

Journal of Intelligent Digital Systems and Innovation

available at http: www.econjournals.com

Journal of Intelligent Digital Systems and Innovation, 2026, 2(3), 11-25.

Innovation Tempo in Intelligent Systems Software: Self-Exciting Release Dynamics, Dependency Concentration and Maturation in the Python Artificial-Intelligence Ecosystem, 2018-2026

Zainab Hussein Arif*

College of Nursing, University of Al-Qadisiyah, Al-Qadisiyah Province, Iraq. *Email: zhussian94@gmail.com

Received: 05 July 2026 **Accepted:** 22 September 2026 **DOI:** https://doi.org/10.32479/jidsi.25437

ABSTRACT

Intelligent systems are assembled from open-source libraries, and they can change no faster than those libraries do. We measure the pace of change for Python packages that the Python Package Index classifies under artificial intelligence, using release histories cut at 31 May 2026 and modelling each release stream as an exponential-kernel Hawkes process estimated by maximum likelihood. Two-thirds of projects reject a Poisson stream. The median branching ratio is 0.29 and the median excitation half-life 58 h. A break at the public release of ChatGPT in November 2022 lowers the baseline rate; placebo dates a year apart lower it more, and an interrupted time series of releases per active project finds no change in level or slope, so the growth after 2022 comes from entry and from a secular trend. In the runtime-dependency network, in-degree follows a discrete power law with exponent near 1.95. Removing one package exposes 66% of AI projects. Projects first uploaded after November 2022 reach version 1.0 sooner (hazard ratio 1.41). Self-excitation, entry and concentrated dependency describe innovation tempo better than a single generative-AI shock.

Keywords: Intelligent Systems, Open-Source Innovation, Hawkes Process, Software Ecosystems, Dependency Networks, Survival Analysis
JEL Classifications: O33, O31, L86, C41

1. INTRODUCTION

Almost every intelligent system shipped today, from a retrieval assistant to a vision pipeline on a factory line, is built from packages that someone else maintains, and most of those packages live on the Python Package Index (PyPI). A team that installs a model-serving framework inherits its release schedule. It also inherits the schedules of a few dozen libraries underneath, and when one of them ships a breaking change the system above has to absorb it. Each upload in this layer carries a timestamp to the microsecond, so innovation here can be counted with a precision that patent or publication data rarely allow (Korkmaz et al., 2024; Tang et al., 2020).

How fast does that layer move? Timing is one half of the question: Releases may arrive independently, as in a homogeneous Poisson stream, or one release may raise the chance of the next, as when a fix exposes a second bug or a new feature draws a run of follow-up patches. Structure is the other half, meaning which packages sit under which and how far a change in one reaches. Both halves bear on a claim heard often since late 2022, that generative AI accelerated the whole ecosystem (Leon, 2026; Truong and Papagiannidis, 2022). If it did, the release process should shift measurably around 30 November 2022, the day ChatGPT was opened to the public. That can be tested.

We test it on AI-classified PyPI projects drawn from the 15,000 most downloaded packages of April 2026 (van Kemenade, 2026), with metadata from the PyPI JSON interface (Python Software Foundation, 2026) and a cut-off of 31 May 2026. The final releases of each project are treated as a point process with conditional intensity.

This Journal is licensed under a Creative Commons Attribution 4.0 International License

$$\lambda\left(t \mid \mathcal{H}_t\right)=\mu+\alpha\beta\sum_{t_i<t}e^{-\beta\left(t-t_i\right)},\mu>0,0\le\alpha<1,\beta>0, \quad (1)$$

Where H_t is the release history up to t, μ the baseline rate, α the expected number of releases triggered directly by one release (the branching ratio) and $ln2/\beta$ the half-life of that triggering. The long-run release rate under Equation (1) is $\mu/(1–\alpha)$. An identical observed pace can therefore come from a high baseline or from strong self-excitation, and the difference matters to anyone planning upgrades, because a baseline is a steady load while excitation arrives in bursts.

The paper adds three things to what is known. Hawkes parameters are estimated project by project, with likelihood-ratio tests whose critical values had to be simulated (under the Poisson null β is not identified) and with time-rescaling checks of fit. The generative-AI acceleration claim is then tested inside established projects and across the ecosystem, against placebo break dates that turn out to matter more than the real one. Finally, release tempo is placed inside the dependency network and the maturation path of new projects, through a discrete power law for in-degree, the exposure of AI projects when central packages are removed, and a Cox model for the time to a first 1.0 release.

Section 2 reviews related work and states four hypotheses; Section 3 describes the data and the models, Section 4 reports results, Section 5 discusses implications and limits, and Section 6 concludes.

2. LITERATURE REVIEW AND HYPOTHESES

2.1. Innovation Tempo in Artificial Intelligence

Papers and patents have carried most measurements of AI innovation speed. Tang et al. (2020) defined the pace of an AI innovation as the time from the first paper on a technique to its widespread use, traced how that interval shortened, and named talent and trial-and-error as drivers. Damioli et al. (2025) asked whether AI patents form a new technological paradigm. They found pervasiveness and fast growth in applications, the profile of a general-purpose technology, and Leon (2026) applied the same lens to generative models with projections of labour-market effects to 2030. At the level of the firm, AI is tied to innovation output through collaboration: Research and development alliances in AI raise performance up to a point (Ming et al., 2025), AI enters innovation management at every stage from idea generation to portfolio decisions (Roberts and Candi, 2024), and Carayannis and Preissler (2025) added AI as a sixth element to the quintuple helix of innovation, with semiconductors as their case. Two reviews round this out, one of AI as an enabler of innovation (Truong and Papagiannidis, 2022) and one of diffusion through open-innovation networks (Zhang and Wang, 2021).

Software artefacts rarely serve as the unit. Where they do, open code shows up as a channel of diffusion: Machine-learning papers with usable GitHub repositories collect more citations (Kang et al., 2023), and a small set of Python libraries came to carry most of machine-learning practice (Raschka et al., 2020). In those accounts release cycles are background. Their timing is seldom modelled.

2.2. Open-Source Software as Innovation Infrastructure

Open-source code has also been measured as an economic input. Korkmaz et al. (2024) estimated the cost of developing it from GitHub activity and argued that it belongs in national accounts, while Wright et al. (2023) linked national participation in GitHub to the founding and financing of technology ventures. For AI, Vake et al. (2025) compared open and closed models through benchmark trajectories and download data, finding that open models close the gap with a lag. Shi and Jiang (2025) proposed a framework to evaluate the impact of open-source foundation-model ecosystems and the mechanisms that promote them. Openness is treated as an asset throughout. The flow of releases that keeps the asset current gets far less attention.

Software engineering has looked at that flow directly. In a systematic review of frequent releases in open-source projects, rapid release went with faster feedback and with more pressure on testing (Da Silva et al., 2017). Ochoa et al. (2022) checked semantic-versioning discipline in Maven Central and found breaking changes concentrated in a minority of libraries; minor and patch releases broke clients less often than earlier studies had claimed. Deep-learning frameworks accumulate technical debt of distinct types, introduced and removed at different rates (Liu et al., 2021). Machine-learning assets such as models and datasets bring management problems of their own (Zhao et al., 2024), and bindings that expose machine-learning libraries in other package ecosystems often lag the host library's releases (Li and Bezemer, 2025).

2.3. Dependency Networks and their Concentration

Package ecosystems form directed networks, with an edge from A to B whenever A requires B, and Decan et al. (2019), comparing such networks across seven ecosystems, documented heavy-tailed in-degree and a steady growth in transitive dependency. For Python, Paramitha and Massacci (2023) measured technical leverage, the ratio of dependency code to own code, and its link to vulnerability exposure. Hubs carry the risk. A defect or a breaking change in one of them propagates to every dependant, which is why the degree distribution, often reported as a power law, draws so much attention:

$$\Pr\left(K=k\right)=\frac{k^{-\gamma}}{\zeta\left(\gamma,k_{\min}\right)},k\ge k_{\min}, \quad (2)$$

With $\zeta(\gamma, k_{min}) = \sum_{j\ge0} (j+k_{min})^{-\gamma}$ the Hurwitz zeta function. A power-law claim carries weight only after a goodness-of-fit test and a comparison with other heavy-tailed laws, and the same caution holds for inter-event times, where power-law behaviour has been reported for word occurrences (Ogura et al., 2026) and where renewal processes with ageing produce similar patterns (Vanni and Lambert, 2024).

2.4. Self-Exciting Point Processes

Hawkes processes describe events that raise the rate of later events. The earliest application to innovation data that we found

is that of Jang et al. (2017), who measured the technological impact of a patent as the excitation it produces among later citations; Ganeshbabu et al. (2026) carried the idea to scientific impact, with excitation modulated by a virtual geography of citing papers. Intelligent-systems research has turned the Hawkes intensity into neural architectures. Zhang et al. (2021) built a universal transformer Hawkes process with adaptive recursion. Zhou et al. (2023) dropped the explicit intensity altogether and modelled event context with convolutions. On the statistical side there is variational inference for nonlinear Hawkes processes with Gaussian-process self-effects (Malem-Shinitski et al., 2022), work on inverse Hawkes processes with non-Markovian structure (Seol, 2022), the coarse-grained process that appears when events are counted in bins (Koyama, 2025), and tests for comparing two point processes (Mu and Wu, 2026). Neural variants predict well. They return no branching ratio to interpret, though, and for the questions asked here a parametric kernel is the better tool.

2.5. Hypotheses

Table 1 locates the gap: Release streams of AI software have not been modelled as point processes, and nobody has tested the generative-AI acceleration claim against them. Four hypotheses follow.

H_1. Release streams of AI-classified projects are self-exciting: The branching ratio exceeds zero and a Hawkes model fits better than a homogeneous Poisson process.

H_2. The public release of ChatGPT on 30 November 2022 raised the release process, as a higher baseline or a higher branching ratio inside established projects and as a level or slope change across the ecosystem.

H_3. The runtime-dependency network of AI projects is concentrated: In-degree is heavy-tailed, and the removal of a few packages exposes most AI projects.

H_4. Projects first uploaded after 30 November 2022 reach a first 1.0 release sooner than earlier cohorts.

H_2 and H_4 can diverge. A shock could speed the maturation of new projects without changing how often established ones release, and the analysis is set up to tell the two apart.

3. MATERIALS AND METHODS

Figure 1 summarises the design (three data layers built from the download list and the PyPI metadata, three groups of models, and a Monte Carlo check of the estimators before they touch the data).

3.1. Data and Sample

The sampling frame is the list of the 15,000 PyPI projects with the most downloads in April 2026, published on 1 May 2026 (van Kemenade, 2026); its counts come from the public PyPI download records. For every project the JSON record was retrieved from PyPI (Python Software Foundation, 2026) in the 1st week of November 2026, and each record holds the trove classifiers, the requirement strings of the latest release and, for every version, the upload time of each distribution file. A project enters the sample if its classifiers include Topic: Scientific/Engineering: Artificial Intelligence. Membership is self-declared. What the sample contains is therefore what maintainers call AI; general-purpose dependencies such as numpy enter the network analysis as dependencies without being AI projects themselves.

A version's upload time is the earliest upload among its files. Versions that do not parse under PEP 440 are dropped, as are versions whose files have all been yanked, while pre-releases and development releases stay in the descriptive counts but are kept out of the point-process models, which use final releases only. Nothing uploaded after 31 May 2026 enters any estimate, although the metadata were retrieved later. Table 2 gives the counts.

Two biases come with this frame and cannot be removed. Because the list is a 2026 snapshot, projects that were popular in 2019 and later abandoned are missing, so survivorship inflates the apparent maturity of early cohorts and deflates entry counts for the last months, when new projects have had little time to collect downloads. The classifier is optional. Torch, transformers and vllm carry it; scikit-learn, keras, jax, xgboost and the openai client do not, and they enter the analysis only where AI projects depend on them. Section 5.4 returns to both biases.

3.2. Point-Process Model of Releases

For project the observation window opens at $s_p = \max(1\text{ January }2018, f_p)$, where f_p is its first upload of any kind, and closes at $E = 31$ May 2026 23:59:59 UTC. A project born inside the window has its first upload as the time origin, and that upload is not counted as an event, which keeps the years before birth out of the exposure. Times are in days from s_p, giving events $0 < t_1 < \cdots < t_n \le T_p$. Batch uploads produce identical timestamps; these are separated by 10^{-6} days. Under Equation (1) the log-likelihood is

$$\ell(\mu,\alpha,\beta) = \sum_{i=1}^{n} \ln\left(\mu + \alpha\beta A_i\right) - \mu T_p - \alpha \sum_{i=1}^{n} \left(1 - e^{-\beta\left(T_p - t_i\right)}\right), \quad (3)$$

Table 1: Selected related studies, grouped by the gap each leaves for the present analysis

Stream	Representative studies	Unit of analysis	What remains open
AI innovation pace	Tang et al. (2020); Damioli et al. (2025)	Papers, patents	Release timing of software artefacts
Open source as infrastructure	Korkmaz et al. (2024); Wright et al. (2023); Shi and Jiang (2025)	Repositories, countries, model ecosystems	Dynamics of the release flow
Release engineering	Da Silva et al. (2017); Ochoa et al. (2022); Liu et al. (2021)	Releases, versions, debt	Stochastic model of release clustering
Dependency networks	Decan et al. (2019); Paramitha and Massacci (2023)	Package graphs	AI-specific closure and exposure
Hawkes applications	Jang et al. (2017); Ganeshbabu et al. (2026); Zhang et al. (2021)	Citations, events	Software release streams, break tests

Figure 1: Analytical framework. Three data layers built from the download list and the PyPI metadata feed three groups of models; the Monte Carlo study checks the Hawkes estimator and the break test before they are applied

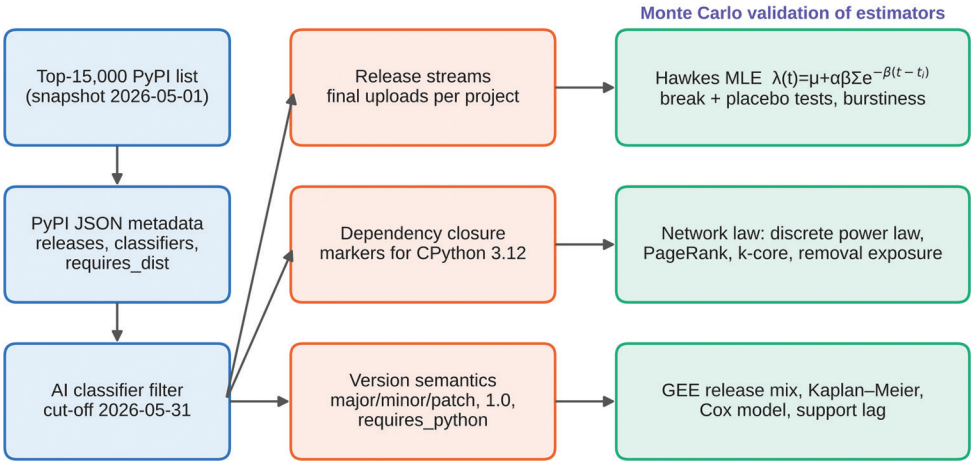


Table 2: Construction of the analysis samples

Item	Count
Projects in the top-15,000 snapshot (downloads in April 2026)	15,000
Projects with the AI topic classifier	656
Non-yanked uploads with a PEP 440 version, to 31 May 2026	84,587
Pre-releases or development releases among those uploads	40,351 (47.7%)
Final releases, 1 January 2018 to 31 May 2026	43,441
Projects with at least 25 final releases in the window (Hawkes sample)	336
Hawkes-sample projects testable at the 30 November 2022 break (12+ events each side)	128
Nodes in the runtime-dependency closure	2,226
Directed dependency edges	8,803
Projects born in the window and eligible for the time-to-1.0 analysis	486

With the recursion

$$A_1 = 0,\ A_i = e^{-\beta(t_i - t_{i-1})}(1 + A_{i-1}),\ i \geq 2, \tag{4}$$

Which evaluates the sum over past events in $O(n)$ operations, so that the parameters can be estimated on the transformed scale ($\ln\mu$, logit (α/0.999), $\ln\beta$) by L-BFGS-B, started from five points that spread the half-life from minutes to weeks, with the best of the five fits kept. The bound $\alpha < 0.999$ keeps the process stationary. Setting $\alpha = 0$ gives the homogeneous Poisson process, with $\hat{\mu} = n / T_p$, log-likelihood $l_0 = n\ln(n/T_p)$–n and likelihood-ratio statistic

$$\Lambda = 2\left[\ell\left(\hat{\mu}, \hat{\alpha}, \hat{\beta}\right) - \ell_0\right]. \tag{5}$$

The χ^2 reference does not apply to Equation (5). Under $\alpha = 0$ the decay β drops out of the model and the null lies on the edge of the parameter space, so critical values were simulated: For each $n \in \{25, 50, 100, 200, 400, 800\}$, 600 Poisson samples were fitted with the same optimiser and the 95% and 99% quantiles of Λ were interpolated in $\ln n$. The 99% values lie between 6.7 and 8.8. Only projects with at least 25 final releases in the window are modelled, a threshold fixed before any fitting.

Fit is checked by time rescaling. With the fitted compensator $\Lambda(t) = \int_0^t \lambda(u)\,du$, the increments

$$\tau_i = \Lambda(t_i) - \Lambda(t_{i-1}) = \hat{\mu}(t_i - t_{i-1}) + \hat{\alpha}\, S_{i-1}\left(1 - e^{-\hat{\beta}(t_i - t_{i-1})}\right), S_i = 1 + e^{-\hat{\beta}(t_i - t_{i-1})} S_{i-1}, \tag{6}$$

Are independent unit exponentials when the model is right, and a one-sample Kolmogorov-Smirnov test compares them with Exp (1). A second benchmark lets the Poisson rate change every calendar year, as a step function with up to nine levels; when the Akaike criterion still prefers the Hawkes model, clustering at short lags cannot be put down to slow drift in the rate.

3.3. Break and Placebo Tests

Inside projects, H_2 is tested by letting the baseline and the branching ratio change at a date τ:

$$\lambda(t) = \mu_1 1\{t < \tau\} + \mu_2 1\{t \geq \tau\} + \beta \sum_{t_i < t} \alpha(t_i) e^{-\beta(t - t_i)}, \alpha(t_i) = \begin{cases} \alpha_1, & t_i < \tau \\ \alpha_2, & t_i \geq \tau \end{cases} \tag{7}$$

Excitation from releases before τ carries across the break, and β is shared. Against the unbroken model the likelihood-ratio statistic has two degrees of freedom, and with α away from the boundary on both sides a χ^2_2 reference (critical value 5.99) applies; its size is checked by simulation in Section 4.7. A project is testable at τ when it has at least 12 events on each side, and across testable projects the ratio $\hat{\mu}_2 / \hat{\mu}_1$ and the difference $\hat{\alpha}_2 - \hat{\alpha}_1$ are summarised with Wilcoxon signed-rank tests. Release rates also fall as projects age. A decline at τ = 30 November 2022 therefore means nothing on its own, so the same test is repeated at four placebo dates (30 November of 2019, 2020, 2021 and 2023) and the ChatGPT date is read against them.

Across the ecosystem, monthly counts Y_m of final releases from January 2018 to May 2026 are modelled as negative binomial,

with an offset for the number of active projects N_m (those whose first upload precedes the end of month m):

$$\ln E[Y_m] = \ln N_m + \gamma_0 + \gamma_1 m + \gamma_2 D_m + \gamma_3 (m - m_\tau) D_m, \quad (8)$$

Where $D_m = 1$ from December 2022 onward and m_τ is that month. The dispersion ϕ is fixed at its moment estimate from a Poisson fit and the standard errors are Newey-West with three lags; without the offset, the same model describes the monthly entry of new projects.

3.4. Burstiness and Memory

Two model-free statistics describe the inter-release times $\Delta_i = t_{i+1} - t_i$, the first being burstiness

$$B = \frac{\sigma_\Delta - m_\Delta}{\sigma_\Delta + m_\Delta} \quad (9)$$

Which is –1 for a regular stream, close to 0 for a Poisson stream and near 1 for extreme bursts. The memory coefficient M is the Pearson correlation between consecutive gaps (Δ_i, Δ_{i+1}). Both statistics are biased in small samples, which is why each project's values are set against 400 Poisson streams of the same length, the one-sided P-value being the share of null draws at or above the observed value.

3.5. Dependency Network

Requirement strings of the latest release of every AI project are parsed under PEP 508. Requirements tied to an optional extra are dropped and environment markers are evaluated for one reference environment, CPython 3.12 on 64-bit Linux, so that a requirement such as exceptiongroup; python_version <"3.11" does not count; every new dependency is then fetched in turn, breadth first, until no new name appears, which took four rounds. Names are normalised as in PEP 503. The result is a directed graph $G = (V, E)$ with an edge (u, v) whenever u requires v.

For a node v, the direct AI in-degree k_v^{AI} counts the AI projects that list v, and the transitive reach

$$r_v = |\{u \in A : v \in Desc_G(u)\}| \quad (10)$$

Counts the AI projects A whose closure contains v. PageRank runs on G with damping 0.85. Importance then flows from dependants to dependencies. Core numbers come from the undirected simple graph. For each candidate k_{min} the power law of Equation (2) is fitted by maximum likelihood, the k_{min} with the smallest Kolmogorov-Smirnov distance is kept, and 200 semi-parametric bootstrap samples give the goodness-of-fit test. The power law is then set against a discrete exponential and a discrete log-normal on the same tail through the normalised log-likelihood ratio of Vuong,

$$z = \frac{\sum_j \left[\ln p_{\mathrm{PL}}(k_j) - \ln p_{\mathrm{alt}}(k_j)\right]}{\sqrt{n}\, s_d} \quad (11)$$

With s_d the standard deviation of the pointwise differences; a positive z favours the power law.

Exposure is measured by removal. Nodes are taken out in order of r_v, of PageRank or of in-degree, the share of AI projects whose closure contains at least one removed node is recorded after each removal, and 300 random orders supply the reference band. A project counts as exposed once any part of its closure is gone. That is the position of a team whose build breaks when one package is withdrawn or ships a broken release.

3.6. Release Taxonomy, Maturation and Support Lag

Each final release is classified against the highest final version released before it. Raising the first component of the version makes a release major, raising the second under the same major makes it minor, and raising a later component (or issuing a post-release) makes it a patch; anything below the running maximum is a backport. Calendar-versioned projects, with a first component of 1990 or more, are left out of the taxonomy. Shares of each type are modelled with generalised estimating equations (binomial, exchangeable working correlation within project) against a post-break indicator and, separately, a linear year trend.

Maturation is the time from a project's first upload to its first final release with major version 1 or higher, and the analysis keeps projects that were first uploaded in the window and started below 1.0. The 106 projects launched at 1.0 or above have no waiting time to model. Projects with no 1.0 release by 31 May 2026 are right-censored. Kaplan-Meier curves are drawn for four birth cohorts, and the Cox model:

$$h(t|x) = h_{0,s}(t) \exp(\theta_1 \mathrm{post} + \theta_2 \log_{10} \mathrm{downloads} + \theta_3 \mathrm{classifiers}) \quad (12)$$

Is stratified by runtime-dependency count (s: None, one to four, five or more), because that count failed the proportional-hazards check when entered as a covariate. Downloads are measured at the end of the window and are themselves an outcome of maturation, so their coefficient describes association and nothing more.

Support lag uses the requires_python field of each release. Its lowest admitted minor version is compared with the oldest Python version still maintained on the upload date, using the end-of-life dates published by the Python core developers (2.7 on 1 January 2020, 3.5 on 30 September 2020, 3.6 on 23 December 2021, 3.7 on 27 June 2023, 3.8 on 7 October 2024 and 3.9 on 31 October 2025). A release admitting a version already past end of life carries legacy support. The lag is the number of months since that end of life.

3.7. Simulation Design

The estimator and the break test were checked by Monte Carlo before the results were read. Hawkes processes were simulated by Ogata thinning with $\mu = 0.05$ per day, $\alpha \in \{0.2, 0.5, 0.8\}$ and $\beta \in \{0.1, 1, 10\}$ per day, over horizons chosen so that the expected count $\mu T/(1-\alpha)$ equals 50, 150 or 400, with 150 replicates in each of the 27 cells. The break test was run on a process with $\alpha = 0.5$ and $\beta = 1$ over 800 days, the baseline multiplied by 1, 1.25, 1.5, 2 or 0.5 after day 400, in 200 replicates each. All computation used Python 3.13 with NumPy, SciPy, statsmodels, lifelines and NetworkX. The likelihood was compiled with Numba.

4. RESULTS

4.1. Growth of the Release Stream

Final releases from AI-classified projects rose from 794 in 2018 to 10,638 in 2025, and the first 5 months of 2026 added 5,010 (Figure 2, panel a). Entry drives most of it. Surviving projects first uploaded in 2024 number 107, more than in any earlier year, and the drop in 2025 and 2026 is what survivorship predicts for projects that have had little time to reach the download list (panel b). Per active project, releases grew from about 0.85 a month in early 2018 to about 1.4 in 2025 (panel c). Pre-release and development uploads were 30.5% of all uploads in 2018 and 63.0% in 2019, then fell to 40.2% in 2025 (panel d). Since 2019, final releases have grown faster than release candidates.

The segmented model in Equation (8) puts the trend in releases per active project at $\hat{\gamma}_1 = 0.0078$ a month (Table 3), a compound rate of 9.8% a year. Neither the level ($\hat{\gamma}_2 = -0.064$, P = 0.458) nor the slope ($\hat{\gamma}_3 = -0.0025$, P = 0.401) changes at December 2022. Entry is another matter. It grew at 0.0151 a month until the break and more slowly after it ($\hat{\gamma}_3 = -0.0193$, P = 0.025), and with fewer recent entrants on the download list, part of that slowdown is a survivorship artefact that the entry series cannot separate from a real one.

4.2. Self-Excitation within Projects

The Hawkes model was fitted to the 336 projects with at least 25 final releases in the window (Table 4). Their median baseline is 9.2 releases a year. With a median branching ratio of 0.289, a typical release triggers about 0.29 further releases directly and, counting the whole cascade, $1/(1-\alpha) - 1 \approx 0.41$ in total. The spread is wide. In 28.9% of projects $\hat{\alpha} < 0.1$, in 36.6% $\hat{\alpha} > 0.5$, and the half-lives fall into two regimes (Figure 3, panel b): In 45.5% of projects excitation decays within a day, the signature of hot-fix releases that follow a faulty upload within hours, whereas in 26.5% it lasts longer than 30 days, which fits a period of intense development better than a reaction to one release.

Figure 4 shows four fitted intensities. langchain ran at close to one release a day through most of 2023, with $\hat{\alpha} = 0.91$ and a half-life of 16.5 days, and its intensity fell to about 0.2 a day after langchain-core, first uploaded in November 2023, took over much of the release activity. mL flow combines a high baseline with a half-life of 280 days and comes close to a slowly drifting Poisson stream. In diffusers each release is followed by a short spike with a half-life under a day. Sentence-transformers sits between them.

Baseline and branching ratio trade off against each other (Figure 3, panel c). Projects with a low baseline and a high $\hat{\alpha}$ release

Figure 2: Growth of the AI-classified segment of PyPI. (a) Final releases per calendar year; 2026 covers January to May. (b) First-upload year of the projects in the sample, which survive to the 2026 download list. (c) Monthly final releases per active project with the segmented negative binomial fit of Equation (8); the dashed line marks 30 November 2022. (d) Share of pre-release and development uploads among all uploads

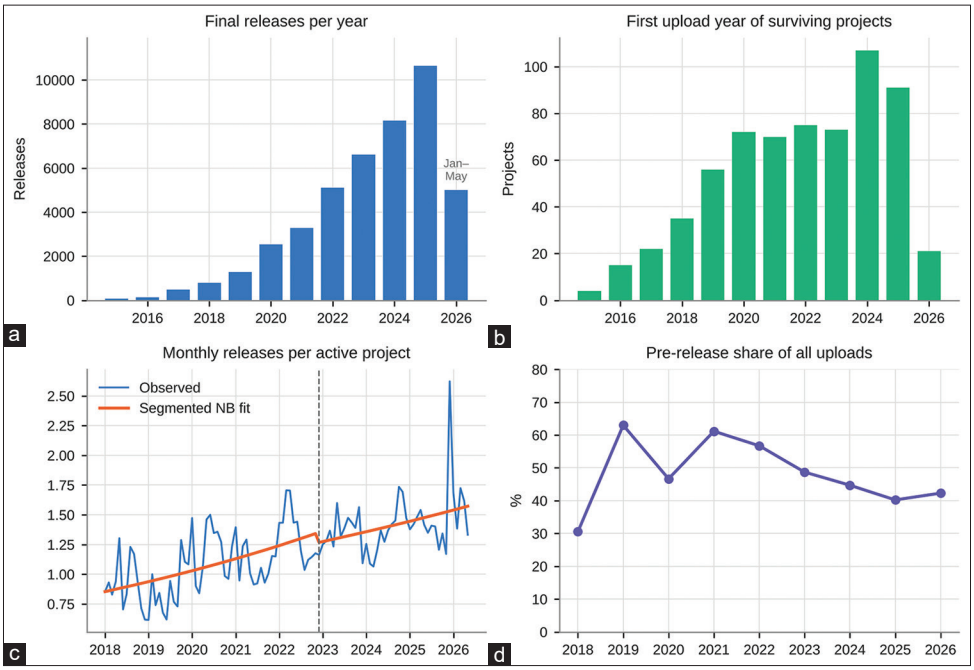


Table 3: Segmented negative binomial models of monthly final releases (offset: Active projects) and of monthly entry of new projects, January 2018-May 2026. Newey-West standard errors with three lags

Term	Releases: Coef.	SE	P	Entries: Coef.	SE	P
Intercept γ_0	−0.1582	0.0760	0.037	1.1670	0.1244	<0.001
Month trend γ_1	0.0078	0.0022	<0.001	0.0151	0.0035	<0.001
Level shift γ_2	−0.0642	0.0866	0.458	−0.0156	0.2208	0.944
Slope change γ_3	−0.0025	0.0030	0.401	−0.0193	0.0086	0.025
NB dispersion ϕ	0.032			0.151		

in clusters separated by quiet months, and projects with a high baseline and a low $\hat{\alpha}$ release on something close to a schedule. Eighteen projects have $\hat{\alpha} > 0.95$, at the stationarity bound. Two are nightly-build channels (tfg-nightly, mct-quantizers-nightly); the others include long-lived projects such as transformers, datasets and huggingface-hub, whose fitted half-lives run to months, so the stationary model absorbs a slowly varying rate as near-critical excitation. Without the eighteen, the median $\hat{\alpha}$ is 0.244.

4.3. Goodness of Fit and Burstiness

Against the simulated 99% critical value the Hawkes model wins in 64.9% of projects (Table 5), and the Akaike criterion prefers it to the homogeneous Poisson process in 76.2%. Slow drift in the rate accounts for part of that clustering. Against the yearly step-function benchmark the Hawkes model still wins in 63.1% of projects. The time-rescaling test rejects it at 5% for 76 projects, or 22.6% (Figure 5). In the pooled Q-Q plot the rescaled residuals follow the diagonal up to about the 95th percentile and bend upward beyond it, so the largest gaps (long pauses between bursts) are longer than an exponential kernel allows; a power-law kernel or a time-varying baseline would address that tail, and neither was fitted here.

Table 4: Distribution of the fitted Hawkes parameters and of the model-free statistics across the 336 modelled projects

Quantity	Q1	Median	Q3
Baseline rate $\hat{\mu}$ (per year)	4.46	9.21	24.80
Branching ratio $\hat{\alpha}$	0.067	0.289	0.684
Kernel half-life $\ln 2 / \hat{\beta}$ (h)	1.8	58.6	898
Final releases per project	35	62	107
Likelihood-ratio statistic against poisson	4.25	21.10	68.96
Burstiness B	−0.054	0.077	0.230
Memory coefficient M	0.022	0.129	0.281

Table 5: Model comparison and goodness-of-fit criteria across the 336 modelled projects

Criterion	Share of projects (%)
LR above the simulated 99% critical value	64.9
LR above the simulated 95% critical value	74.4
AIC favours Hawkes over homogeneous poisson	76.2
AIC favours Hawkes over yearly piecewise poisson	63.1
Time-rescaling KS test rejects at 5%	22.6
B above its poisson null at 5%	46.1
M above its poisson null at 5%	40.8
$\hat{\alpha} > 0.95$ (near the stationarity boundary)	18 projects

Burstiness confirms the clustering without any model. The median B is 0.077 and the median M is 0.129; B exceeds its Poisson null at 5% in 46.1% of projects and M in 40.8% (Figure 6). The rank correlation of B with $\hat{\alpha}$ is 0.66, against 0.25 for M, so the branching ratio picks up mostly the burst structure and only partly the serial dependence of gaps. H_1 is supported.

Release rates fall with age. In a regression on years observed, log downloads and classifier count, the log annual release rate drops by 0.266 for each year observed (Table 6), a decline of 23% a year, with $R^2 = 0.376$. The branching ratio moves the other way, rising by 0.130 on the logit scale per year ($P = 0.046$), so older projects release less often and, when they do, more in clusters. Downloads predict neither quantity.

4.4. The generative-AI Break and its Placebos

At 30 November 2022, 128 projects are testable. In 75.8% of them the baseline is lower after the break, with a median ratio $\hat{\mu}_2 / \hat{\mu}_1$ of 0.67 (Wilcoxon $P < 0.001$), while the branching ratio does not move (median change 0.000, $P = 0.236$) and the break is significant at 5% in 71 projects, or 55.5% (Table 7). For 13 projects the fitted post-break baseline collapsed to the lower bound of the optimiser, because nearly every later release in those projects is attributed to excitation. They sit at the bottom of Figure 7, panel b. Without them the median ratio is 0.73, still below one.

Read in isolation, the 2022 result would say that generative AI slowed established projects. The placebo dates rule that out. Every placebo gives a decline, and the three earlier ones give larger declines than the real date does: 0.18 in 2019, 0.31 in 2020 and 0.47 in 2021, against 0.67 in 2022 and 0.62 in 2023 (Figure 7, panel a). Any date produces a break of this size in projects whose rate decays with age. The ChatGPT date is distinguished only by having the smallest decline of the three closest dates, a pattern too weak to

Figure 3: Fitted parameters. (a) Branching ratio $\hat{\alpha}$, with the median marked. (b) Excitation half-life $\ln 2 / \hat{\beta}$ on a log scale, with 1 h, 1 day and 1 month marked. (c) Baseline rate against branching ratio, coloured by the number of events

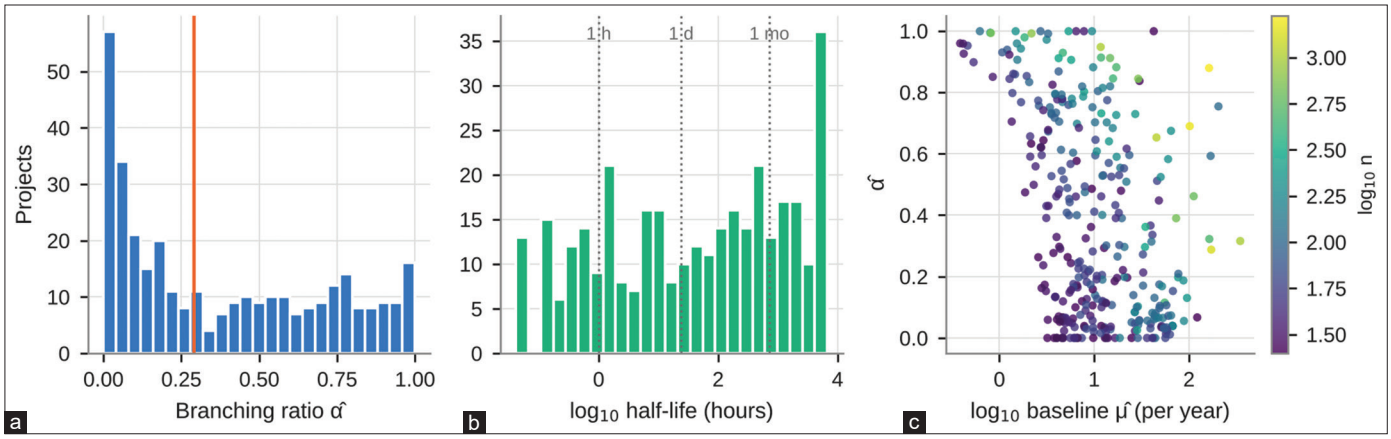


Figure 4: Fitted conditional intensities $\hat{\lambda}(t)$ for four projects. Tick marks below each curve are final releases; spikes are truncated at the 99.5th percentile of the intensity for display

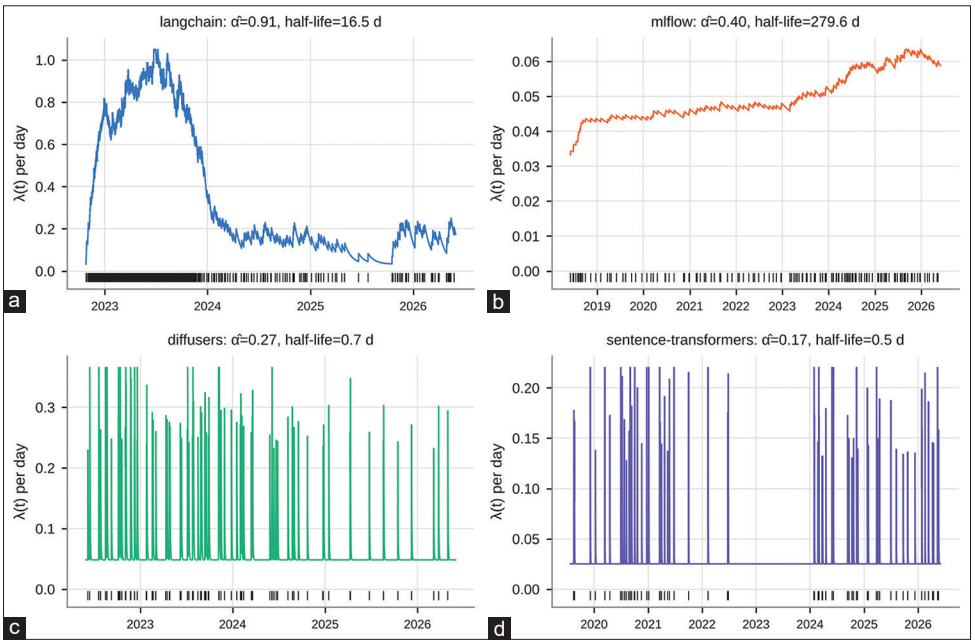
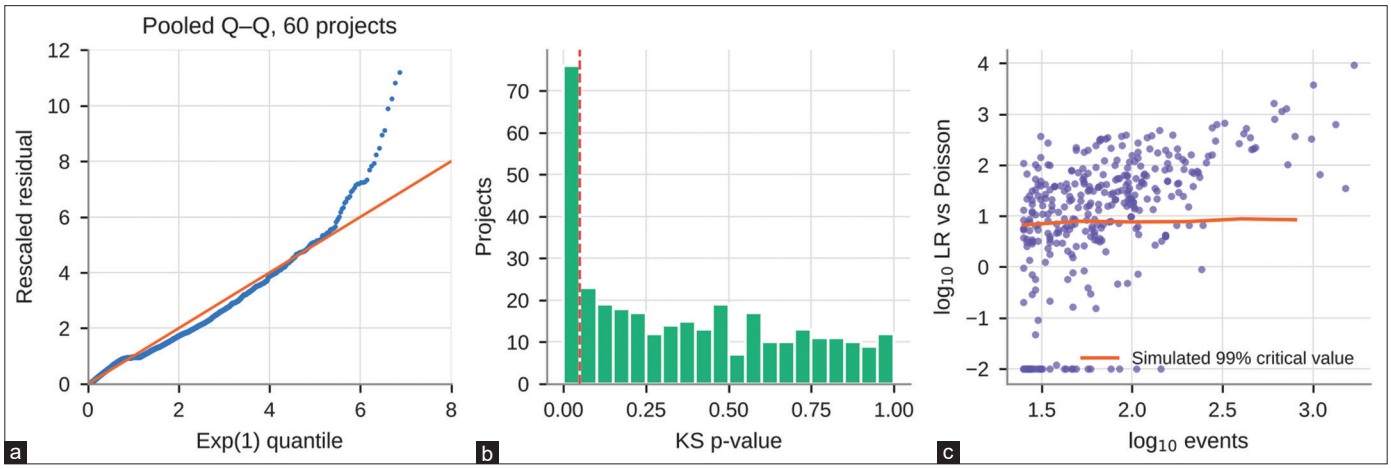


Figure 5: Goodness of fit. (a) Pooled Q-Q plot of time-rescaled residuals for 60 randomly chosen projects against unit-exponential quantiles. (b) Kolmogorov-Smirnov P-values; the dashed line marks 0.05. (c) Likelihood-ratio statistic against the number of events, with the simulated 99% critical value of Equation (5)



read as acceleration when the placebo intervals overlap it and the age effect of Section 4.3 predicts a flattening decline anyway. With the null level and slope terms of Table 3 added, H2 finds no support at either level.

4.5. Concentration of the Dependency Network

After marker evaluation the runtime-dependency closure of the AI projects has 2,226 nodes and 8,803 edges (Table 8). A median AI project pulls in 23 packages, with an interquartile range of 4-47; 13.7% declare no runtime dependency. Direct in-degree from AI projects follows the discrete power law of Equation (2) from $k_{min} = 1$ with $\hat{\gamma} = 1.955$ (SE 0.028), and the bootstrap finds no evidence against the fit (P = 0.47). The exponential loses decisively ($z = 7.64$). Against the log-normal the Vuong statistic is −0.76 (P = 0.45), so the data cannot tell the two laws apart; the fitted log-normal has a scale parameter so large that over the observed range of degrees it behaves almost like a power law itself. In-degree over all edges of the closure gives nearly the same exponent, 1.943 (Figure 8).

Direct use understates concentration (Table 9). Typing-extensions is listed directly by 99 AI projects and sits in the closure of 432, or 65.9%, while numpy, the package most used directly (288 projects), reaches 58.2%. Idna and charset-normalizer are each listed directly by three AI projects, yet through requests they reach half and two-fifths of them. Over all nodes the Gini coefficient of reach is 0.846, 87 nodes reach at least a tenth of the AI projects, and the maximum core has 113 members, half of which are AI projects themselves (Figure 9).

Removal makes the concentration concrete (Figure 10). Taking out the package with the largest reach exposes 65.9% of AI projects, three removals expose 80.2%, and after that the curve flattens,

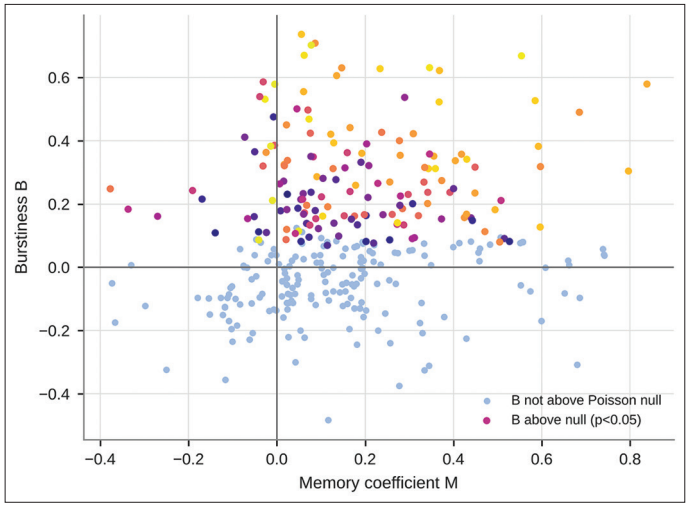
to 80.6% at ten removals and 83.8% at one hundred. Random removal of ten packages exposes 11.8%; of one hundred, 57.3%. What caps the targeted curves just above 80% is the 13.7% of projects with no dependencies, along with a few whose closures are idiosyncratic. By the third removal, ranking by PageRank or by in-degree gives the same picture. H_3 is supported.

Tempo and network position are related in one direction only. Projects with larger closures release more often (Spearman $P = 0.32$, $P < 0.001$), as frameworks that integrate many libraries also chase their updates, while PageRank, which measures being depended on, has a weak negative relation to the release rate ($\rho = -0.14$, $P = 0.013$), and neither measure relates to $\hat{\alpha}$ (Figure 11). Packages at the bottom of the stack change less often than the frameworks built on them.

4.6. Release Semantics and Maturation

Of the 40,182 classified final releases from 2018 onward, 73.4% are patches, 20.9% minor releases, 1.1% major releases and 4.6% backports (Table 10 and Figure 12). Major releases fell from 1.33% of the total before the break to 1.02% after it. The GEE coefficient on the post indicator is −0.238 ($P = 0.010$), and because the linear year trend for major releases is negative and not significant ($P = 0.195$), a step describes the drop better than a slope. Minor and patch shares hold steady. The backport share fell from 12.4% to 1.6%, but one project, relevanceai-dev, uploaded 763 of the 889 backports recorded in 2022 and dominates the pre-break figure. Successive major releases of a project are a median of 207 days apart (interquartile range 83-476).

Figure 6: Burstiness B against memory M for the 336 modelled projects. Coloured points have B above its Poisson null at 5%, with colour giving $\hat{\alpha}$; pale points do not



Maturation sped up (Table 11 and Figure 13). Of 486 eligible projects, 251 reached a 1.0 release before the end of the window. Median waiting times were 1,171 days for the 2018-2019 cohort and 1,456 days for 2020-2021, against 819 days for 2022-2023 and 697 for projects born from 2024, and within two years of birth 39% of the 2018-2019 cohort had reached 1.0 against 58% of the 2024-2026 cohort. Across the four cohorts the log-rank test gives $P = 0.078$. In the stratified Cox model of Equation (12), birth after 30 November 2022 multiplies the hazard of reaching 1.0 by 1.41 (95% CI 1.08–1.85, $P = 0.012$) with downloads and classifier count held fixed (Table 12). Both controls raise the hazard too, and neither violates proportional hazards ($P = 0.83$ and 0.56). H_4 is supported. One caveat applies: Early cohorts may look slow partly because fast-maturing projects from 2018 that later lost popularity are absent from the 2026 list.

Support for old Python versions lingers. Of final releases since 2018, 77.6% declare requires_python, and 35.7% of those admit a version that had already reached end of life at upload, with a median lag of 10.9 months past that date (Figure 14). Each end-of-life date produces a spike. The share of releases admitting a newly retired version jumps on that date and decays over the following year, and the GEE year trend in legacy support is positive (0.434 on the logit scale, $P < 0.001$), while the post-break indicator adds nothing ($P = 0.817$); maintainers drop retired versions more slowly than the end-of-life calendar retires them, and the gap has widened year by year.

Table 6: OLS regressions of logit $\hat{\alpha}$ and of the log annual release rate on project covariates (336 projects, HC3 standard errors)

Covariate	Logit : $\hat{\alpha}$ b	SE	P	Log rate: b	SE	P
Intercept	1.027	1.430	0.473	4.239	0.382	<0.001
$\log_{10}$ downloads	−0.331	0.240	0.168	0.016	0.059	0.780
Years observed	0.130	0.065	0.046	−0.266	0.018	<0.001
Classifier count	−0.051	0.030	0.094	−0.012	0.013	0.343
R^2	0.029			0.376		

Table 7: Break tests of Equation (7) at the ChatGPT date and at four placebo dates. A project is testable at a date when it has at least 12 final releases on each side

Break date	Projects	Median $\hat{\mu}_2 / \hat{\mu}_1$	$\hat{\mu}_2 > \hat{\mu}_1$ (%)	Wilcoxon p(μ)	Median $\Delta\hat{\alpha}$	Wilcoxon P (α)	LR>5.99 (%)
November 30, 2019	41	0.18	4.9	<0.001	0.291	<0.001	82.9
November 30, 2020	84	0.31	6.0	<0.001	0.133	<0.001	70.2
November 30, 2021	113	0.47	18.6	<0.001	0.034	<0.001	62.8
November 30, 2022 (ChatGPT)	128	0.67	24.2	<0.001	0.000	0.236	55.5
November 30, 2023	132	0.62	27.3	<0.001	0.013	0.018	59.8

Figure 7: Break tests. (a) Median ratio of post-break to pre-break baseline at the ChatGPT date and at four placebo dates, with 95% bootstrap intervals over projects. (b) Pre-break against post-break baseline at 30 November 2022; the line marks equality, and points at the bottom are projects whose post-break baseline hit the lower bound

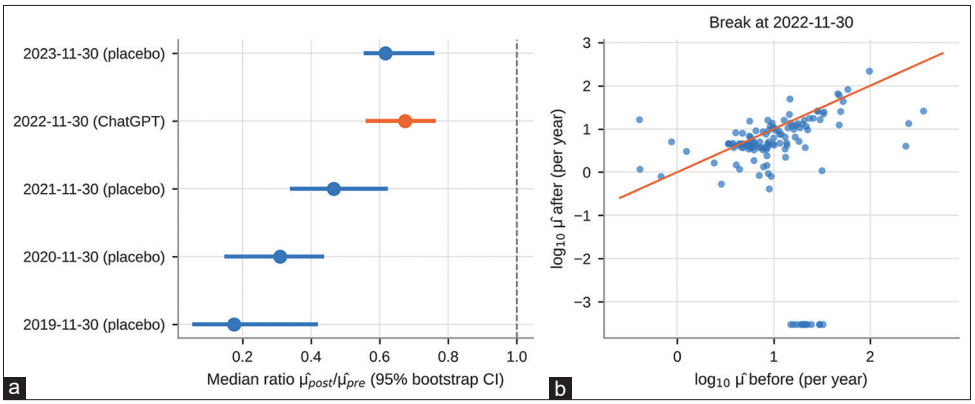


Table 8: Structure of the runtime-dependency closure and discrete power-law fits to in-degree

Statistic	AI-direct in-degree	Closure in-degree
Nodes/edges	2,226/8,803	
Median closure size of an AI project (Q1-Q3)	23 (4-47)	
AI projects with no runtime dependency	13.7%	
Maximum *k*-core (members)	11 (113)	
Gini coefficient of transitive reach	0.846	
Nodes reaching at least 10% of AI projects	87	
Power-law exponent $\hat{\gamma}$ (SE)	1.955 (0.028)	1.943 (0.022)
k_{min}/tail size	1/1,195	1/1,795
KS distance *D*	0.0109	0.0089
Bootstrap goodness-of-fit P (200 draws)	0.47	0.51
Vuong *z*, power law versus exponential	7.64 (P<0.001)	8.86 (P<0.001)
Vuong *z*, power law versus log-normal	−0.76 (P=0.45)	−0.47 (P=0.64)

4.7. Validation and Sensitivity

At the sizes found in the data, the simulation study supports the estimator (Table 13 and Figure 15). With $\beta = 1$ per day and 150 expected events, the bias of $\hat{\alpha}$ stays within 0.031 of zero and its RMSE lies between 0.045 and 0.088. Slow kernels with strong excitation are the hard case. At $\alpha = 0.8, \beta = 0.1$ and 50 events, $\hat{\alpha}$ is too low by 0.19 and $\hat{\mu}$ too high by 63%, because a short window cannot separate a long-memory cascade from a higher baseline; with a median of 62 events in the modelled projects, their $\hat{\alpha}$ is, if anything, conservative. Detection at the 1% level reaches at least 94% for $\alpha \geq 0.5$ at 150 events and falls to 10% for weak, slow excitation at 50 events, the one cell of the grid in which the test is close to useless. The break test has a size of 6.5% at the nominal 5% and modest power: 19% for a 50% rise in the baseline, 61% for a doubling and 32% for a fall to half.

Three changes of specification leave the main results in place (Table 14). Including pre-releases raises the median $\hat{\alpha}$ to 0.313 and shortens the median half-life to 34 h, since release candidates and their finals arrive close together. With a minimum of 50 events, 202 projects remain and their median $\hat{\alpha}$ is 0.422, in line with the downward bias the simulation found for short series. A window starting in 2021 gives 0.272. Throughout, between 60% and 78% of projects reject a Poisson stream.

5. DISCUSSION

5.1. What the Hypotheses Show

Table 15 puts the four results side by side. Release streams of AI software are self-exciting, the dependency network is concentrated and new projects mature faster than they used to, yet no discontinuity in how often established projects release can be traced to late 2022. The composition of the ecosystem is what changed. Active projects kept multiplying, so the ecosystem as a whole released more, while each established project released somewhat less as it aged and each new entrant moved to 1.0 sooner. Entry and a secular trend of about 10% a year in releases per active project carry the tempo of innovation in this layer, and that trend was already in place in 2018.

This reading agrees with accounts of AI as a general-purpose technology, which predict diffusion through new applications more than a jump in the activity of existing producers (Damioli et al., 2025; Leon, 2026). Newcomers carry the diffusion. It also matches the pace that Tang et al. (2020) measured in publications, where the time to adoption shortened over decades without a single break. Where accounts of a generative-AI surge are right, our data put the surge at the entry margin, in new projects such as langchain and llama-index, whose branching ratios are 0.91 and 0.88.

5.2. Implications for Intelligent Systems Engineering

Self-excitation changes how upgrade work should be planned. Under a Poisson stream with rate $\mu/(1-\alpha)$ the probability of k releases in a week depends on that rate alone; under a Hawkes stream with the same long-run rate, releases come in bursts, quiet weeks are more frequent and busy weeks busier, and the variance of the count in a window of length $w \gg 1/\beta$ grows by roughly $1/(1-\alpha)^2$, a factor of 2.0 at the median $\hat{\alpha}$. Fixed upgrade cycles suffer most. A team that pins versions and reviews upgrades on a schedule will meet the burst all at once. Hot-fix cascades, the regime of 45.5% of projects, die out within a day of a faulty upload,

Table 9: The ten packages with the largest transitive reach into AI projects

Package	Direct AI dependants	Transitive AI dependants	Share of AI projects (%)	PageRank	Core number
Typing-extensions	99	432	65.9	0.0325	11
Numpy	288	382	58.2	0.0388	11
Idna	3	330	50.3	0.0054	8
Packaging	99	297	45.3	0.0106	11
Certifi	22	290	44.2	0.0083	11
Urllib3	33	273	41.6	0.0083	11
Pyyaml	101	269	41.0	0.0069	11
Charset-normalizer	3	266	40.5	0.0027	5
Requests	116	264	40.2	0.0097	11
h11	5	215	32.8	0.0021	6

Figure 8: Complementary cumulative distribution of in-degree on log-log axes with the fitted discrete power law. (a) Direct requirements from AI projects. (b) All edges in the closure

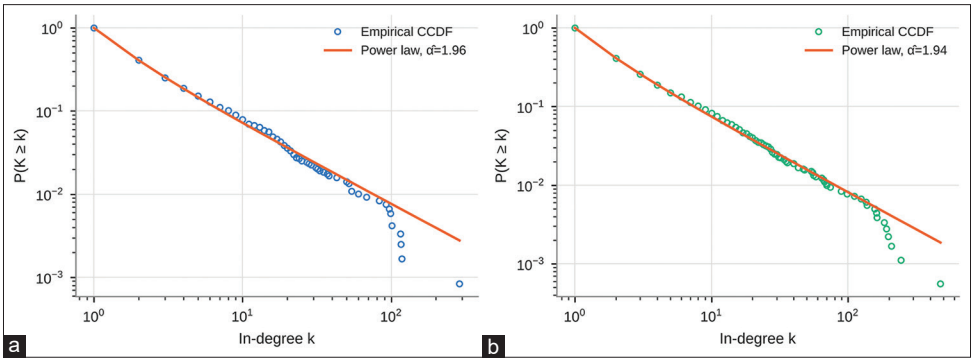


Figure 9: Core of the dependency network: the 60 packages with the largest transitive reach and the 25 AI projects most used by other AI projects, drawn with a Kamada-Kawai layout. Node area is proportional to transitive reach

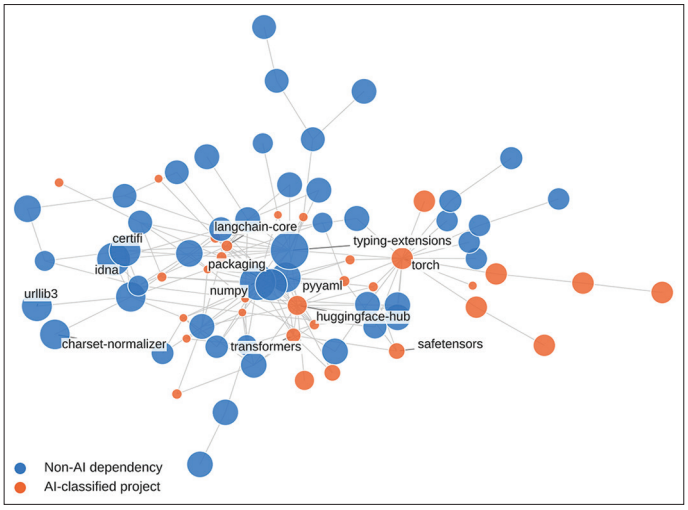
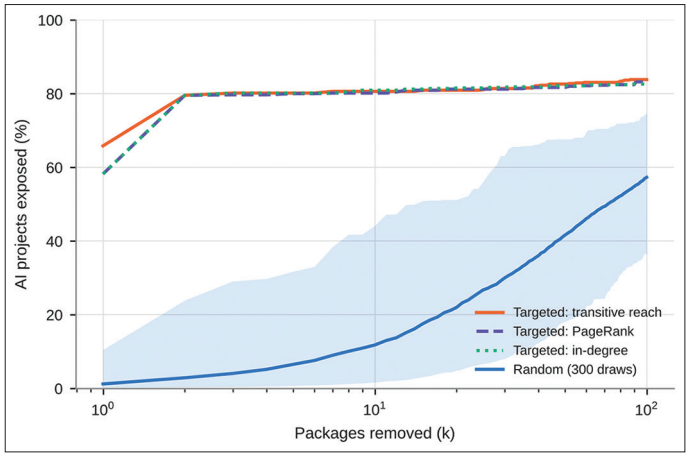


Figure 10: Share of AI projects whose closure contains at least one removed package, as packages are removed in order of transitive reach, PageRank or in-degree, and in random order (mean and 95% band over 300 draws)



so a team that waits a day before taking a new release will often skip the faulty upload altogether and install its fix instead.

Concentration means that the exposure of an intelligent system is set mostly by packages that are not AI packages at all. The top of Table 15 reads like a list of plumbing: Typing back-ports, numerical arrays, character encodings, HTTP. A breaking change or a compromised release in one of them reaches two-thirds of AI projects, and three such packages reach four-fifths, a pattern that fits the leverage measured for the wider Python ecosystem (Paramitha and Massacci, 2023) and the heavy-tailed in-degree found across packaging ecosystems (Decan et al., 2019). For developers of intelligent systems the practical consequence is to audit and pin the shared substrate as carefully as the model framework. That costs less than it sounds. Hubs release less often than the frameworks built on them (Section 4.5).

Fewer major releases after 2022, together with faster arrival at 1.0, suggest that AI projects now commit to stable interfaces earlier and then evolve them through minor and patch releases. Whether they also break clients less often is a separate question, and since Ochoa et al. (2022) found semantic-versioning discipline to vary

Figure 11: Release tempo and position. (a) Annual release rate against the size of a project's dependency closure. (b) Annual release rate against years observed in the window. Lines are least-squares fits

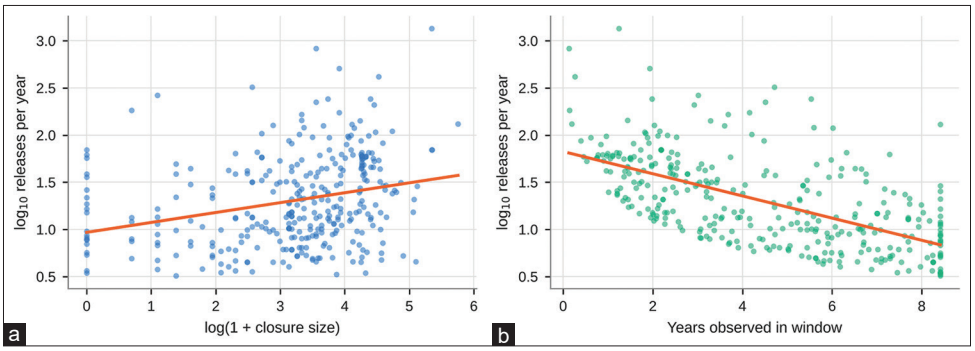


Figure 12: Semantics of final releases by year. (a) Shares of patch, minor, major and backport releases. (b) Share of major releases

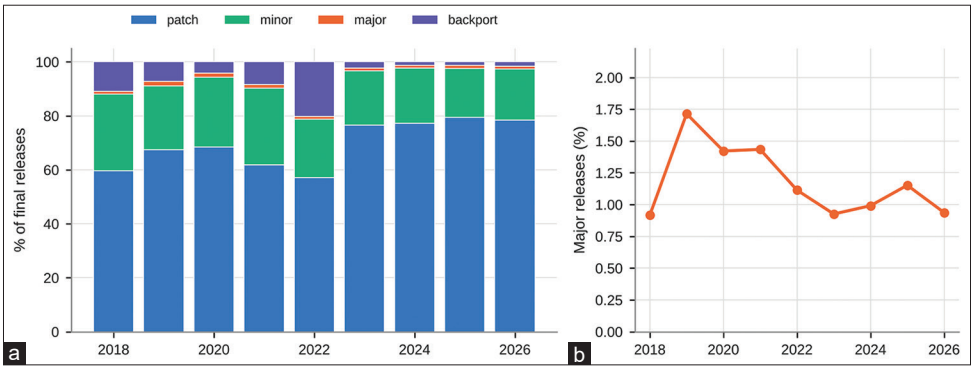


Table 10: Release types before and after 30 November 2022 among final releases from 2018 onward, with GEE coefficients (binomial, exchangeable within project) for the post indicator and for a linear year trend

Release type	Before break (%)	After break (%)	GEE post (SE)	P	GEE year (SE)	P
patch	61.4	78.0	−0.009 (0.093)	0.926	−0.018 (0.020)	0.371
minor	24.9	19.4	0.090 (0.104)	0.390	0.041 (0.022)	0.067
major	1.3	1.0	−0.238 (0.093)	0.010	−0.031 (0.024)	0.195
backport	12.4	1.6	–	–	–	–

Table 11: Kaplan-Meier summaries of the time from first upload to a first 1.0 release by birth cohort; *F (t)* is the cumulative share reaching 1.0

Cohort	Projects	1.0 reached	Median days	*F* (1 year)	*F* (2 year)	*F* (3 year)
2018-19	70	43	1171	0.19	0.39	0.49
2020-21	116	65	1456	0.22	0.37	0.45
2022-23	122	66	819	0.31	0.48	0.54
2024-26	178	77	697	0.34	0.58	–

Table 12: Cox model of the hazard of a first 1.0 release, stratified by runtime-dependency count (486 projects, 251 events; concordance 0.59)

Covariate	Hazard ratio	95% CI	P	PH test P
Born after 30 Novemebr 2022	1.41	1.08-1.85	0.012	0.47
$\log_{10}$ downloads	1.34	1.16-1.56	<0.001	0.83
Classifier count	1.07	1.04-1.09	<0.001	0.56

widely between libraries, the shift seen here should be checked against breaking-change data for AI libraries. It may equally reflect the reputational value of a 1.0 label for projects competing for adoption.

5.3. Implications for the Measurement of Innovation

Release metadata measure software innovation in a way that is timely, fine-grained and public, and they can sit beside the cost-based measures of Korkmaz et al. (2024) and the venture-based measures of Wright et al. (2023). Two cautions come out of our results. Counts of releases mix a steady baseline with self-excited bursts, which the branching ratio of Equation (1) separates, and a rise in counts driven by α signals clustered rework more than new functionality. Comparisons before and after an event on ageing units also need placebo dates. Without them our break test would have reported a slowdown at the ChatGPT date in three-quarters of established projects, significant in more than half. The design costs little, and it belongs with any claim of a technology-driven shift measured on software activity, claims

Figure 13: Maturation. (a) Kaplan-Meier curves of the probability that a project has not yet released 1.0, by birth cohort. (b) Hazard ratios from the stratified Cox model with 95% intervals

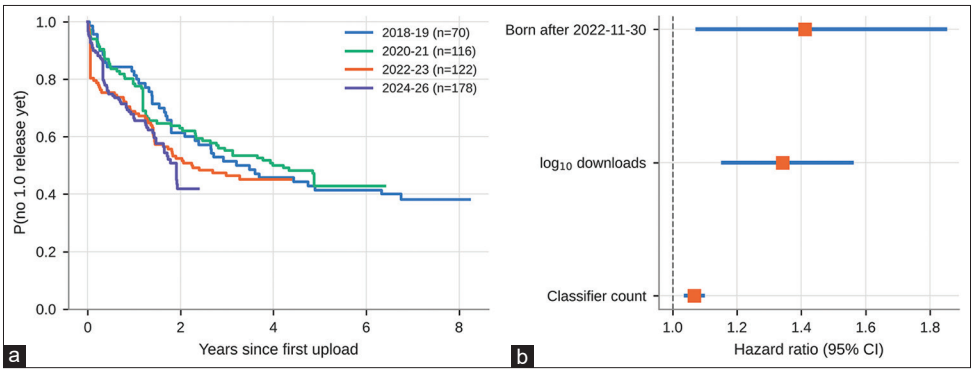


Figure 14: Legacy Python support. (a) Share of final releases, by ½-year, whose requires_python admits a Python version past its end of life; dotted lines mark the end-of-life dates of 2.7-3.9. Before 2020 the oldest maintained version was 2.7, so no release can qualify. (b) Months since the end of life of the oldest admitted version, among releases with legacy support

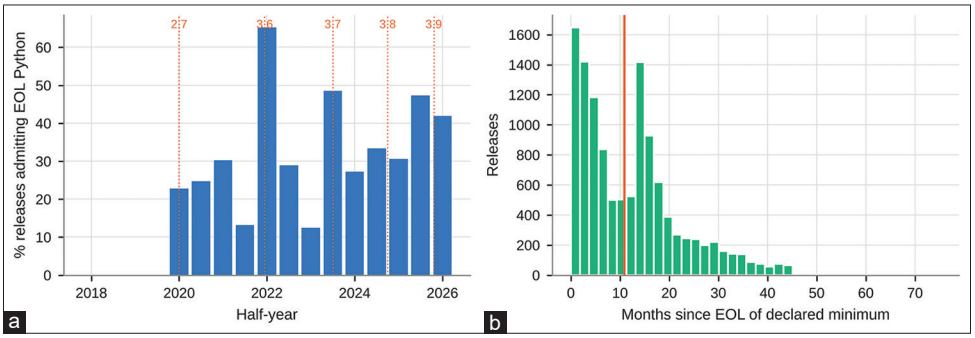
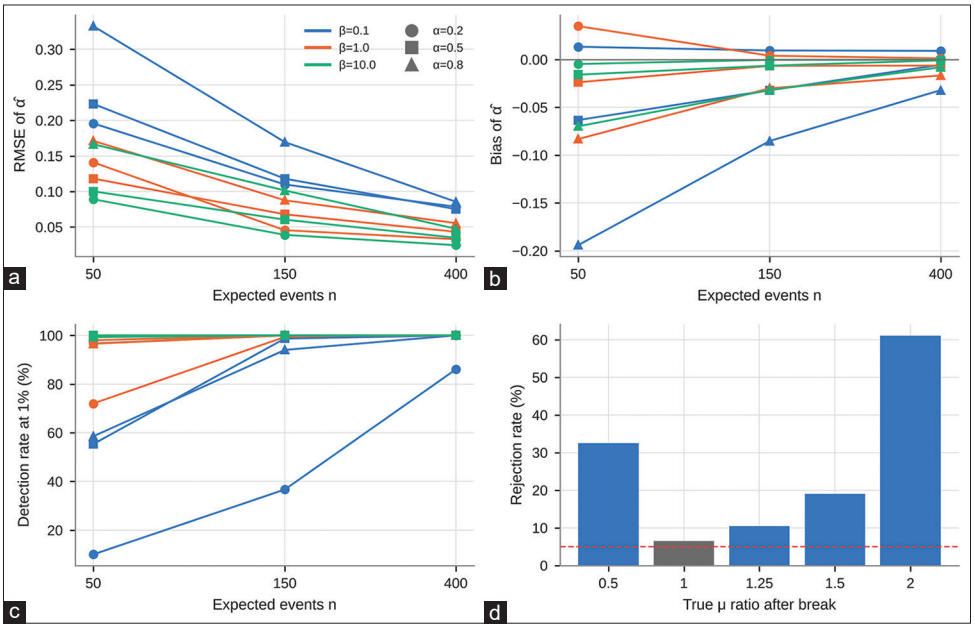


Figure 15: Monte Carlo validation. (a) RMSE and (b) bias of $\hat{\alpha}$ by expected number of events, true α (marker) and β (colour). (c) Share of samples in which the likelihood-ratio test detects excitation at 1%. (d) Rejection rate of the break test at 5% by true ratio of post-break to pre-break baseline; the dashed line marks 5%



about open foundation-model ecosystems included (Shi and Jiang, 2025; Vake et al., 2025).

For innovation policy the dependency results point to a familiar public-goods question. The packages on which AI projects rely most are general-purpose libraries outside the AI segment. Whether their maintenance is funded in proportion to their reach, our data cannot say; the reach itself shows where support for open-source infrastructure would protect the largest number of AI applications per unit of effort.

Table 13: Monte Carlo performance of the Hawkes estimator for β=1 per day and μ=0.05 per day (150 replicates per cell). Detection uses the simulated 99% critical value

True α	Expected n	Bias $\hat{\alpha}$	RMSE $\hat{\alpha}$	Rel. bias $\hat{\mu}$	Median rel. bias $\hat{\beta}$	Detected at 1%
0.2	50	+0.035	0.141	−0.034	+0.088	72.0
0.2	150	+0.004	0.045	−0.010	+0.023	99.3
0.2	400	+0.001	0.033	−0.001	+0.020	100.0
0.5	50	−0.024	0.118	−0.029	+0.007	98.0
0.5	150	−0.007	0.068	+0.008	+0.022	100.0
0.5	400	−0.006	0.043	+0.006	+0.030	100.0
0.8	50	−0.083	0.171	+0.061	+0.052	96.6
0.8	150	−0.030	0.088	−0.003	+0.011	100.0
0.8	400	−0.017	0.056	+0.006	−0.015	100.0

Table 14: Sensitivity of the Hawkes estimates to the sample definition

Specification	Projects	Median $\hat{\alpha}$	Mean $\hat{\alpha}$	Median half-life (h)	LR >99% value
Main specification	336	0.289	0.381	58.6	64.9
Pre-releases included	381	0.313	0.396	34.4	74.0
At least 50 events	202	0.422	0.446	54.9	78.2
Window from 1 Jan 2021	302	0.272	0.362	35.1	60.3

Table 15: Summary of hypothesis tests

Hypothesis	Main evidence	Outcome
H_1 Self-excitation	64.9% reject Poisson at 99%; median $\hat{\alpha}$ =0.289; B and M above null in 46.1% and 40.8%	Supported
H_2 Generative-AI acceleration	Baseline ratio 0.67 at the ChatGPT date against 0.18-0.62 at placebos; no level or slope change in releases per active project	Not supported
H_3 Concentrated dependency	Power law with $\hat{\gamma}$ =1.955; one removal exposes 65.9% and three removals 80.2%	Supported
H_4 Faster maturation	Hazard ratio 1.41 (1.08-1.85) for projects born after the break	Supported, subject to survivorship

5.4. Limitations

The sampling frame is a 2026 snapshot of popularity. It holds the survivors, so the cross-cohort comparisons of Sections 4.1 and 4.6 are conditioned on survival to 2026, and if projects that reached 1.0 in 2019 and then lost their users were common, their absence biases early cohorts toward slow maturation. Yearly download snapshots would remove that bias. Because the AI classifier is self-declared and misses major libraries (Section 3.1), the sample is the part of the ecosystem that labels itself AI. Dependencies come from the latest release only. The network is therefore a 2026 cross-section; how concentration evolved could be measured by parsing the requirement metadata of every historical release, which PyPI stores in the distribution files.

The exponential kernel misfits the longest pauses in about a fifth of projects, and eighteen streams, two of them nightly builds, sit at the stationarity bound. A sum-of-exponentials or power-law kernel, or a baseline that declines with age inside the Hawkes model, would show whether the age effect and the excitation can be separated. Power is a weak point. A 25% rise in the baseline of a typical project would be detected about one time in ten (Section 4.7), so the absence of evidence for H_2 at the project level rests more on the placebo comparison and the aggregate series than on individual tests. Whether excitation crosses the network, from a framework's releases to those of its dependants, needs a marked or multivariate Hawkes process; the cross-sectional correlations reported here cannot settle it.

6. CONCLUSION AND FUTURE WORK

AI-classified Python packages release in self-exciting streams. Two-thirds of projects reject a Poisson process, a typical release triggers 0.29 further releases, and the triggering fades within a day in nearly half of projects and lasts more than a month in a quarter. Release rates decline by about 23% for each year a project ages. That decline explains the apparent slowdown at the public release of ChatGPT, since placebo dates produce larger declines. Across the ecosystem, releases per active project have grown by about 10% a year since 2018 without a break in level or slope, and growth since 2022 rests on three things: Entry, faster maturation of new projects (which reach 1.0 with a hazard 41% higher than earlier cohorts), and a dependency network in which a handful of general-purpose packages reach most AI projects.

For builders of intelligent systems the results argue for upgrade policies that expect bursts and for audits of the shared substrate beneath the AI frameworks. Students of innovation can use release metadata to measure the tempo of a technology, provided self-excitation and ageing are modelled and placebo dates are used. Three extensions follow from the limitations. Yearly snapshots of the download list would take survivorship out of the cohort comparisons. With requirement metadata from historical releases, the network becomes a panel. And a multivariate Hawkes model over framework-dependency pairs would show how far a release in one package travels through the stack.

6.1. Data Availability

The sampling frame is the 2026.05 release of the top PyPI packages data set (van Kemenade, 2026), and package metadata were retrieved from the public PyPI JSON interface (Python Software Foundation, 2026). The processed release tables, the dependency closure, the estimation scripts and the figure code are supplied as supplementary material. Every number in the tables is produced by those scripts from the stored data.

REFERENCES

Carayannis, E.G., Preissler, S. (2025), The quintuple innovation helix plus AI: A prospective retrospective approach and the use case of semiconductors. Technological Forecasting and Social Change, 216, 124122.

Da Silva, A.C.B.G., De Carneiro, G.F., Brito e Abreu, F., Monteiro, M.P. (2017), Frequent releases in open source software: A systematic review. Information, 8(3), 109.

Damioli, G., Van Roy, V., Vertesy, D., Vivarelli, M. (2025), Is artificial intelligence leading to a new technological paradigm? Structural Change and Economic Dynamics, 72, 347-359.

Decan, A., Mens, T., Grosjean, P. (2019), An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering, 24(1), 381-416.

Ganeshbabu, B., Liu, X., Matono, A., Kim, K.S., Shen, X. (2026), Scientific impact prediction via virtual geography Hawkes process. Applied Sciences, 16(4), 2085.

Jang, H.J., Woo, H.G., Lee, C. (2017), Hawkes process-based technology impact analysis. Journal of Informetrics, 11(2), 511-529.

Kang, D., Kang, T., Jang, J. (2023), Papers with code or without code? Impact of GitHub repository usability on the diffusion of machine learning research. Information Processing and Management, 60(6), 103477.

Korkmaz, G., Santiago Calderón, J.B., Kramer, B.L., Guci, L., Robbins, C.A. (2024), From GitHub to GDP: A framework for measuring open source software innovation. Research Policy, 53(3), 104954.

Koyama, S. (2025), Coarse-grained Hawkes processes. Entropy, 27(6), 555.

Leon, M. (2026), Generative AI as a general-purpose technology: Foundations, applications, and labor market implications through 2030. Big Data and Cognitive Computing, 10(3), 69.

Li, H., Bezemer, C.P. (2025), Bridging the language gap: An empirical study of bindings for open source machine learning libraries across software package ecosystems. Empirical Software Engineering, 30, 6.

Liu, J., Huang, Q., Xia, X., Shihab, E., Lo, D., Li, S. (2021), An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks. Empirical Software Engineering, 26, 16.

Malem-Shinitski, N., Ojeda, C., Opper, M. (2022), Variational Bayesian inference for nonlinear Hawkes process with Gaussian process self-effects. Entropy, 24(3), 356.

Ming, X., Wang, Q., Liu, Y. (2025), The performance implications of R&D collaborations on artificial intelligence. Technovation, 145, 103271.

Mu, Y., Wu, W. (2026), Hypothesis tests for comparing point processes. Mathematics, 14(4), 727.

Ochoa, L., Degueule, T., Falleri, J.R., Vinju, J. (2022), Breaking bad? Semantic versioning and impact of breaking changes in Maven Central. Empirical Software Engineering, 27, 61.

Ogura, H., Hanada, Y., Kondo, M. (2026), Power-law behavior in the inter-event times of word occurrences. Applied Sciences, 16(6), 2818.

Paramitha, R., Massacci, F. (2023), Technical leverage analysis in the Python ecosystem. Empirical Software Engineering, 28, 139.

Python Software Foundation. (2026), PyPI JSON API [Data Set]. Available from: https://docs.pypi.org/api/json

Raschka, S., Patterson, J., Nolet, C. (2020), Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information, 11(4), 193.

Roberts, D.L., Candi, M. (2024), Artificial intelligence and innovation management: Charting the evolving landscape. Technovation, 136, 103081.

Seol, Y. (2022), Non-Markovian inverse Hawkes processes. Mathematics, 10(9), 1413.

Shi, J., Jiang, S. (2025), Toward open-source foundation model ecosystem: Impact evaluation framework and promotion mechanism. Technological Forecasting and Social Change, 221, 124328.

Tang, X., Li, X., Ding, Y., Song, M., Bu, Y. (2020), The pace of artificial intelligence innovations: Speed, talent, and trial-and-error. Journal of Informetrics, 14(4), 101094.

Truong, Y., Papagiannidis, S. (2022), Artificial intelligence as an enabler for innovation: A review and future research agenda. Technological Forecasting and Social Change, 183, 121852.

Vake, D., Šinik, B., Vičič, J., Tošić, A. (2025), Is open source the future of AI? A data-driven approach. Applied Sciences, 15(5), 2790.

Van Kemenade, H. (2026), Top PyPI Packages (Version 2026.05) [Data Set]. GitHub. Available from: https://github.com/hugovk/top-pypi-packages

Vanni, F., Lambert, D. (2024), Aging renewal point processes and exchangeability of event times. Mathematics, 12(10), 1529.

Wright, N.L., Nagle, F., Greenstein, S. (2023), Open source software and global entrepreneurship. Research Policy, 52(9), 104846.

Zhang, B., Wang, H. (2021), Network proximity evolution of open innovation diffusion: A case of artificial intelligence for healthcare. Journal of Open Innovation Technology Market and Complexity, 7(4), 222.

Zhang, L.N., Liu, J.W., Song, Z.Y., Zuo, X. (2021), Universal transformer Hawkes process with adaptive recursive iteration. Engineering Applications of Artificial Intelligence, 105, 104416.

Zhao, Z., Chen, Y., Bangash, A.A., Adams, B., Hassan, A.E. (2024), An empirical study of challenges in machine learning asset management. Empirical Software Engineering, 29, 98.

Zhou, W.T., Kang, Z., Tian, L., Su, Y. (2023), Intensity-free convolutional temporal point process: Incorporating local and global event contexts. Information Sciences, 646, 119318.