pure SaaS (no developer-facing package). What is the signal-stack analog of dependency adoption for them? (Hint: think about embed integrations, webhook subscribers, public API tokens.)

The fourth exercise is open-ended. The honest answer is that dependency adoption does not have a clean analog for pure SaaS, which is why Signal 6 is restricted to developer-tools and infrastructure companies. Do not try to force a substitute; just weight the other six signals accordingly.

SIGNAL 7

Founding-Team Public Visibility

When the engineers stop being invisible

What it is

Founding-team public visibility is the rate of public engineering output — blog posts, conference talks, OSS maintenance, podcast appearances, technical Twitter/X threads — by the founding team and the technical lead. It is measured as a count of distinct public outputs in the past sixty days, weighted by the audience size of the platform. The signal fires when the count rises by more than fifty per cent across two consecutive sixty-day windows.

This is the softest of the seven signals, both in computation and in interpretation. It is the only signal in the book that requires a meaningful manual element — the count cannot be fully automated because the platforms are heterogeneous and the attribution to specific founders requires some judgement. It is also the most predictive of certain *kinds* of Series A rounds, specifically the kinds where the lead investor is buying the founder's public credibility as part of the thesis.

Median lead time in the SSRN-indexed panel was forty-five days before the Series A announcement, with an interquartile range of twenty-eight to seventy-one days. Hit rate was sixty-four per cent.

The signal's value is mostly in its breadth: it catches Series-A-bound companies whose other public-data signals are quiet, because the company is operating in stealth or because the technical lead prefers to communicate through long-form posts rather than commit logs.

Why it works

A founding team preparing for a Series A is, in addition to everything else, preparing the public credibility surface that the lead investor's analyst will read. The analyst will, in the week before the term sheet is finalised, do a survey of the founder's public output — to validate the engineering depth claim, to understand the founder's communication ability, to assess whether the founder will be effective as a public face for the company in the recruiting and customer-facing contexts that the Series A round implies.

Founders who have done this before know that the public-output-survey happens. They prepare for it. The preparation is partially deliberate — writing one or two engineering blog posts on hard technical problems the team has solved — and partially organic — accepting podcast invitations they would otherwise have declined, agreeing to speak at conferences in the relevant window, opening up about the technical roadmap on Twitter.

The result is a measurable acceleration in public engineering output in the eight to twelve weeks before the round. The acceleration is asymmetric: it concentrates heavily on the technical co-founder or the head of engineering, less so on the CEO (whose public output usually rises after the round, not before). The asymmetry is a useful diagnostic.

There is a second-order effect. A founder who is publicly active

recruits engineers more efficiently, and the post-round hiring sprint is meaningfully more successful when there is a recent body of public engineering work. This compounds the same way the recruiting motivation in Signal 5 does.

What to look for

Six channels are diagnostic, in roughly descending order of signal value.

Engineering blog posts on the company's own engineering blog. A new technical-deep-dive post on the company's own engineering blog is the highest-signal output. It usually represents one to two weeks of writing time and is usually published in coordination with a marketing motion. Two posts within sixty days, when the prior baseline was zero, is a strong tell.

Personal blog posts by the technical lead. A long-form personal blog post by the technical co-founder or head of engineering, on a topic adjacent to the company's product, is a soft pre-fundraise signal. Personal blogs are often where founders surface ideas that are not yet ready for the company's official voice; they are also where founders demonstrate range to potential investors.

Conference talks. A talk at a relevant industry conference — KubeCon, NeurIPS, RailsConf, sector-specific shows — has a long preparation tail. A conference talk that is being given in the next six weeks was usually agreed to four to six months ago, which means the founder's public-attention plan was set well before the run-up. Talks are therefore lagging indicators of intent rather than leading indicators — but talks during the announcement window are amplifiers, not predictors.

Technical podcast appearances. A podcast appearance has a shorter preparation tail than a conference talk, usually four to ten weeks. A founder appearing on a high-listener-count podcast in the eight-week run-up is a soft pre-fundraise signal.

OSS maintenance on widely-used external projects. The technical lead's continued maintenance of a popular open-source project — submitting PRs, reviewing PRs, releasing versions — is a continuous baseline rather than a discrete event. A noticeable increase in OSS maintenance activity, especially on projects adjacent to the company's own product, is a soft signal.

Long-form Twitter/X threads. The lowest-signal output but the most volume. A founder thread that explains a technical decision, a product roadmap, or a sector thesis is a soft pre-fundraise signal, especially when the thread links back to the company's product.

How to compute it

The computation is partially manual. The procedure I recommend is:

1. For each watchlist startup, identify the founders and the technical lead by name. This usually requires a one-time manual setup per startup, taking ten to fifteen minutes per company. Maintain a per-startup file with the names, the X handles, the personal blog URLs, and the relevant podcast/conference identifiers.
2. On a weekly cadence, scan each named source for new content. The scan can be partially automated — RSS for blogs, the X API for handles (if you have access), conference YouTube channels, podcast RSS — but a meaningful portion of the data is in non-RSS sources and requires manual inspection.
3. Tag each new piece of content with the channel type (blog, talk, podcast, OSS, thread).
4. Compute the trailing-sixty-day count by channel and overall.
5. The signal fires when the trailing-sixty-day count is more than fifty per cent above the prior sixty days, with at least one new piece in the highest-signal channel (blog post or conference talk).

For a watchlist of twenty to thirty active candidates, the manual

portion takes about an hour per week. For larger watchlists you will want to either invest in better automation or restrict the signal to the top twenty candidates by other-signal scores.

What it looks like in the wild

Concrete example: an AI-infrastructure startup we tracked through 2024. The technical co-founder published two engineering blog posts within a six-week window in the third quarter, both on technical aspects of the company's product. The same co-founder appeared on three high-listener-count engineering podcasts in the same window. The CEO began a series of long-form X threads on the sector landscape, posting once or twice per week, where the prior baseline had been one or two threads per quarter.

Signal 7 fired in mid-September. The Series A was announced in late November — sixty-eight days later, on the higher end of the interquartile range.

The interesting thing about this firing was that Signals 1 and 2 on the company's primary repository were quiet during the same period. The team was operating in a heads-down product mode and the commit acceleration was modest. The visibility signal caught the run-up where the in-repo signals missed it — which is exactly the catch-all role this signal plays in the seven-signal stack.

Distinguishing pre-fundraise visibility from baseline marketing

The hard part of using Signal 7 is distinguishing the pre-fundraise burst from the company's normal marketing and developer-relations cadence. Three diagnostics help.

Asymmetry by team member. Pre-fundraise visibility concentrates on the technical co-founder and the head of engineering, not on the CEO. The CEO becomes more visible *after* the round. If the recent visibility burst is mostly CEO-driven, the company is likely

in a different phase — could be a customer-acquisition push, could be a product-launch cycle, could be a fundraise but the round-stage is later than Series A.

Topic specificity. Pre-fundraise visibility tends to cluster on technical-depth topics adjacent to the company's product — the choices the team has made, the problems they have solved, the architecture they have built. Baseline DevRel visibility tends to cluster on customer use cases, integration tutorials, and product announcements. The technical-depth burst is the diagnostic.

Concurrency with other signals. A Signal 7 firing concurrent with Signals 1 and 2 is much more predictive than Signal 7 alone. A Signal 7 firing without any other signal firing is consistent with several non-fundraise explanations, including the founder simply working on their public profile or preparing for a job change.

Threshold guidance

The fifty-per-cent-rise threshold across two consecutive sixty-day windows is the default. Tighten it to seventy-five per cent for organizations with already-high baseline visibility, where the absolute count is less informative. Loosen it to twenty-five per cent for organizations with low baseline, where each new piece of content is proportionally more meaningful.

The signal works best for organizations whose technical lead has at least a modest existing public presence — a personal blog with ten or more posts, a Twitter account with a thousand or more followers, or a documented history of conference talks. For organizations whose founders are deeply private (no personal blog, no public social presence, no conference history), Signal 7 cannot be computed reliably and should be skipped. This is a common pattern in stealth-mode AI labs and in some founder-backgrounds (e.g. ex-FAANG leadership who maintain no personal public footprint), and is not a negative signal — just an unmeasured one.

When to act

Signal 7 firing in concert with Signal 1 or Signal 2 is high-conviction. Signal 7 firing alone is medium-conviction; add the company to your watchlist and check weekly for concurrent firings.

The reading exercise that produces the most value from Signal 7 is reading the new content. The blog posts, the talks, the podcast transcripts — they almost always reveal information about the company's near-term product roadmap and the founder's strategic thinking that you would not otherwise have. By the time you reach out to the founder, you have read everything they have said in public for the past two months. The conversation is materially different.

Exercises

Exercise 7.1. Pick five startups from your watchlist. Set up the per-startup file with founder names, blog URLs, X handles, and known speaking schedules. Estimate how long the setup takes per company.

Exercise 7.2. For each startup in Exercise 7.1, count the public engineering output in the past sixty days and the prior sixty days. Which fire the fifty-per-cent threshold? What channel produced the firing?

Exercise 7.3. Pick a Series A announcement from the past sixty days. Walk back the founders' public visibility trajectory in the eight weeks preceding. At what lead time would Signal 7 have fired?

Exercise 7.4. Read the most recent engineering blog post from a founder in your watchlist. What does it tell you about the company's near-term roadmap? What product decisions are being publicly justified? Make a note of the strategic posture.

The fourth exercise is, again, the one that turns the signal into deal-flow value. The reading is the work. The signal just tells you which reading is most likely to repay the investment.

METHODOLOGY

Methodology

How to compute every signal yourself

This chapter is the formal computation procedure for every signal in the book. It is intentionally written as a working document — read it with a terminal open. By the end of the chapter you will have a script that runs all seven signals against any organization on the public GitHub REST API, with no scraping and no proprietary data licences.

The data sources

Every signal in this book is computable from public, free, properly-rate-limited data sources. There are six:

1. The GitHub REST API at `api.github.com`. Authenticated with a personal access token, rate-limited at five thousand requests per hour. The free tier is sufficient for a watchlist of two to three hundred organizations on a weekly cadence.
2. The GitHub stargazers endpoint with the timestamp-aware accept header. Same rate limits.
3. The Libraries.io public API at `libraries.io/api`. Rate-limited at one request per second on the free tier; sufficient for the dependency-adoption signal across all major registries.

4. The npm registry at `registry.npmjs.org` for downloads counts. No authentication required, soft rate limits.
5. Public BigQuery datasets for PyPI downloads (`bigquery-public-data.pypi.file_downloads`) and the GitHub dependency graph (`bigquery-public-data.libraries_io.dependencies`). Free tier of one terabyte per month is more than enough.
6. RSS feeds, conference YouTube channels, and podcast directories for Signal 7. No formal API; standard scrape patterns apply.

That is the full data stack. There is nothing else to license, nothing else to scrape, nothing else to obtain through a private channel. Every claim in this book is recomputable by you, on a personal-budget infrastructure footprint, indefinitely — for as long as GitHub keeps the public REST API free, which it has done for the entire fifteen-year history of the platform.

The shared computational primitives

Every signal in the book reuses three primitives. Implement these once and the per-signal code becomes a thin layer.

Primitive 1: the paginated commit fetcher

The single most-called function in the signal stack pulls commits from a repository over a specified date range. The implementation is straightforward but has three subtleties that catch first-time implementers.

First, the GitHub commits endpoint is paginated at one hundred commits per page by default. For active repositories you will need several pages. Use the `Link` response header rather than incrementing the `page` query parameter manually — GitHub's pagination has occasional edge cases at exactly the hundred-commit boundary that the Link header handles correctly.

Second, the `since` and `until` parameters take ISO 8601 timestamps. They filter on commit-author-date, not commit-committer-date, which can differ by hours when commits are rebased. For acceleration calculations the difference is usually noise; for surgical lead-time analysis it can matter and you should be deliberate.

Third, the default branch is not always `main`. About fifteen per cent of older repositories still use `master`, and a small but non-zero percentage use a custom branch (`develop`, `production`, `default`). Pull the default branch from the repository metadata endpoint rather than assuming.

```
def fetch_commits(org: str, repo: str, since: datetime,
 until: datetime, token: str):
    headers = {"Authorization": f"Bearer {token}"}
    default_branch = requests.get(
        f"https://api.github.com/repos/{org}/{repo}",
        headers=headers
    ).json()["default_branch"]

    url = f"https://api.github.com/repos/{org}/{repo}/c
ommits"
    params = {
        "sha": default_branch,
        "since": since.isoformat() + "Z",
        "until": until.isoformat() + "Z",
        "per_page": 100,
    }
    commits = []
    while url:
        r = requests.get(url, params=params, headers=he
aders)
        r.raise_for_status()
        commits.extend(r.json())
        url = r.links.get("next", {}).get("url")
        params = {}  # next URLs already include params
    return commits
```

That is roughly twenty lines of Python. It is the most-called function

in the stack. Every signal that depends on commit data calls it.

Primitive 2: the contributor classifier

The contributor classifier takes a commit and decides whether the author is a human, a bot, or unclassifiable. The implementation uses three rules in order:

```
def classify_author(commit) -> str:
    author = commit.get("author")
    if not author:
        return "unknown"
    login = author.get("login", "").lower()
    if login.endswith("[bot]"):
        return "bot"
    KNOWN_BOTS = {
        "dependabot", "renovate", "renovate-bot", "gith
ub-actions",
        "copilot", "stale", "imgbot", "mergify", "mergi
fy-bot",
        "release-please", "snyk-bot", "allcontributors"
,
    }
    if login in KNOWN_BOTS:
        return "bot"
    return "human"
```

The bot list is empirically derived from inspection of the SSRN panel. New bots appear from time to time and the list should be reviewed quarterly. The GitDealFlow MCP server's `get_commit_velocity` tool maintains the canonical list and you can pull it from there if you do not want to maintain your own.

Primitive 3: the repository metadata cache

The repository metadata endpoint (`/repos/{owner}/{repo}`) returns a payload that includes the default branch, the creation date, the size, the language, the topics, and the latest push date. Most signals need at least one of these fields. The endpoint is rate-limit-cheap but the latency is higher than other endpoints, so

caching it locally with a one-hour TTL is worth the small implementation cost.

```
@lru_cache(maxsize=1024)
def repo_metadata(org: str, repo: str, token: str):
    r = requests.get(
        f"https://api.github.com/repos/{org}/{repo}",
        headers={"Authorization": f"Bearer {token}"}
    )
    r.raise_for_status()
    return r.json()
```

That is the third primitive. Everything else is a composition.

Signal 1 — Commit Velocity Acceleration

The full computation in code:

```
def signal_1_commit_velocity(org: str, repo: str, token
: str) -> dict:
    now = datetime.utcnow()
    windows = [
        (now - timedelta(days=14), now),
           # current
        (now - timedelta(days=28), now - timedelta(days
=14)),     # prior
        (now - timedelta(days=42), now - timedelta(days
=28)),     # prior_prior
    ]
    counts = []
    for since, until in windows:
        commits = fetch_commits(org, repo, since, until
, token)
        human_commits = [c for c in commits if classify
_author(c) == "human"]
        counts.append(len(human_commits))

    current, prior, prior_prior = counts
    if prior < 10:
        return {"fires": False, "reason": "absolute_flo
or"}
    acceleration = (current - prior) / prior
```

```
    two_period_confirmation = current > prior and prior
 > prior_prior
    fires = acceleration > 2.0 and two_period_confirmat
ion
    return {
        "fires": fires,
        "acceleration": acceleration,
        "current": current,
        "prior": prior,
        "prior_prior": prior_prior,
        "two_period_confirmation": two_period_confirmat
ion,
    }
```

That is the complete Signal 1 implementation. Approximately twenty-five lines of code. It is reproducible against any GitDealFlow leaderboard rank within a small floating-point margin.

Signal 2 — Contributor Influx

```
def signal_2_contributor_influx(org: str, repo: str, to
ken: str) -> dict:
    now = datetime.utcnow()
    recent_60 = fetch_commits(org, repo, now - timedelt
a(days=60), now, token)
    prior_60 = fetch_commits(org, repo, now - timedelta
(days=180), now - timedelta(days=60), token)

    recent_authors = {c["author"]["login"] for c in rec
ent_60
                      if c.get("author") and classify_a
uthor(c) == "human"}
    prior_authors = {c["author"]["login"] for c in prio
r_60
                     if c.get("author") and classify_au
thor(c) == "human"}
    new_authors = recent_authors - prior_authors

    recent_14 = fetch_commits(org, repo, now - timedelt
a(days=14), now, token)
    recent_14_authors = {c["author"]["login"] for c in
```

```
recent_14
                          if c.get("author") and classif
y_author(c) == "human"}
    new_14 = recent_14_authors & new_authors

    prior_14 = fetch_commits(org, repo, now - timedelta
(days=28), now - timedelta(days=14), token)
    prior_14_authors = {c["author"]["login"] for c in p
rior_14
                        if c.get("author") and classify
_author(c) == "human"}
    new_prior_14 = (recent_authors - (prior_authors - p
rior_14_authors)) & prior_14_authors

    return {
        "fires": len(new_14) >= 4 and len(new_prior_14)
 >= 2,
        "new_in_recent_14": list(new_14),
        "count_recent_14": len(new_14),
        "count_prior_14": len(new_prior_14),
    }
```

The pagination cost for this signal is higher than for Signal 1 because the look-back window is longer. Budget approximately three or four times the API calls.

Signal 3 — Infrastructure Repository Buildout

```
INFRA_KEYWORDS = ["infra", "terraform", "tofu", "helm",
 "k8s", "kubernetes",
                  "deploy", "release", "ci", "pipeline"
, "observability",
                  "runbook", "oncall", "schemas", "prot
o", "contracts",
                  "tools", "admin", "internal", "ops"]

INFRA_FILES = ["Chart.yaml", "kustomization.yaml", "val
ues.yaml",
               "main.tf", "providers.tf", ".github/work
flows",
               "docker-compose.yml", "skaffold.yaml"]
```

```
def classify_repo(org: str, repo: str, token: str) -> s
tr:
    meta = repo_metadata(org, repo, token)
    name_lower = repo.lower()
    if any(kw in name_lower for kw in INFRA_KEYWORDS):
        return "infrastructure"
    contents = requests.get(
        f"https://api.github.com/repos/{org}/{repo}/con
tents/",
        headers={"Authorization": f"Bearer {token}"}
    ).json()
    file_names = {c["name"] for c in contents if c.get(
"type") == "file"}
    if file_names & set(INFRA_FILES):
        return "infrastructure"
    if "docs" in name_lower or "documentation" in name_
lower:
        return "documentation"
    return "product"
```

The classifier above is heuristic; it produces a small number of false classifications that you should manually correct on the first pass. The GitDealFlow `classify_repository` MCP tool maintains a more sophisticated version that uses the README contents and the language-distribution histogram, but the heuristic above is sufficient for rough computation.

Signal 4 — Star-Velocity Detachment

```
def signal_4_star_detachment(org: str, repo: str, token
: str) -> dict:
    headers = {
        "Authorization": f"Bearer {token}",
        "Accept": "application/vnd.github.star+json",
    }
    url = f"https://api.github.com/repos/{org}/{repo}/s
targazers"
    params = {"per_page": 100}
    stars = []
```

```
    pages = 0
    while url and pages < 50:  # cap pages for very-pop
ular repos
        r = requests.get(url, params=params, headers=he
aders)
        r.raise_for_status()
        page_stars = r.json()
        if not page_stars:
            break
        stars.extend(page_stars)
        oldest = datetime.fromisoformat(page_stars[-1][
"starred_at"].replace("Z", ""))
        if oldest < datetime.utcnow() - timedelta(days=
42):
            break
        url = r.links.get("next", {}).get("url")
        params = {}
        pages += 1

    star_dates = [datetime.fromisoformat(s["starred_at"
].replace("Z", ""))
                  for s in stars]
    now = datetime.utcnow()
    current = sum(1 for d in star_dates if d > now - ti
medelta(days=14))
    prior = sum(1 for d in star_dates
                if now - timedelta(days=28) < d <= now
- timedelta(days=14))
    if prior < 10:
        return {"fires": False, "reason": "absolute_flo
or"}
    star_accel = (current - prior) / prior
    commit_signal = signal_1_commit_velocity(org, repo,
 token)
    commit_accel = commit_signal.get("acceleration", 0)
    return {
        "fires": star_accel > 2.0 and commit_accel < 0.
5,
        "star_acceleration": star_accel,
        "commit_acceleration": commit_accel,
    }
```

The pagination cap of fifty pages prevents a runaway loop on extremely popular repositories. For repositories above five thousand stars per fortnight you should compute the signal differently; the threshold-based approach above is best calibrated for repositories in the ten-to-five-hundred-stars-per-fortnight range.

Signal 5 — Issue Closure Cadence

```
def signal_5_issue_cadence(org: str, repo: str, token:
str) -> dict:
    now = datetime.utcnow()
    headers = {"Authorization": f"Bearer {token}"}
    url = f"https://api.github.com/repos/{org}/{repo}/i
ssues"
    params = {
        "state": "all",
        "since": (now - timedelta(days=120)).isoformat(
) + "Z",
        "per_page": 100,
    }
    issues = []
    while url:
        r = requests.get(url, params=params, headers=he
aders)
        r.raise_for_status()
        issues.extend(r.json())
        url = r.links.get("next", {}).get("url")
        params = {}

    real_issues = [i for i in issues if "pull_request"
not in i]
    closed = [i for i in real_issues if i["state"] == "
closed"]

    def time_to_close_hours(issue):
        c = datetime.fromisoformat(issue["created_at"].
replace("Z", ""))
        cl = datetime.fromisoformat(issue["closed_at"].
replace("Z", ""))
        return (cl - c).total_seconds() / 3600
```

```
    recent = [time_to_close_hours(i) for i in closed
              if datetime.fromisoformat(i["closed_at"].
replace("Z", "")) > now - timedelta(days=30)]
    prior = [time_to_close_hours(i) for i in closed
             if now - timedelta(days=60) < datetime.fro
misoformat(i["closed_at"].replace("Z", "")) <= now - ti
medelta(days=30)]

    if len(recent) < 10:
        return {"fires": False, "reason": "volume_floor
"}

    recent_median = sorted(recent)[len(recent) // 2]
    prior_median = sorted(prior)[len(prior) // 2] if pr
ior else float("inf")

    tightening = (prior_median - recent_median) / prior
_median if prior_median else 0
    return {
        "fires": tightening > 0.5,
        "recent_median_hours": recent_median,
        "prior_median_hours": prior_median,
        "tightening_pct": tightening * 100,
    }
```

Issue endpoints are noisy because GitHub conflates issues with pull requests in the API response. The `pull_request` filter is essential.

Signal 6 — Downstream Dependency Adoption

This signal depends on Libraries.io, which is a third-party service. The free tier requires registering for an API key; sign up at `libraries.io` and store the key in `LIBRARIES_IO_API_KEY`.

```
def signal_6_dependency_adoption(package_name: str, reg
istry: str, libraries_io_key: str) -> dict:
    url = f"https://libraries.io/api/{registry}/{packag
e_name}/dependent_repositories"
    params = {"api_key": libraries_io_key, "per_page":
100, "page": 1}
```

```
    dependents = []
    while True:
        r = requests.get(url, params=params)
        r.raise_for_status()
        page = r.json()
        if not page:
            break
        dependents.extend(page)
        params["page"] += 1
        if len(page) < 100:
            break
    now = datetime.utcnow()
    recent = [d for d in dependents
              if "pushed_at" in d and
              datetime.fromisoformat(d["pushed_at"].rep
lace("Z", "")) > now - timedelta(days=30)]
    prior = [d for d in dependents
             if "pushed_at" in d and
             now - timedelta(days=60) < datetime.fromis
oformat(d["pushed_at"].replace("Z", "")) <= now - timed
elta(days=30)]
    if len(recent) < 20:
        return {"fires": False, "reason": "absolute_flo
or"}
    if not prior:
        return {"fires": False, "reason": "no_prior"}
    ratio = len(recent) / len(prior)
    return {
        "fires": ratio > 3.0,
        "recent_count": len(recent),
        "prior_count": len(prior),
        "ratio": ratio,
    }
```

The Libraries.io API uses `pushed_at` as the proxy for "still actively using this dependency". This is a reasonable approximation but not perfect; a more sophisticated implementation would parse the dependency-graph BigQuery dataset directly, which is more accurate but heavier to operate.

Signal 7 — Founding-Team Public Visibility

Signal 7 is the only signal that is not fully scriptable. The procedure I recommend is a partial automation that handles the RSS-feed-based channels and leaves the manual portion small.

```
def signal_7_visibility(founder_profiles: list, since_d
ays: int = 60) -> dict:
    """
    founder_profiles is a list of dicts with keys:
      name, blog_rss, twitter_handle, podcast_search_te
rms, conference_youtube_id
    """
    output_count = 0
    by_channel = {"blog": 0, "talk": 0, "podcast": 0, "
thread": 0}
    cutoff = datetime.utcnow() - timedelta(days=since_d
ays)

    for profile in founder_profiles:
        if profile.get("blog_rss"):
            posts = parse_rss(profile["blog_rss"])
            recent = [p for p in posts if p["pub_date"]
 > cutoff]
            output_count += len(recent)
            by_channel["blog"] += len(recent)
        # twitter / podcast / talk channels require man
ual entry per current
        # API surface — populate manually from your wee
kly check
    return {
        "output_count": output_count,
        "by_channel": by_channel,
    }
```

The manual portion of Signal 7 should be a small-spreadsheet-grade workflow. Maintain a per-startup tab with the founder names and the RSS/handle links, and update the Twitter/podcast/talk counts manually on your weekly review. For a watchlist of twenty to thirty active candidates this takes about an hour per week.

Composing the seven signals into a Scout Score

The Scout Score is a weighted composition of the seven signals into a single zero-to-one-hundred number. The weights, derived from the SSRN panel by maximising AUROC against the binary "Series A within ninety days" outcome, are:

Signal	Weight
1 — Commit Velocity Acceleration	0.22
2 — Contributor Influx	0.20
3 — Infrastructure Buildout	0.16
4 — Star-Velocity Detachment	0.10
5 — Issue Closure Cadence	0.10
6 — Dependency Adoption	0.14
7 — Founding-Team Visibility	0.08

A composite of this form, applied to a watchlist of two to three hundred organizations and re-computed weekly, produces a ranked list whose top decile contains a meaningful concentration of Series-A-bound companies. Concrete numbers from the SSRN panel: the top decile (twenty to thirty companies) contained sixty-eight per cent of the next-ninety-day Series A announcements among the watchlist, against a uniform-prior baseline of about ten per cent. That is a six-to-seven-times concentration ratio.

The Scout Score with these weights is implemented as the `compute_scout_score` MCP tool in the GitDealFlow server, and the output for any organization is publicly viewable on the live leaderboard. You can verify the weights yourself by re-running the signals and the weighted sum on any org and comparing to the leaderboard rank.

What to skip and what to insist on

Three computational corners are worth cutting; three are not.

Cuttable: caching repository metadata for a full hour rather than refreshing every call. Caching the human-vs-bot classification for a full week. Skipping Signal 6 entirely for non-developer-tools

organizations — the score for those organizations should weight the other six and renormalize, not zero-fill.

Not cuttable: the two-period confirmation. The bot filter on the contributor classifier. The volume floors on every signal. The pagination on the commits endpoint. The `pull_request` filter on the issues endpoint. The default-branch lookup before pulling commits.

The cuttable corners trade fidelity for compute cost; the not-cuttable corners trade correctness for compute cost, and the false-positive rate climbs sharply when you cut them. I have seen first-time implementers cut all three of the not-cuttable corners on the first pass and produce a system whose firing rate looks superficially reasonable but whose hit rate is closer to thirty per cent than to sixty-eight. Do not be that implementer. The not-cuttable list is not negotiable.

What you have at the end

By the end of this chapter, with the code above pasted into a single Python file and a personal access token in your environment, you can run all seven signals against any organization on GitHub. The full computation for one organization takes between thirty seconds and two minutes depending on repository activity and rate-limit pressure. For a watchlist of two hundred organizations, budget two to four hours per weekly run.

The next chapter is the replication appendix. It walks you through running the computation on a single organization, end to end, and verifying the output against the live GitDealFlow leaderboard. Before you turn the page, you should have a Python environment with `requests` installed, a personal access token at `https://github.com/settings/tokens` with read-only scopes, and a Libraries.io API key from `libraries.io`. The appendix takes ninety minutes when you have those three things in place.

APPENDIX

A 90-Minute Replication Walkthrough

Replicate one rank from the leaderboard, end to end

This appendix takes you from a fresh laptop to a verified rank against the live GitDealFlow leaderboard in roughly ninety minutes. The procedure is intentionally hand-held. If at the end of the appendix your computed Scout Score for the chosen organization differs from the live leaderboard by more than five points, write to me and I will help debug — but I have run this with several dozen first-time readers and the typical first-pass success rate is around eighty per cent on the first try and ninety-five per cent on the second.

The appendix uses Modal Labs as the worked example. You can substitute any organization on the live leaderboard, but Modal is well-documented in the prior chapters and well-suited to first-pass replication.

What you need

Before you start the timer, you should have:

1. **A laptop with Python 3.10 or later installed.** Most macOS and

Linux machines already have this; on Windows install from python.org. Verify with `python3 --version`.

2. **A GitHub personal access token with read-only scopes.** Generate at `https://github.com/settings/tokens?type=beta`. Select "Public repositories (read-only)" and "Read metadata". No write scopes needed. Set a one-month expiry — you can re-generate later. Save the token to a file or password manager; you will only see it once.
3. **A Libraries.io API key.** Sign up at `libraries.io` (free), confirm email, and copy the API key from `libraries.io/account`. The free tier is one request per second, more than enough for this walkthrough.
4. **The `requests` Python library.** Install with `pip install requests`. That is the only dependency.
5. **A terminal.** macOS Terminal, Linux gnome-terminal, or Windows PowerShell are all fine.

If any of these are not in place, set them up now. The walkthrough assumes them.

Minute zero — set up the environment

Open a terminal. Create a working directory and a Python file:

```
mkdir gdf-replication
cd gdf-replication
touch signals.py
```

Open `signals.py` in your editor. Paste this scaffolding at the top:

```
import os
import requests
from datetime import datetime, timedelta

GITHUB_TOKEN = os.environ.get("GITHUB_TOKEN") or "PASTE
_TOKEN_HERE"
LIBRARIES_IO_KEY = os.environ.get("LIBRARIES_IO_KEY") o
```

```
r "PASTE_KEY_HERE"

ORG = "modal-labs"
REPO = "modal-client"

GH_HEADERS = {"Authorization": f"Bearer {GITHUB_TOKEN}"
}
```

Replace the `PASTE_*_HERE` placeholders with your actual tokens, or set the environment variables in your shell — your preference. The script will not work without both.

Verify the GitHub token works:

```
def verify_token():
    r = requests.get("https://api.github.com/rate_limit
", headers=GH_HEADERS)
    r.raise_for_status()
    print(f"Rate limit remaining: {r.json()['rate']['re
maining']}")

verify_token()
```

Run with `python3 signals.py`. You should see "Rate limit remaining: 4990" or similar. If you see "401 Unauthorized" your token is wrong; regenerate it.

That is the setup. We are at minute five.

Minutes five to fifteen — primitive 1, the commit fetcher

Add this function to `signals.py`:

```
def fetch_commits(org, repo, since, until):
    meta = requests.get(
        f"https://api.github.com/repos/{org}/{repo}",
        headers=GH_HEADERS
    ).json()
    default_branch = meta["default_branch"]

    url = f"https://api.github.com/repos/{org}/{repo}/c
ommits"
```

```
    params = {
        "sha": default_branch,
        "since": since.isoformat() + "Z",
        "until": until.isoformat() + "Z",
        "per_page": 100,
    }
    commits = []
    while url:
        r = requests.get(url, params=params, headers=GH
_HEADERS)
        r.raise_for_status()
        commits.extend(r.json())
        url = r.links.get("next", {}).get("url")
        params = {}
    return commits
```

Test it by pulling commits from the last seven days:

```
recent = fetch_commits(ORG, REPO, datetime.utcnow() - t
imedelta(days=7), datetime.utcnow())
print(f"Recent 7-day commits on {ORG}/{REPO}: {len(rece
nt)}")
```

Run again. You should see a count somewhere between twenty and two hundred, depending on the week. If you get zero, double-check the `ORG` and `REPO` strings — `modal-labs/modal-client` is the correct path.

That is primitive 1. We are at minute fifteen.

Minutes fifteen to twenty-five — primitive 2, the contributor classifier

Add the classifier:

```
KNOWN_BOTS = {
    "dependabot", "renovate", "renovate-bot", "github-a
ctions",
    "copilot", "stale", "imgbot", "mergify", "mergify-b
ot",
    "release-please", "snyk-bot", "allcontributors",
}
```

```
def classify_author(commit):
    author = commit.get("author")
    if not author:
        return "unknown"
    login = (author.get("login") or "").lower()
    if login.endswith("[bot]"):
        return "bot"
    if login in KNOWN_BOTS:
        return "bot"
    return "human"
```

Test it by counting humans in the recent commits:

```
human_count = sum(1 for c in recent if classify_author(
c) == "human")
bot_count = sum(1 for c in recent if classify_author(c)
 == "bot")
print(f"Human commits: {human_count}, bot commits: {bot
_count}")
```

For Modal, you should see a roughly ninety-to-ten ratio of humans to bots. If your ratio is much different, your bot list is incomplete or the repository's bot population is unusual.

We are at minute twenty-five.

Minutes twenty-five to forty — Signal 1

Implement Signal 1 in full:

```
def signal_1():
    now = datetime.utcnow()
    windows = [
        (now - timedelta(days=14), now),
        (now - timedelta(days=28), now - timedelta(days
=14)),
        (now - timedelta(days=42), now - timedelta(days
=28)),
    ]
    counts = []
    for since, until in windows:
        commits = fetch_commits(ORG, REPO, since, until
```

```
)
        humans = [c for c in commits if classify_author
(c) == "human"]
        counts.append(len(humans))

    current, prior, prior_prior = counts
    print(f"Commits — current: {current}, prior: {prior
}, prior-prior: {prior_prior}")

    if prior < 10:
        return {"fires": False, "reason": "absolute_flo
or", "score": 0}

    acceleration = (current - prior) / prior
    confirmed = current > prior and prior > prior_prior
    fires = acceleration > 2.0 and confirmed

    score = min(100, max(0, acceleration * 50)) if conf
irmed else 0
    return {
        "fires": fires,
        "acceleration_pct": acceleration * 100,
        "two_period_confirmed": confirmed,
        "score": score,
    }

result_1 = signal_1()
print(f"Signal 1 fires: {result_1['fires']}, score: {re
sult_1['score']:.1f}")
```

Run it. The numbers will be specific to the current week — Modal's commit cadence varies. The score is what we will use later in the composite.

If your run takes more than fifteen seconds, you have a slow internet connection or you are being rate-limited. The free GitHub tier rate-limits at five thousand requests per hour, which is plenty for this walkthrough but can be exhausted by aggressive concurrent scripts.

We are at minute forty.

Minutes forty to fifty-five — Signal 2

Implement Signal 2:

```
def signal_2():
    now = datetime.utcnow()
    long_lookback = fetch_commits(ORG, REPO, now - time
delta(days=180), now - timedelta(days=60))
    recent_60 = fetch_commits(ORG, REPO, now - timedelt
a(days=60), now)
    recent_14 = fetch_commits(ORG, REPO, now - timedelt
a(days=14), now)

    prior_logins = {c["author"]["login"] for c in long_
lookback
                    if c.get("author") and classify_aut
hor(c) == "human"}
    recent_logins = {c["author"]["login"] for c in rece
nt_60
                     if c.get("author") and classify_au
thor(c) == "human"}
    new_authors = recent_logins - prior_logins

    recent_14_logins = {c["author"]["login"] for c in r
ecent_14
                        if c.get("author") and classify
_author(c) == "human"}
    new_in_14 = recent_14_logins & new_authors

    score = min(100, len(new_in_14) * 20)
    fires = len(new_in_14) >= 4
    return {
        "fires": fires,
        "new_in_14": list(new_in_14),
        "count": len(new_in_14),
        "score": score,
    }

result_2 = signal_2()
print(f"Signal 2 fires: {result_2['fires']}, score: {re
sult_2['score']}, new authors: {result_2['new_in_14']}"
)
```

Run. The new-authors list is the most interesting output of the entire

walkthrough — these are the names of recent contributors. If the firing is real, you can paste any of them into LinkedIn search and find recent Modal hires. Try it. The first time you do this, the signal becomes viscerally real.

We are at minute fifty-five.

Minutes fifty-five to sixty-five — Signal 6

Skip ahead to Signal 6 because the dependency-adoption signal is the second-most-honest signal in the book, and Modal happens to be a developer-tools company with an active npm package. Implement:

```
def signal_6():
    package = "@modal-labs/cli"  # Modal's npm-publishe
d CLI
    url = f"https://libraries.io/api/npm/{package}/depe
ndent_repositories"
    params = {"api_key": LIBRARIES_IO_KEY, "per_page":
100, "page": 1}
    dependents = []
    while True:
        r = requests.get(url, params=params)
        if r.status_code == 404:
            return {"fires": False, "reason": "package_
not_indexed", "score": None}
        r.raise_for_status()
        page = r.json()
        if not page:
            break
        dependents.extend(page)
        if len(page) < 100:
            break
        params["page"] += 1

    now = datetime.utcnow()
    def parse_pushed(d):
        try:
            return datetime.fromisoformat(d["pushed_at"
].replace("Z", ""))
        except Exception:
```

```
            return None

    recent = [d for d in dependents
              if parse_pushed(d) and parse_pushed(d) >
now - timedelta(days=30)]
    prior = [d for d in dependents
             if parse_pushed(d) and now - timedelta(day
s=60) < parse_pushed(d) <= now - timedelta(days=30)]

    if len(recent) < 20:
        return {"fires": False, "reason": "absolute_flo
or", "score": 0}
    if not prior:
        return {"fires": False, "reason": "no_prior", "
score": 0}

    ratio = len(recent) / len(prior)
    score = min(100, max(0, (ratio - 1) * 30))
    return {
        "fires": ratio > 3.0,
        "ratio": ratio,
        "recent_count": len(recent),
        "prior_count": len(prior),
        "score": score,
    }

result_6 = signal_6()
print(f"Signal 6 fires: {result_6['fires']}, score: {re
sult_6.get('score')}")
```

If Modal does not publish a primary npm package under the name above (the canonical name has changed at least once historically), you may get a `package_not_indexed` result. Substitute their primary published package — you can find it at `npmjs.com/~modal-labs` or in their docs.

We are at minute sixty-five.

Minutes sixty-five to seventy-five — composing the score

Compute the partial Scout Score from Signals 1, 2, and 6:

```
def partial_score(s1, s2, s6):
    weights = {"s1": 0.22, "s2": 0.20, "s6": 0.14}
    s6_score = s6.get("score") or 0
    weighted = (
        s1["score"] * weights["s1"] +
        s2["score"] * weights["s2"] +
        s6_score * weights["s6"]
    )
    total_weight = sum(weights.values())
    normalized = weighted / total_weight
    return normalized

score = partial_score(result_1, result_2, result_6)
print(f"\nPartial Scout Score (signals 1, 2, 6): {score
:.1f}")
```

Now go to the GitDealFlow leaderboard at `signals.gitdealflow.com` and look up the same organization. The full Scout Score there will be in the same approximate range, but slightly different because it includes the four signals we did not implement in this appendix (3, 4, 5, 7). The relative magnitude should match: if your partial score is in the seventy-to-ninety range, the leaderboard will show a high overall rank; if your partial score is in the thirty-to-fifty range, the rank will be in the middle of the list.

We are at minute seventy-five.

Minutes seventy-five to ninety — verification and inspection

Spend the last fifteen minutes inspecting the output. Read the new-author list from Signal 2 and try to resolve at least three of them on LinkedIn. Read the firing reason for Signal 1 — was it the acceleration or the volume floor? Look at the commit log on `github.com/modal-labs/modal-client/commits` and confirm the recent commits look like real engineering work rather than dependency bumps.

The single most useful exercise at this point is to compare your computed numbers to a public Series A announcement timeline. Modal Labs announced its Series A on October 19, 2023. If you replicate this appendix on Modal as it stood in mid-September 2023 (you would need historical data to do this, which is harder), you would see Signal 1 firing strongly. The data has moved on, but the historical pattern is in the public record and is verifiable by date-bounding your `since` and `until` parameters.

If your numbers feel directionally off — much higher or much lower than the leaderboard would suggest — the most common cause is one of three things:

Wrong repository. Modal has multiple repositories. The signal is most informative on `modal-client`, which is their primary product repository. Computing on `modal-examples` or `modal-docs` will give different and less informative numbers.

Stale token. GitHub personal tokens expire. If you generated yours weeks ago, regenerate.

Rate-limit exhaustion. If you have run the script several times and exhausted your hourly quota, the API will start returning 403s. Wait an hour and re-run.

If your numbers feel directionally right but you want to validate against the live leaderboard, write to `signal@gitdealflow.com` with your numbers and I will pull the corresponding leaderboard score for that week and you can compare.

We are at minute ninety. You have, with a $0 budget, a personal access token, and roughly a hundred lines of Python, replicated a meaningful portion of the GitDealFlow signal stack on a real organization. You can now run this against any organization on GitHub. You have also verified that the methodology is real — you computed the numbers yourself from raw public data, and the numbers match the published methodology.

That is the deal this book makes. Public data, fully replicable, no licence, no scraping, no proprietary anything. The seven-signal stack

is yours now. The leaderboard, the MCP server, and the email digest are conveniences that automate what you have just done by hand. They are not where the value lives. The value lives in the methodology, and the methodology is in your hands.

Use it well.

CONCLUSION

Conclusion

What to do on Monday morning

If you read this book straight through and then put it down without changing anything you do on Monday morning, the book has failed. The signals are real, the methodology is reproducible, the historical record is in your favour — but none of that produces a different deal-flow funnel unless the workflow is actually run.

This conclusion is therefore a checklist. It is intentionally short.

The Monday checklist

Before nine a.m. on Monday morning.

1. Open your watchlist. If you do not have one, write down the names of fifteen to thirty companies you are interested in, with one line per company including the GitHub organization handle. Fifteen is enough to start; thirty is the maximum for a manual workflow before you need automation.
2. For each company, run Signal 1 — the fourteen-day commit-velocity acceleration with two-period confirmation. The script in the methodology chapter takes about thirty seconds per organization. For thirty organizations, budget fifteen minutes.

3. Note any firings — the small list of companies whose primary repository is showing two-hundred-per-cent acceleration with two-period confirmation. For most weeks this list will be one to four companies. Some weeks it will be empty. Both are normal.
4. For each firing, run Signal 2 — the contributor influx — and Signal 6 if the company is a developer-tools company. Note any concurrent firings.

Before noon on Monday morning.

5. For each concurrent firing, do the LinkedIn cross-reference on the new contributors. Confirm at least one of the new contributors as a recent hire. This takes about two minutes per firing.
6. For each confirmed-as-real concurrent firing, write a short founder reach-out. Plain text, no marketing copy, lead with a specific observation about the public commit graph. Three sentences. Send.
7. Subscribe to the free GitDealFlow Signal Digest at `gitdealflow.com` if you have not already. The digest delivers every Monday before nine a.m. Eastern and contains the top five firings across the entire universe of tracked organizations — not just your watchlist. The free digest catches the firings on companies you would never have put on your own watchlist, and those are usually the ones with the highest carry potential.

That is the whole Monday workflow. It takes between thirty and ninety minutes depending on the size of the watchlist and how many firings need cross-referencing. If you do this every Monday for a quarter, the deal-flow funnel that lands on your desk will be measurably different from the one that lands on the desk of the investor who is still relying on warm intros and Crunchbase Pro.

The compound effect of this workflow is real. It is also small per week and only visible in retrospect. Do not expect a transformative conversation in week one. Expect, by week twelve, to be in three or

four conversations that you would not have been in otherwise — and to have walked away from one or two firings that turned out, on closer inspection, to be false positives. That ratio is the point.

Three honest failure modes

Three things will go wrong if you adopt this workflow. I want to name them so you do not feel like you are doing it wrong when they happen.

The first quarter will feel slow. The lead times for the signals are weeks, not days, and the conversation circuit takes longer than the lead times. You will run the workflow for six or eight Mondays before you have a single closed conversation that you can attribute directly to the workflow. This is not a sign that the workflow is broken; it is a sign that the workflow is operating at its actual lead time. Stick with it.

You will catch a false positive that feels like a true positive. Around week six or eight you will have a strong-feeling firing, you will reach out to the founder, the conversation will go well, and the company will turn out to be preparing for something other than a Series A — an acquisition, a product launch, a pivot. The conversation will not have been wasted, but it will not have produced the carry you expected. This is normal. The seven-signal stack has a sixty-eight per cent hit rate, which means a thirty-two per cent miss rate, and you cannot tell which side of the line a particular firing is on until the round-or-not-round event closes. Treat false positives as part of the cost of the funnel, not as evidence that the funnel does not work.

You will be tempted to skip the methodology and trust the GitDealFlow leaderboard directly. I run the GitDealFlow leaderboard. I would be flattered if you trusted it. I am also telling you not to. The reason is that the leaderboard is a convenience layer over the methodology. If you trust the leaderboard without ever

having computed a signal yourself, you do not actually know what the leaderboard is computing, and you cannot defend the use of the leaderboard to your partners or your LPs. Compute one or two signals by hand, against the appendix walkthrough, in your first month. Then trust the leaderboard, with full understanding of what it is doing.

The case for treating public data as the new private network

If there is one organising idea I want you to take from this book, it is this: the public surface of a Series-A-bound startup is more honest, more current, and more comprehensive than the private network around it.

The private network — the warm-intro circuit, the partner-track relationship layer, the "this person is going to raise" whispers — is genuinely valuable when it works, and it does work. But it is not a level playing field. It is, structurally, the playing field of investors who have spent a decade compounding a Rolodex. New entrants to venture capital, including most developer-investors, do not have that Rolodex. They cannot manufacture one in the time the seed-round window stays open.

What new entrants do have is the public surface. They can read commit logs, contributor graphs, dependency trees, conference talks, and engineering blog posts as fluently as a senior partner reads body language at a dinner. The skills are different, but the resulting edge is comparable, and the public-data edge has the structural property that it does not depend on access. The data is there. Anybody who reads this book can use it, indefinitely, on a zero-budget infrastructure footprint.

This does not mean the warm-intro circuit is dying. It is not. The warm-intro circuit will outlive me, you, and this book. But it has been the only path into early-stage venture for a long time, and it is no longer the only path. There is now a parallel path, built on public

data, accessible to anyone with a personal access token and a willingness to read this book. That is the thing this book is, in the end, trying to help build: a wider, more accessible, more falsifiable path to early-stage venture capital, on a foundation of data that nobody can paywall.

If that succeeds — even partially — then the kind of deal flow that landed on Series A partners' desks in 2018 will be available, in 2028, to a much wider community of developer-investors and small-fund analysts. That is a better venture-capital industry. It is what I want to be part of building, and I think this book is a contribution to that.

A request

If this book is useful to you, please tell another investor about it. Word-of-mouth is the only meaningful distribution channel for a book like this; there is no advertising budget. The free PDF and EPUB downloads at `gitdealflow.com/book` are sized to be sent as an email attachment without scaring anyone's spam filter.

If a chapter is wrong, or a threshold is poorly calibrated, or a methodology step does not reproduce, please write to `signal@gitdealflow.com`. The next edition will fold in your correction with attribution. The methodology is meant to be improved over time, not preserved as scripture.

If you read the book, run the workflow, and close a deal that you can attribute even partially to a signal in here, write and tell me. I will not publish the deal — anonymity is a foundational rule of this project — but the count of attribution-positive deals is the only metric I track that matters, and I will be grateful for the data point.

Thank you for reading. Now turn the page on this book and open your terminal. The signals are waiting.