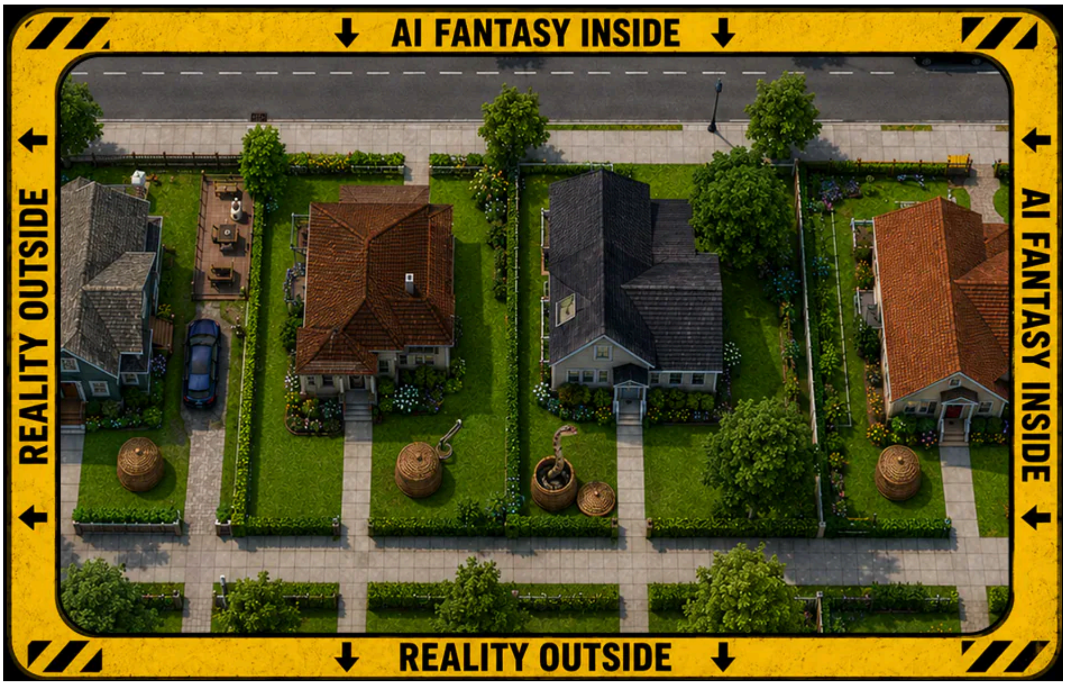
Python Package Installations as an Index of Local AI Adoption

Originally published at https://www.unite.ai/python-package-installations-as-an-index-of-local-ai-adoption/

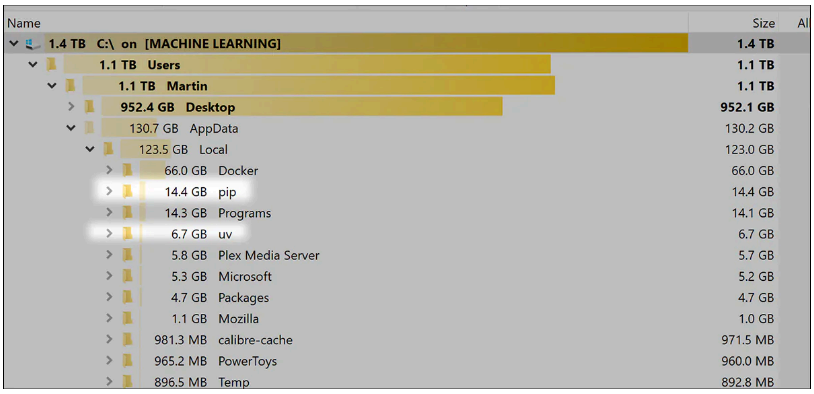


The hard-to-quantify rise of package downloads for Python spell out the story of AI diffusion, if you know where to look.

Opinion Anyone who has explored the potential of local AI will by now have lost a scary amount of disk space not only to the gargantuan models involved, but also to the related Python and similar systems that orchestrate inference, training, and hundreds of other potential uses – not least, in generating local automation systems.

This last use case is the most common across the various machines on my LAN, where I have used AI to set up 'flat' Python-based automated routines, such as backup, periodic checks, the administration of my website, and a dozen other uses.

Practically everything I want to achieve during these coding sessions seems to require a new Python library, to the extent that PIP (the Package Installer for Python) is one of the biggest space-hogs across my LAN machines:



Name	Size
1.4 TB C:\ on [MACHINE LEARNING]	1.4 TB
1.1 TB Users	1.1 TB
1.1 TB Martin	1.1 TB
952.4 GB Desktop	952.1 GB
130.7 GB AppData	130.2 GB
123.5 GB Local	123.0 GB
66.0 GB Docker	66.0 GB
14.4 GB pip	14.4 GB
14.3 GB Programs	14.1 GB
6.7 GB uv	6.7 GB
5.8 GB Plex Media Server	5.7 GB
5.3 GB Microsoft	5.2 GB
4.7 GB Packages	4.7 GB
1.1 GB Mozilla	1.0 GB
981.3 MB calibre-cache	971.5 MB
965.2 MB PowerToys	960.0 MB
896.5 MB Temp	892.8 MB

PIP's diminutive presence in the CLI and GUIs belie its impact on your filesystem. The cache can reach very high, even system-crippling file-sizes.

Experimenting with local AI reverses the significant redundant luxury of local computers, prior to the AI hardware grab of the current era. If you had a generous RAM allowance, it will be eaten up in GPU offloading; a terabyte of hard disk space, likewise, will quickly fill up with high volume models, quantized or not; and even if, as I did, you have outfitted yourself with 24GB of VRAM (a new minimum..?), the system will still need a variety of shortcuts and tricks to run inference at reasonable time-frames.

Installed PyPi (Python Package Index) packages do not show up in standard 'Installed programs' GUIs in a mainstream OS, and need to be explicitly elicited, via Python, for instance with the PowerShell command `pip freeze --all`.

The list can be surprisingly long:

Package	Version	Package	Version	Package	Version
absl-py	2.3.0	Jinja2	3.1.4	pip	25.1.1
attrs	25.3.0	jsonpatch	1.33	platformdirs	4.3.8
beets	2.5.1	jsonpointer	3.0.0	protobuf	4.25.8
certifi	2025.11.12	kiwisolver	1.4.8	psutil	7.2.1
cffi	1.17.1	lap	0.5.12	pybrisque	1.0
charset-normalizer	3.4.4	lazy_loader	0.4	pycparser	2.22
click	8.2.1	libsvm	3.23.0.4	pyparsing	3.2.3
colorama	0.4.6	Markdown	3.8.2	python-dateutil	2.9.0.post0
confuse	2.1.0	MarkupSafe	2.1.5	PyYAML	6.0.2
contourpy	1.3.2	matplotlib	3.10.3	pyyaml_env_tag	1.1
cycler	0.12.1	mediafile	0.13.0	requests	2.32.5
dlib	20.0.0	mediapipe	0.10.21	scikit-image	0.25.2
face-recognition	1.3.0	mergedeep	1.3.4	scipy	1.16.0
face_recognition_models	0.3.0	mkdocs	1.6.1	sentencepiece	0.2.0
filelock	3.13.1	mkdocs-get-deps	0.2.0	setuptools	65.5.0
filetype	1.2.0	ml_dtypes	0.5.1	six	1.17.0
flatbuffers	25.2.10	mpmath	1.3.0	sounddevice	0.5.2
fonttools	4.58.4	musicbrainzngs	0.7.1	sympy	1.13.1
fsspec	2024.6.1	mutagen	1.47.0	tifffile	2025.6.11
ghp-import	2.1.0	networkx	3.3	torch	2.5.1+cu118
idna	3.11	numpy	2.1.2	torchvision	0.20.1+cu118
image-quality	1.2.7	opencv-contrib-python	4.11.0.86	tqdm	4.67.1
imageio	2.37.0	opencv-python	4.11.0.86	typing_extensions	4.12.2
internetarchive	5.7.1	opt_einsum	3.4.0	Unidecode	1.4.0
jax	0.6.2	packaging	25.0	urllib3	2.6.2
jaxlib	0.6.2	pathspec	0.12.1	watchdog	6.0.0
jellyfish	1.2.1	pillow	11.0.0		

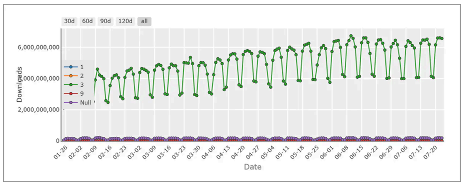
A dump of PyPi installations on one of my busier machines. Some of these tiny words hide gigabytes of disk space – notably anything Torch/PyTorch-related.

In short, local AI can take your computing experience back to the early 1990s, just slightly past the period where a program like Word had to be loaded into a few KB of RAM, one large floppy disk at a time; but before hard disk allocations gave you any room to breathe.

Finding the Undercurrents

I was curious if an upsurge in PyPi installations over the last few years could serve as an indices for local AI adoption, which is, in general, otherwise quite hard to track, except through intuition when participating in various AI communities and noting which platforms and packages gain attention.

A casual glance at the PyPi stats website shows a disappointingly narrow range of dates, all of which, however, indicate constant upward growth in downloads:



Rise in download across all Python packages since the start of 2026 – though the available range of dates is very narrow. Source

It would be wonderful to expand that date range to 2020 and see if the rise remains constant, or (as one might suspect), rose more sharply 2024-26 – if only for the unchaperoned actions of agentic AI systems.

Sadly, official PyPi sources state that download statistics for PyPi are not made available, for various reasons, including CDN caching overheads, inaccurate download counts caused by caches, mirrors, unofficial download inflation and historical data quality issues – and, according to the Python site, their limited usefulness as a measure of a package's quality.

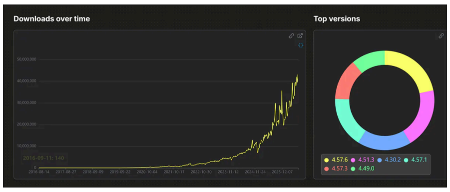
Not So Fast

However, PyPi gives out the components needed for *others* to analyze download trends; there are therefore diverse websites that allow more than a month's range, such as PepySite – but most are subscription-based, and you must pay to see the longer-range trends and stats (up to $490 USD p/m in one case, for the 'VC' package).

Is there anything that we can glean, then, about Python as an index of AI acceleration, away from the costly APIs that lay out these trends for the well-heeled?

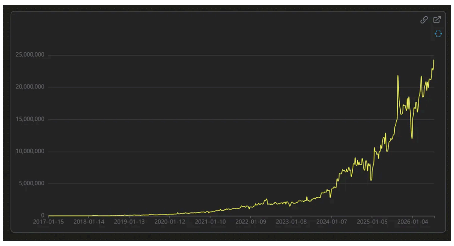
Though PyPiStats is a FOSS resource, it for some reason offers only the most recent month's stats, and this range cannot be extended even for payment.

Fortunately, one site is more generous – at ClickPy, one can at least search for the adoption history of individual packages, even if the sum of these is not orchestrated into an overview that easily yields broader, coordinated trends. Here, for instance, is the passage of the Transformers library since 2016 (though only valid since its launch in 2019):

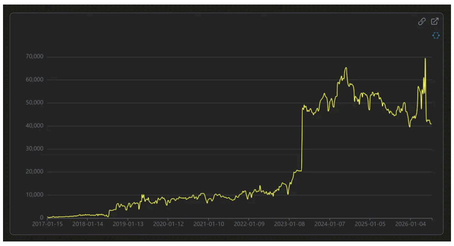


The meteoric rise of transformers. [Source]

What about the venerable Torch, [first released in 2002], but not destined to [torment] home-brew AI enthusiasts until its development into PyTorch, and the rise of Transformer-based VLMs and generative AI related to images and videos..?

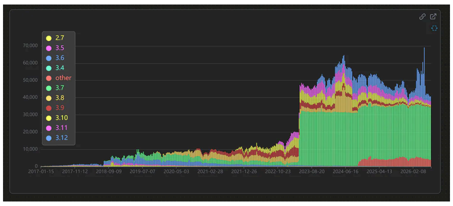


Torch downloads since 2017, with a steep incline upwards over the last 2-3 years.



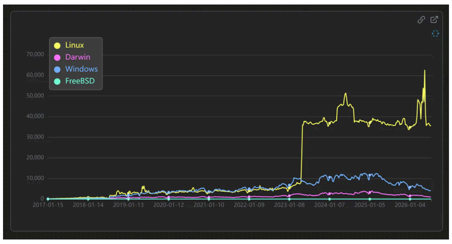
PyTorch over the same period, plateauing and then spiking sometimes over a stable base.

More interesting, for instance in the case of PyTorch (see image directly above), is the overview of adoption *across diverse Python versions* since 2017:



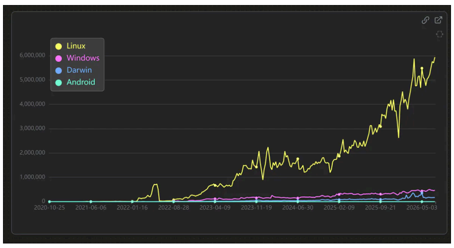
Downloads of PyTorch over time since 2017, by Python version, supplying an extra dimension of insight into uptake since the 'hot' era of this third AI revolution.

If the stats above illuminate the surge in take-up since the advent of truly capable LLM and VLM systems, this becomes even more apparent when viewed by system:



Adoption of PyTorch since 2017, interpreted by operating system.

It's a similar story for the [accelerate] package – one of the most required libraries for VRAM-starved AI users looking to squeeze as much as possible out of the limitations of their systems:



The ascent of 'accelerate', by OS.

Thus it goes for any number of packages or libraries one researches at ClickPy, and it certainly would be interesting to cook up an overview instead of looking up the pieces one by one.

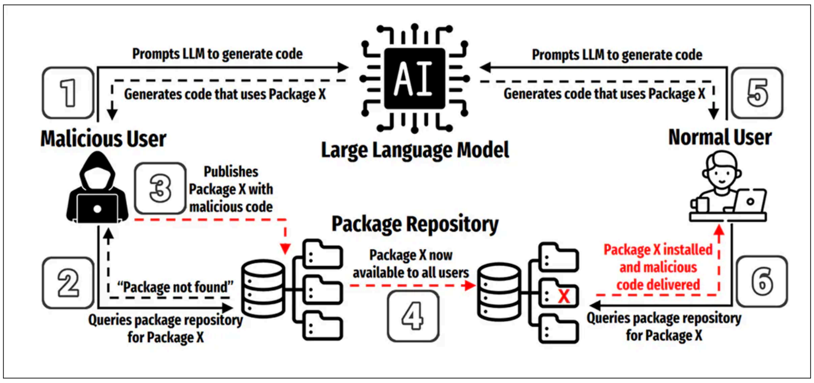
One thing that becomes evident from the 'by OS' graphs is the increase of Linux adoption. While Linux desktop adoption is making a steady but plodding rise in the face of unpopular Windows strategies, the use of Linux *environments* is clearly in a significant ascent.

For instance, I run Ubuntu virtualized on WSL2 in Windows for diverse Docker containers, wherein Docker can leverage a genuine Linux system on the same terms as the (presumably Linux-native) developers envisioned. That's two active Linux systems in an entirely Windows LAN (from the point of view of bare metal), with a native Linux laptop waiting unplugged in the wings for more languorous times.

Conclusion

In terms of CLI-based package downloads, it's interesting to note how avid and pursued this very geeky space has become in the wake of AI.

The increase in package consumption is beginning to become a security hazard downstream, with the advent of slopsquatting. Since LLMs are prone to hallucinate packages, they are likely to actually send requests for invented packages. Due to similarity of convergence, the fake packages are likely to reoccur in other library calls, to the point where it is worthwhile for bad actors to actually make the fake packages available for future pulls/calls – of course, with a malicious payload:



From the paper 'We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs' – AI-driven code generation expar surface. LLMs frequently hallucinate nonexistent package names, creating 'slopsquatting' vulnerabilities that bad actors exploit when automated pipelines blindly p
Source

First published Friday, July 24, 2026

Writer on machine learning, domain specialist in human image synthesis. Former head of research content at Metaphysic.ai, until its dissolution into DNEG's Brahma.ai.
Portfolio site: martinanderson.ai
Contact: martin@martinanderson.ai

Discover More