

Data Deduplication Done The Right Way



How To Fine-Tune LLMs For Domain-Specific Adaptation

₹120
ISSN-2456-4885

An EFY GROUP Publication

Volume: 14 | Issue: 05 | Pages: 100 | March 2026

OpenSource ForYou

THE COMPLETE MAGAZINE ON OPEN SOURCE

AI: Powered By Open Source?

The role open source is playing in accelerating the development of AI



Why Open Source Large Language Models Are Popular

Popular FOSS Tools For LLM Observability, Monitoring And Evaluation

The Transformation Of Prompts Into Source Code

The Relevance Of Rubber Duck Debugging In The Age Of AI



The Path To Cybersecurity In The Quantum Era

Optimising RAM Usage With Python



www.opensourceforu.com

Wanna Be Your Own Boss?

DO OPEN SOURCE.



Demand for Open Source is sky rocketing. Be it for managing IT infrastructure or development of software--Open Source solutions are what customers are seeking.

All you need to do is develop expertise in an Open Source stack, and then build a team around it!

And, Open Source For You can be your friend and a guide through this journey.

TO READ OUR PRINT EDITION **Visit:** https://subscribe.efyindia.com

TO READ OUR EZINE EDITION **Visit:** https://ezine.lfymag.com



WORLD'S LEADING PUBLICATION ON OPEN SOURCE

Looking for marketing solutions to engage with cutting edge techies?
Contact us at growmybiz@efy.in OR call us at +91-9811155335.

The Simple Cloud: user-friendly & fast!

Getting started made easy



HETZNER

Get started now with the CPX12 and benefit from low latency at our Singapore location.

cloud.hetzner.com

HetznerXOSFY26

Use the code and test our **Cloud for over a month for free.**

Promo code worth US$ 10 - valid for all Hetzner products - only for new customers - activate before 31 January 2027 - valid for 3 months after activation. All prices exclude VAT, setup fees and are subject to the terms and conditions of Hetzner Online GmbH.
Prices are subject to change.

CONTENTS

MARCH 2026 | ISSN-2456-4885

FOR U & ME

20 Revisiting Popular Programming Languages: The 2026 Edition

FOCUS

38 Building Production-Ready Retrieval Systems

43 Understanding Underfitting and Overfitting in Machine Learning

50 The Quiet Revolution: How Small Language Models are Redefining Enterprise AI Strategy

56 Popular FOSS Tools for LLM Observability, Monitoring and Evaluation

63 How to Fine-Tune LLMs for Domain-Specific Adaptation

68 Why Open Source Large Language Models are Popular

DEVELOPERS

72 How a Job Portal Benefited from Microservices Architecture

86 Modelling the IPL as a Graph System

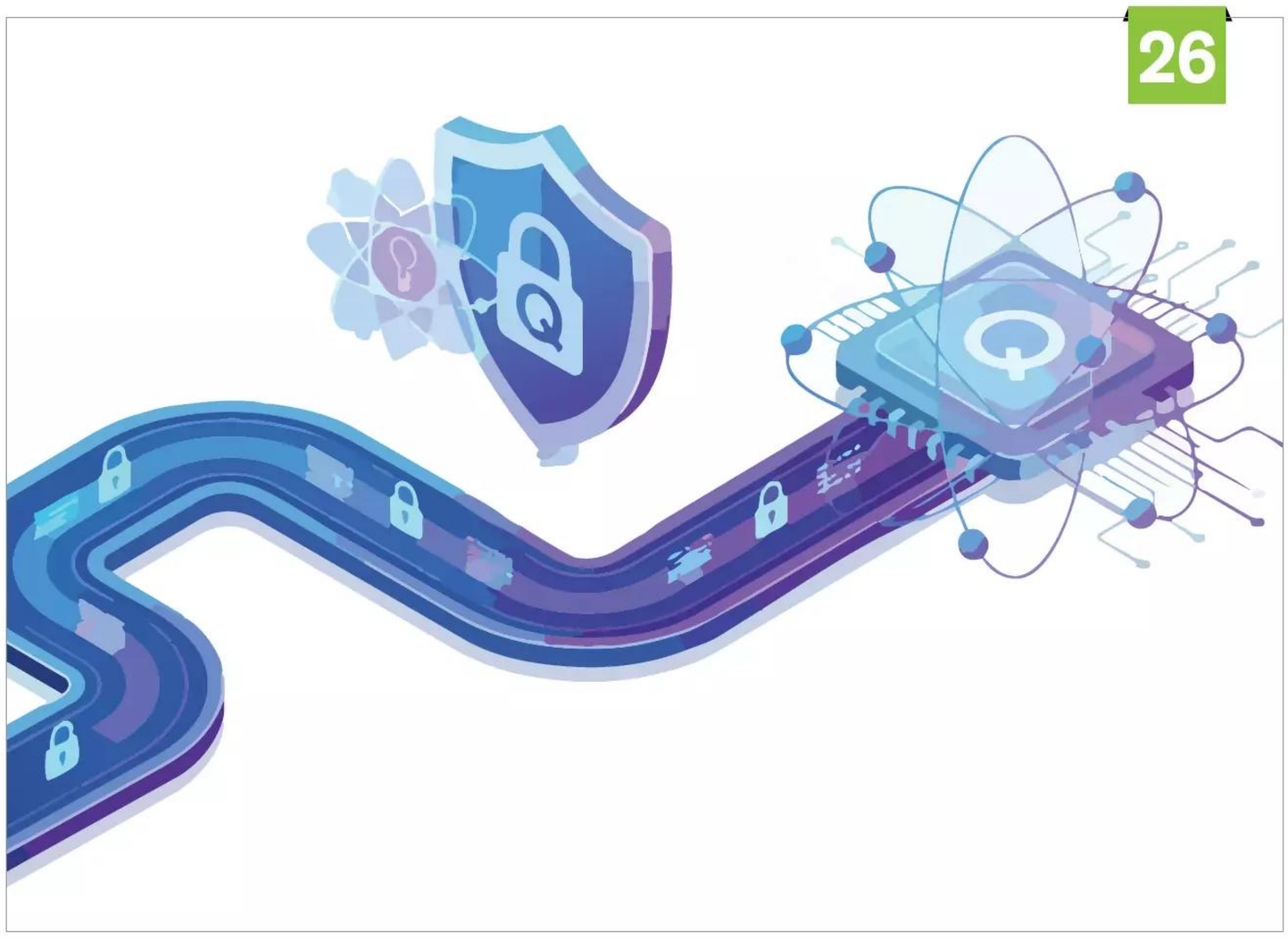
ADMIN

89 Data Deduplication Done the Right Way

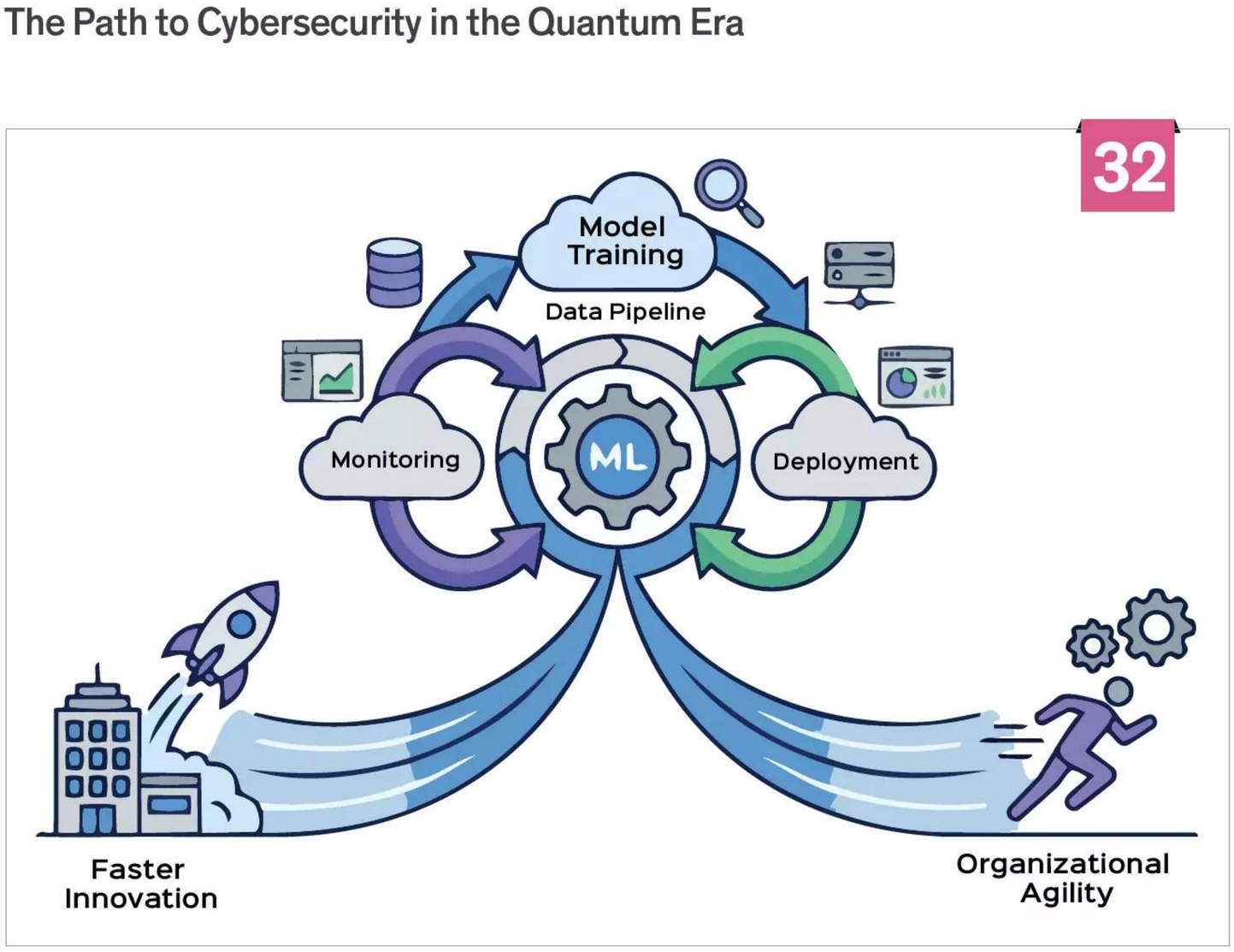
OPEN GURUS

92 Sending IoT Sensor Data to Public or Private Servers

94 I2C and I3C: How Modern Devices Communicate



The Path to Cybersecurity in the Quantum Era



MLOps for Enterprise Agility: An Overview

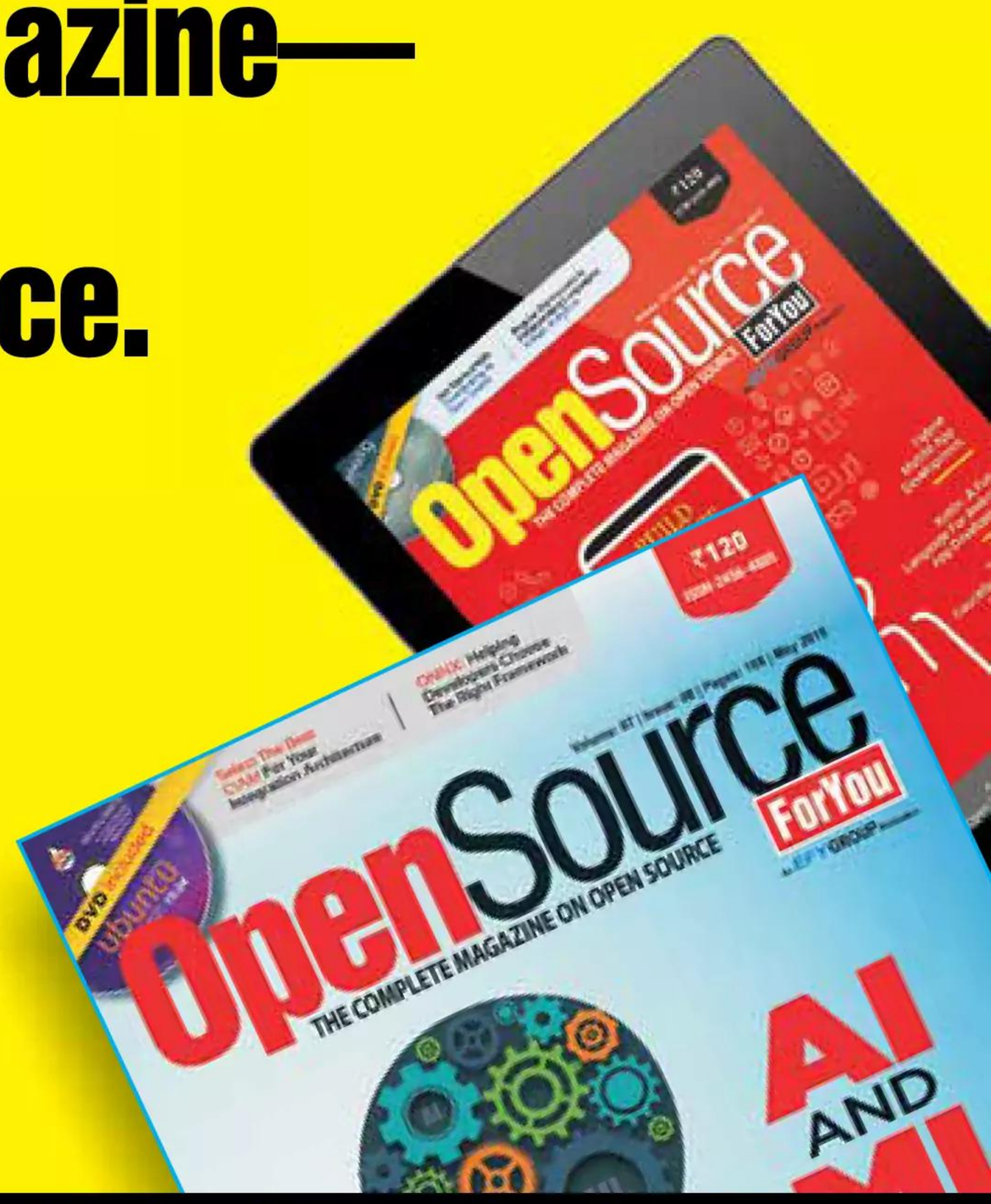
REGULAR FEATURES

07 FossBytes

EFYGROUP
YOURS SINCE 1969

Wanna Support Open Source For You?

Subscribe to the Magazine— so that we can keep promoting Open Source.



AMAZING OFFER

BUY 1, GET 1 FREE

WORLD'S LEADING PUBLICATION ON OPEN SOURCE

TEAR OUT — **ORDER FORM** — TEAR OUT

Please Tick (✓) Your Choice

☐ **Pay for 12 Issues Get 24 Issues**
2 Years (24 copies) + 24 Online Tracked Deliveries worth ₹480
₹1920

☐ **Pay for 24 Issues Get 48 Issues**
4 Years (48 copies) + 48 Online Tracked Deliveries worth ₹960
₹3840

☐ **Pay for 36 Issues Get 72 Issues**
6 Years (72 copies) + 72 Online Tracked Deliveries worth ₹1440
₹5760

To subscribe online, visit https://bit.ly/osfybogo OR **SCAN THIS CODE**

Name_____ Organisation_____ Mailing Address_____

_____ City_____

Pin Code_____ State_____ Phone No._____ Email_____

Subscription No. (for existing subscribers only)_____. **I would like to subscribe to the above (✓)marked Open Source For You magazine starting with the next issue.** Please find enclosed a sum of Rs_____ by DD/MO/crossed cheque bearing the No._____ dt._____ in favour of **EFY Enterprises Pvt Ltd,** payable at Delhi. *(Please add Rs 50 on non-metro cheque)*

Please mark one (nearest) relating to your subscription: ☐ Indian Company ☐ MNC ☐ R&D organisation ☐ Engineering institute ☐ College/School ☐ Any other (specify):_____

Send this filled-in form or its photocopy to : **EFY Enterprises Pvt Ltd,** D-87/1 Okhla Industrial Area, Phase 1, New Delhi 110 020 | Ph: **011-40596600** | e-mail: **support@efy.in**

Terms:- # These rates are applicable for new subscribers as well as renewal by existing subscribers. # The rates are valid for subscribers within India only. # Please allow 4-6 weeks for processing of your subscription. # The subscription copies will be dispatched through ordinary post only # Subscription Agents will not get agency commission against this scheme # Disputes, if any, are subject to exclusive jurisdiction of competent courts and forums in Delhi/New Delhi only. * Replacement will be made if intimation of damaged / non-receipt of copies is received within 30 days of its publication ** After three months, if you are not satisfied with the magazine, your balance amount will be returned (Not applicable for gift offer)

EDITOR
RAHUL CHOPRA

EDITORIAL, SUBSCRIPTIONS & ADVERTISING
Delhi (HQ)
D-87/1, Okhla Industrial Area, Phase I, New Delhi 110020
Phone: +91-9811155335
E-mail: info@efy.in

MISSING ISSUES
Phone: +91-9811155335
E-mail: support@efy.in

BACK ISSUES
Phone: +91-9811155335
E-mail: support@efy.in

NEWSSTAND DISTRIBUTION
Phone: +91-9811155335
E-mail: efycirc@efy.in

ADVERTISEMENTS
NEW DELHI (HEAD OFFICE)
Phone: +91-9811155335
E-mail: efyenq@efy.in

MUMBAI
E-mail: rmowest@efy.in

BENGALURU
E-mail: rmosouth@efy.in

CHINA
Worldwide Focus Media
E-mail: china@efy.in

GERMANY
pms Plantenberg Media Service GmbH
E-mail: germany@efy.in

JAPAN
Tandem Inc.
E-mail: japan@efy.in

TAIWAN
J.K. Media
E-mail: taiwan@efy.in

UNITED KINGDOM
ASA Media
E-mail: uk@efy.in

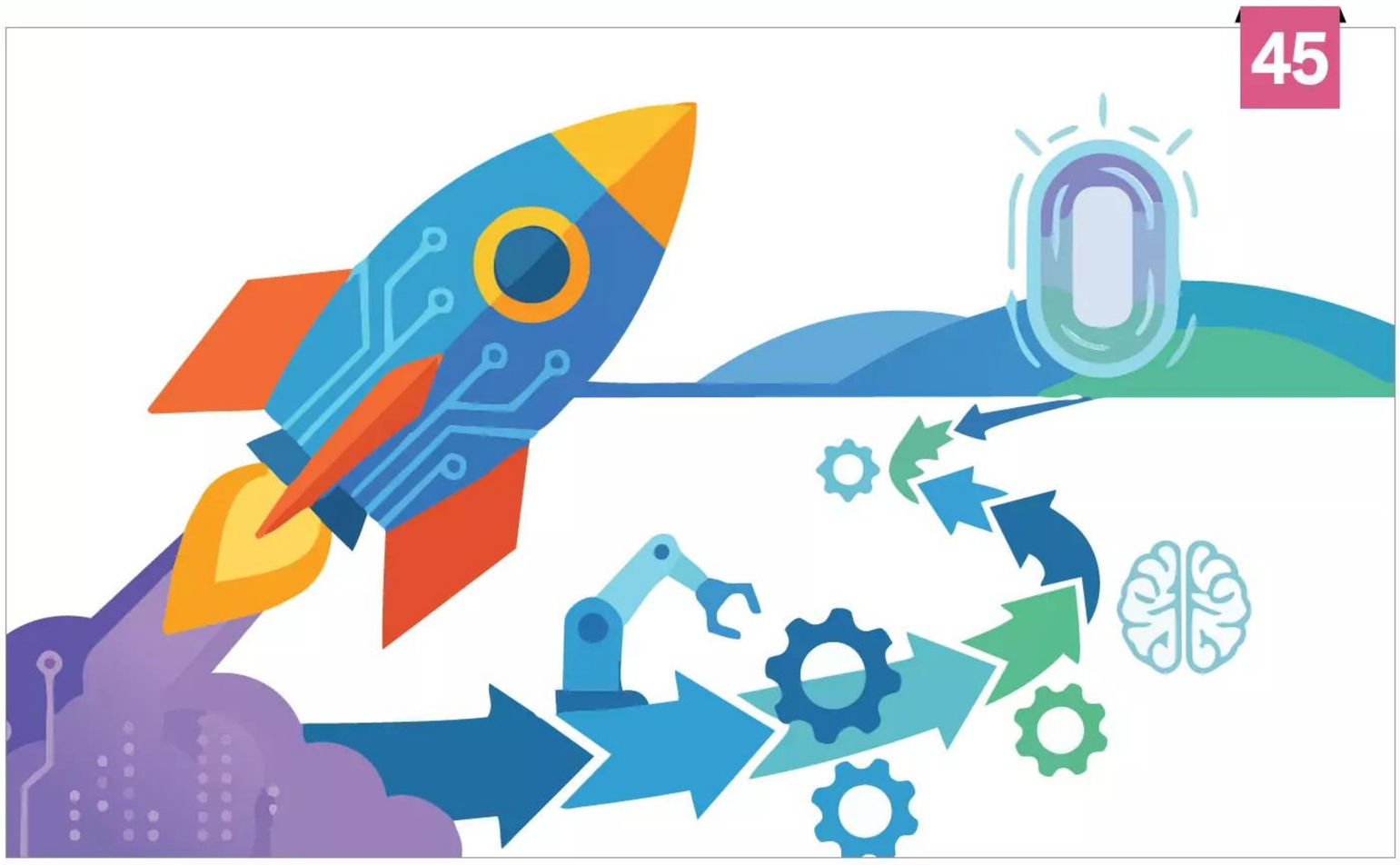
UNITED STATES
E & Tech Media
E-mail: usa@efy.in

Printed, published and owned by Ramesh Chopra. Printed at Tara Art Printers Pvt Ltd, A-46,47, Sec-5, Noida, on 28th of the previous month, and published from D-87/1, Okhla Industrial Area, Phase I, New Delhi 110020. Copyright © 2026. All articles in this issue, except for interviews, verbatim quotes, or unless otherwise explicitly mentioned, will be released under Creative Commons Attribution-NonCommercial 3.0 Unported License a month after the date of publication. Refer to http://creativecommons.org/licenses/by-nc/3.0/ for a copy of the licence. Although every effort is made to ensure accuracy, no responsibility whatsoever is taken for any loss due to publishing errors. Articles that cannot be used are returned to the authors if accompanied by a self-addressed and sufficiently stamped envelope. But no responsibility is taken for any loss or delay in returning the material. Disputes, if any, will be settled in a New Delhi court only.

SUBSCRIPTION RATES			
Year	**Newstand Price (₹)**	**You Pay (₹)**	**Overseas**
Five	7200	4320	—
Three	4320	3030	—
One	1440	1150	US$ 120

Kindly add ₹ 50/- for outside Delhi cheques.
Please send payments only in favour of EFY Enterprises Pvt Ltd.
Non-receipt of copies may be reported to **support@efy.in**—do mention your subscription number.

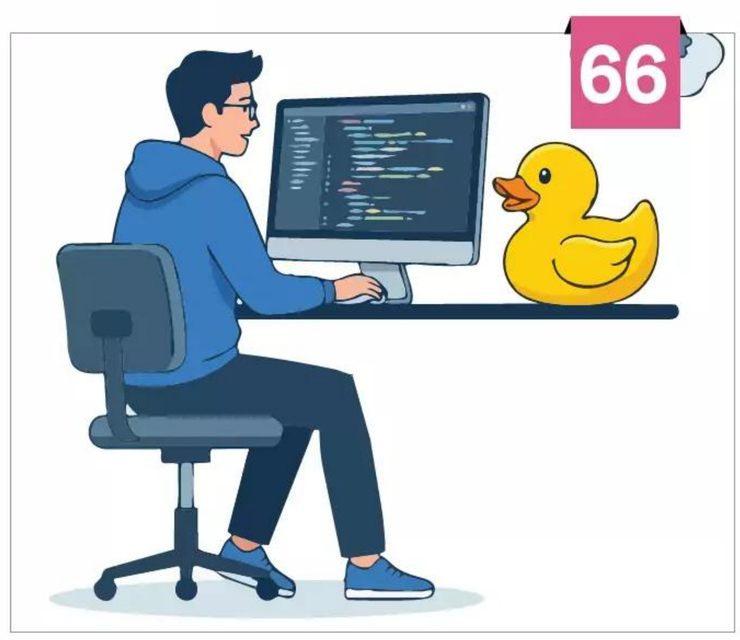
CONTENTS



Moving Towards New Frontiers with Generative AI



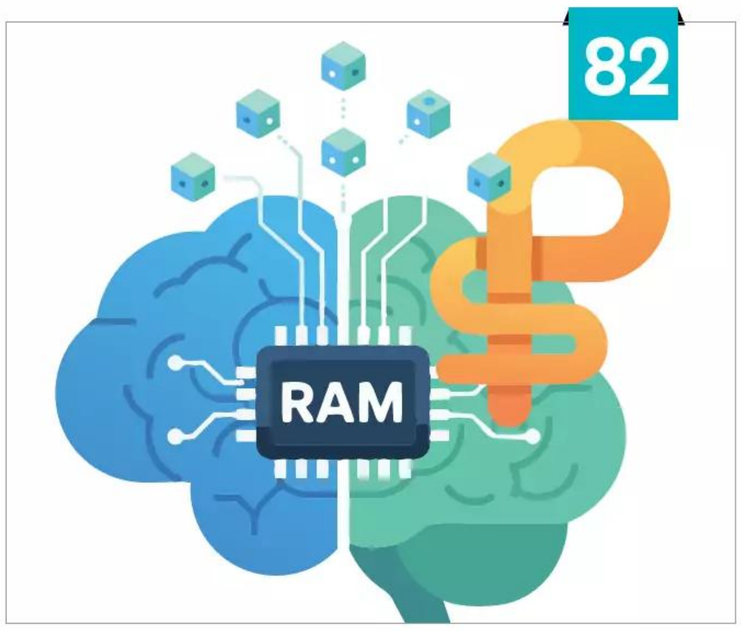
The Transformation of Prompts into Source Code



The Relevance of Rubber Duck Debugging in the Age of AI



The Role that Software Architects Play



Optimising RAM Usage with Python

A14Bharat pushes for open AI at India's AI Impact Summit 2026



India hosted the AI Impact Summit 2026 in New Delhi from February 16–20, convening global technology leaders, researchers, policymakers and investors around a clear priority: building inclusive, socially responsible and widely accessible AI. The event's guiding sutras — People, Planet, Progress — place open ecosystems and public digital infrastructure at the centre of deployment.

The strongest open signal came from Nandan Nilekani, co-founder and chairman, Infosys, whose AI4Bharat open AI initiative has received ₹ 700 million in funding to build models for India's 22 official languages. Nilekani describes India as the "use-case capital of AI" and argues AI must be "inclusive, not extractive."

Infrastructure scale follows. Sundar Pichai, CEO, Google & Alphabet, announced a $15 billion AI hub and data centre in Visakhapatnam to provide gigawatt-scale compute, saying it will "bring our industry-leading technology to enterprises and users in India, accelerating AI innovation." Partner Sunil Bharti Mittal, founder and chairman, Bharti Airtel, is extending AI-led telecom networks to broaden access.

At the hardware core, Jensen Huang, founder and CEO, NVIDIA, characterised modern facilities as "AI factories" producing valuable "tokens," powered by new Grace Blackwell servers and DGX systems.

Industry adoption is expanding through Mukesh Ambani's vision of 'AI Everywhere for Everyone', while Cristiano Amon, CEO, Qualcomm, pushes edge AI and has acquired Arduino to open source its AI computing platform.

From Bill Gates' focus on AI that should "save and improve millions of lives" to OpenAI, Anthropic and Tata's enterprise AI drives, the summit frames AI not as closed technology, but as shared national infrastructure.

Yotta migrates BHASHINI from foreign cloud providers to sovereign infrastructure

Mumbai-based Yotta Data Services has completed the deployment of BHASHINI's full sovereign AI cloud transformation on its Government Community Cloud (GCC) and Shakti Cloud, moving the national language platform entirely onto

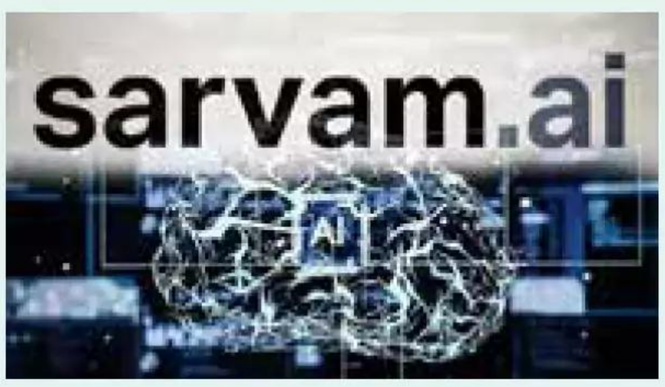
India's Sarvam AI challenges big tech with 84.3% OCR accuracy

Sarvam Vision recorded 84.3% accuracy on the olmOCR-Bench, scoring higher than Gemini 3 Pro, DeepSeek OCR v2, and significantly above ChatGPT. The result places India among a small group of countries, alongside the US and China, with domestically developed foundation or LLM-class AI systems.

Positioned as a sovereign AI platform, Sarvam Vision has been built in Bengaluru using local talent and tailored for India-specific use cases, reducing dependence on foreign AI infrastructure. Its core strengths include Indic language AI, OCR, text-to-speech, and speech-to-text—areas where focused optimisation appears to outperform larger, general-purpose global models.

Pratyush Kumar, co-founder, Sarvam Vision AI, shared the benchmark outcomes in a series of posts on 'X', highlighting the model's comparative performance against ChatGPT and Gemini.

Industry observers have taken note. Deedy Das, tech commentator, said: "I was wrong about Sarvam. When I wrote about them a year ago, I felt like the direction to train small Indic language models was wrong. But boy, have they turned it around. They have the best text-to-speech, speech-to text, and OCR models for Indic languages, and that's actually really valuable. The pricing is very reasonable."



IOS-MCN releases open source Agartala 0.4.0 to enable private 5G

The Indian Open Source for Mobile Communication Networks (IOS-MCN) Consortium has launched Agartala 0.4.0, an open source software platform that enables enterprises and institutions to build and operate private 5G networks, marking a key step towards deployable, industry-grade 5G infrastructure developed in India.

Funded by the Ministry of Electronics and Information Technology (MeitY), Government of India, the platform is designed to support pilot deployments that can pave the way for large-scale adoption of private 5G networks across sectors.

The IOS-MCN Consortium is led by the Indian Institute of Science (IISc), Bengaluru, in collaboration with the Indian Institute of Technology (IIT), Delhi, and the Centre for Development of Advanced Computing (C-DAC). The initiative focuses on translating research-led innovation into operational national infrastructure through publicly funded open source development.

Agartala 0.4.0 is designed for factories, campuses, research institutions and startups, offering faster, more reliable and secure connectivity compared to Wi-Fi or public mobile networks, while remaining more cost-effective. The release positions open source as a strategic foundation for India's private 5G ecosystem, reducing dependence on proprietary telecom platforms.



indigenous infrastructure. The system now runs exclusively on Indian cloud and GPU resources, with all language datasets, models and citizen interactions confined within Indian jurisdiction, ensuring complete data sovereignty.

The milestone aligns directly with IndiaAI Mission objectives and signals a decisive shift towards self-reliant, scalable sovereign AI systems capable of operating without foreign hyperscalers.

At the core of the migration is an open source, cloud-agnostic and interoperable architecture designed to ensure vendor neutrality and long-term strategic autonomy. The framework is modular and reusable, enabling ministries, public sector units and national programmes to replicate hyperscaler-to-Indian-cloud transitions.

Operational metrics underline the shift's viability. The deployment recorded up to 40 percent performance gains, 20–30 percent cost savings, 99.99 percent uptime and zero data loss while migrating more than 200 TiB of data and over 3.5 billion files. Executed within two to three months, the transition covered BHASHINI's complete AI stack, including datasets, models, APIs, containerised services, orchestration pipelines, databases and storage.

Sunil Gupta, co-founder, managing director and CEO, Yotta Data Services, said, "Yotta's successful deployment of BHASHINI on Shakti cloud marks a defining moment for India's data sovereignty journey. This transition highlights that hyperscale, mission-critical AI platforms can be built and operated entirely on sovereign infrastructure, without compromise."

NVIDIA open sources its Alpamayo AI model to help cars think in chaotic traffic

NVIDIA has open sourced its Alpamayo family of reasoning-led AI models, along with simulation tools and datasets, aiming to accelerate autonomous vehicle development and push self-driving beyond controlled test environments into chaotic real-world roads.



Jensen Huang, CEO, NVIDIA

The new stack introduces what the company calls a "thinking" approach, enabling vehicles to reason through unfamiliar situations rather than rely mainly on rules or large-scale pattern matching. NVIDIA says this step-by-step, human-like reasoning is critical for markets such as India, where heterogeneous traffic, weak lane discipline, unpredictable behaviour and complex road conditions have long challenged autonomy.

"The Alpamayo technology is based on a revolutionary new approach called 'thinking'. For the very first time, we have added to the car the ability to reason… the way we humans do," said Jensen Huang, CEO, NVIDIA.

Positioned as the world's first reasoning-based autonomous driving AI stack, Alpamayo shifts development away from closed, proprietary systems towards broader developer participation, faster experimentation and quicker localisation.

Mercedes-Benz has already integrated NVIDIA's platform into production vehicles. Its latest-generation S-Class runs the NVIDIA DRIVE AV Level-4 full-stack system, marking a five-year collaboration between the two companies.

"With our Alpamayo technology, Mercedes is among the safest autonomous platforms in the world. The next one is Tesla, and almost everybody else is quite far behind," Huang said.

EFY GROUP
YOURS SINCE 1969

Stay **Connected.**
Stay **Informed.**
Stay **Ahead.**



SUBSCRIBE AND **SAVE**

TEAR OUT — **ORDER FORM** — TEAR OUT

PRINT MAGAZINE	1 YEAR (12 copies each)	3 YEARS (36 copies each)	5 YEARS (60 copies each)
Electronics For You (Rs 100/copy)		**WITHIN INDIA (IN RUPEES)**	
	960 ☐	2700 ☐	4200 ☐
		SAARC COUNTRIES (IN US$ BY AIR)	
	57 ☐	155 ☐	NA
		OTHER COUNTRIES (IN US$ BY AIR)	
	114 ☐	310 ☐	NA

e-magazine subscriptions within India are available at half the rates mentioned here.

Overseas rates for each e-magazine in US$: 1 year: $12; 3 years: $33; 5 years: $50 only

PRINT MAGAZINE SUBSCRIBERS GET:

- Free delivery of print magazine by post
- And much more (check: subscribe@efy.in)
- For delivery by courier, please add Rs 50 for each copy

To subscribe online, visit https://payment.efyindia.com | **OR** | **SCAN THIS CODE**

Name_____Organisation_____Mailing Address_____

_____City_____

Pin Code_____State_____Phone No._____ Email_____

Subscription No. (for existing subscribers only)_____. I **would like to subscribe to the above (✓)marked magazine(s) starting with the next issue.** Please find enclosed a sum of

Rs_____by DD/MO/crossed cheque bearing the No._____ dt._____in favour of EFY Enterprises Pvt Ltd, payable at Delhi.

Please mark one (nearest) relating to your subscription: ☐ **Indian Company** ☐ **MNC** ☐ **R&D organisation** ☐ **Engineering institute** ☐ **College/School** ☐ **Any other (specify):**_____

Send this filled-in form or copy to : EFY Enterprises Pvt Ltd, D-87/1 Okhla Industrial Area, Phase 1, New Delhi 110 020 | Ph: 011-40596600 | e-mail: support@efy.in

Terms:- # These rates are applicable for new subscribers as well as renewal by existing subscribers # Please allow 4-6 weeks for processing of your subscription.
Please include your pincode for prompt delivery of your copy.

Hyderabad AI meet calls for national open datasets to power India AI

India's AI challenge is not compute scarcity but the absence of truly open, high-quality training datasets, experts said at a discussion organised by the Software Freedom Law Centre at IIIT Hyderabad, calling on the India AI Mission to prioritise public data collection over GPUs.

"My primary focus for the IndiaAI Mission would be to put all the money, first of all, into collecting data," one speaker said.

The Mission has allocated ₹ 103.72 billion over five years, with this year's outlay cut to ₹ 10 billion from ₹ 20 billion, and only ₹ 8 billion utilised last year. Speakers argued the structure is ineffective. "It's not even a drop in the ocean, compared to megacorps in the US and China," one said, adding that long procurement cycles leave hardware outdated.

Instead, India faces a severe shortage of language data. "Everybody is starved of data. And the way out they're taking is using synthetic data," a speaker noted, warning this leads to "bad data" and "a bad model." Even Hindi lacks adequate, dialect-sensitive coverage.

Experts proposed a government-led effort to crowdsource datasets through teachers and public institutions. "Eight thousand samples of questions, answers, and reasoning—can't we get 8000 samples… in Telugu or Hindi? Is that so hard? It's not."

They also criticised "openwashing". "Nobody provides you the training data," one said, stressing that true open source must include data, documentation, and workflows. Frameworks such as the Model Openness Framework were recommended, while Stable Diffusion was cited as an example of fully open AI.

"We have the first thinking car, which reasons about why it is doing something." Huang added that the technology can be deployed rapidly worldwide.

Open source software helps build detailed 3D map of Chandrayaan-3 landing site

An independent Indian researcher has demonstrated that mission-grade lunar mapping no longer needs to remain confined to space agencies. Using publicly available Chandrayaan-2 imagery and open source tools, Germany-based Chandra Tungathurthi has built a high-resolution 3D terrain map of Shiv Shakti Point, the Chandrayaan-3 Vikram lander's touchdown site near the Moon's south pole.



The work resulted in a Digital Elevation Model (DEM) with roughly 30cm per pixel resolution, detailed enough to capture terrain shape, slopes, depressions, craters, and boulders, and to support rover path planning and safe landing analysis. The model was generated from Orbiter High Resolution Camera stereo image pairs using photogrammetry techniques. Tungathurthi spent months building an open processing pipeline, while the final computation alone ran for over 24 hours on a high-performance computer.

Explaining the value of elevation data, Tungathurthi said, "An OHRC image tells you what the surface looks like. A DEM tells you what the surface actually is," adding, "Over the past several months, I spent considerable time understanding how Chandrayaan-2 data is organised and released. During this process, I realised that recent advances in open source tools have made it possible to independently generate high-resolution elevation maps of the lunar surface."

India talks open source but runs on closed AI APIs, says Activate survey

India's AI startup ecosystem may speak the language of open source, but production reality tells a different story.

A survey by venture firm Activate covering 244 CTOs, developers, and founders finds 74% of deployments rely on proprietary Western models accessed through APIs, while only 13% customise or train their own models and just 13% use open source models. As the report states, "The 'open source India' narrative doesn't match what's actually running in production."



Infrastructure use reflects this dependence. 65% of GPU capacity goes to inference, compared with 21% for training and 14% for fine-tuning. "The majority of this ecosystem is calling APIs, not training models," the report notes.

Cost pressures reinforce the trend. 60% spend under $2400 per month, making local training uneconomical. "Investing in training infrastructure makes less sense every month when you can ride the capability curve by calling an API."

India's #1 Event For R&D and New Product Development Of Electronics

11TH EDITION INDIA ELECTRONICS WEEK 2026

13-15 MAY 2026

KTPO Expo Center | Whitefield | Bengaluru | India

For more information on sponsoring, exhibiting or attending please call +91-9811155335 or growmybiz@efy.in



Structured leadership succession plan developed for Linux

A formal, open source succession and continuity plan has been adopted to safeguard the future of the world's most critical open source infrastructure based on Linux, ensuring development continues smoothly if Linus Torvalds steps aside.

The move addresses a long-standing governance risk. Torvalds has served as the ultimate judge and executor of all commits to the Canonical Linux repository for more than 30 years, retaining final authority over the codebase that underpins countless hardware and software products worldwide.

Drafted by Dan Williams and co-signed by Torvalds, the document defines "a plan for navigating events that affect the forward progress of the Canonical Linux repository, torvalds/linux.git." If Torvalds becomes unwilling or unable to continue, successors must step in "without delay."

The process is structured and time-bound. The last Maintainer Summit Organizer initiates discussions, with the Linux Foundation Technical Advisory Board acting as backup. A successor discussion must begin within 72 hours, followed by a meeting of summit invitees, TAB members and additional maintainers. The chosen canonical maintainer is announced within two weeks.

Although Linux has more than 100 maintainers, decision-making has remained centralised with Torvalds. The new plan distributes responsibility and formalises leadership transfer.



Linus Torvalds

Open source preference remains weak, with only 4% choosing open weights and 2% selecting Meta's Llama.

Meanwhile, 62% plan to adopt AI-agentic workflows, despite warnings that "Agentic workflows are also the most expensive and failure-prone pattern," with cost already the top barrier.

Structural gaps persist. 16% cite lack of sovereign datasets, while experts argue India needs multilingual local data more than GPUs. Talent shortages are operational rather than technical: "The talent bottleneck isn't in ML engineering. It's in AI ops."

Western platforms dominate across the stack, with OpenAI and Anthropic leading text and code tools, underscoring a clear reality: India is consuming AI infrastructure, not building it.

Claude goes open source and Indian IT stocks slide

Anthropic has released 11 open source plugins for its Claude Cowork platform, making enterprise AI automation freely accessible to developers and businesses and allowing companies to integrate, customise, and deploy workflows without proprietary lock-in.



The move immediately rattled markets in India, where IT services depend heavily on large teams and billable hours. Indian IT export shares slumped about 6.3 per cent overall, with the NSE/Nifty IT index heading for its worst session since March 2020. Blue-chip stocks bore the brunt of the sell-off.

Investor concerns centre on the plug-ins' ability to automate tasks across legal, sales, marketing, analytics, coding, testing, and documentation — work traditionally handled by large offshore teams.

"As Indian enterprises integrate Claude for critical coding workflows, dependency on large vendor teams may decline, squeezing billable hours and margins. Anthropic's advanced AI systems also threaten entry-level talent pool at Indian IT firms by replacing routine development and testing tasks," said Ambrish Shah, analyst, Systematix Group.

Toyota introduces Fluorite, an open source game engine for vehicle dashboards

Toyota Connected North America has unveiled Fluorite, an open source game engine designed to deliver console-grade interactive experiences for in-vehicle digital cockpits and dashboards, signalling a clear shift towards open technologies for its next-generation vehicle user experience stack.

Developed by Toyota Connected North America in Plano, Texas, the engine was introduced at FOSDEM 2026 in Brussels, a community-driven conference dedicated to open source software. While initially targeted at automotive displays, the platform could eventually extend to consoles such as Xbox or PlayStation.

Fluorite is built entirely on open source foundations, combining Google's Flutter UI toolkit, the Dart programming language, and Google's Filament 3D rendering engine. It runs on an embedded stack already proven inside Toyota vehicles, including Flutter runtime, Yocto Linux, and Wayland, and supports deployment across Android, iOS/macOS, Windows, Linux, and WebGL.

Positioned as the first console-grade engine fully integrated with Flutter, Fluorite delivers hardware-accelerated graphics via Vulkan, alongside physically based rendering, accurate lighting, post-processing effects, custom shaders, and multiple simultaneous 3D scene views.

"Its reduced complexity [is achieved] by allowing [developers] to write your game code directly in Dart and using all of its great developer tools. By using a FluoriteView widget, you can add multiple simultaneous views of your 3D scene, as well as share state between game entities and UI widgets – all in the Flutter way," said the Fluorite development team.

TCS and AMD collaborate for Helios-based AI infrastructure in India

Tata Consultancy Services (TCS) and Advanced Micro Devices (AMD) are co-developing India's first open ROCm-powered AI infrastructure built on AMD's 'Helios' architecture, positioning open software at the core of the country's next wave of sovereign AI factories.

Delivered through the TCS subsidiary HyperVault AI Data Center Limited, the collaboration will introduce an AI-ready data centre blueprint supporting up to 200 MW of capacity. The platform targets hyperscalers, AI companies and enterprises seeking faster, localised AI build-outs while reducing reliance on closed hardware-software ecosystems.



Helios combines AMD Instinct MI455X GPUs, next-generation EPYC 'Venice' CPUs, Pensando Vulcano NICs and the open ROCm software ecosystem, creating a rack-scale, performance-optimised and energy-efficient stack designed for flexibility and long-term interoperability.

"This collaboration lays the foundation for AMD's first Helios-powered AI infrastructure in India. By combining our strengths in AI, connectivity, sustainable power, and advanced data centre engineering, we are poised to deliver state-of-the-art infrastructure solutions for AI companies and global enterprises," said K. Krithivasan, MD and CEO, TCS.

"AI adoption is accelerating from pilots to large-scale deployments, and that shift requires a new blueprint for compute infrastructure. With 'Helios', we are delivering an open, rack-scale AI platform designed for performance, efficiency, and long-term flexibility. Together with TCS, we are enabling enterprises across India to deploy AI at scale today while building the compute foundation of tomorrow," added Lisa Su, chair and CEO, AMD.

Indian government mandates labelling of AI-generated content and quicker deletion of illegal deepfakes

India has introduced sweeping AI content rules that immediately place pressure on social platforms and open source AI ecosystems to label, trace and rapidly remove synthetic media at scale.



Under amendments to the Information Technology Rules, platforms must clearly label AI-generated or edited content, attach permanent metadata or provenance markers, add audible disclosures to AI audio, and deploy "reasonable and appropriate technical measures" to prevent unlawful uploads. Illegal or harmful deepfakes must be removed within three hours of detection or reporting, sharply reduced from the earlier 36-hour window. These rules took effect on February 20, 2026.

Ethereum co-founder puts $45M into privacy-focused tech investment

Ethereum co-founder Vitalik Buterin has earmarked 16,384 Ether (ETH), worth about $45 million, from his personal holdings to fund privacy-focused open source technology development, marking one of the largest founder-led commitments to public digital infrastructure in the ecosystem.

The capital will be deployed gradually over several years rather than spent immediately, supporting privacy-preserving technologies, open hardware, and secure, verifiable software systems. Buterin described the objective as building an open source "full stack" capable of protecting both personal life and public environments.

He will personally take special initiatives that might otherwise sit with the Ethereum Foundation, positioning the work as complementary to Ethereum's decentralised base layer.

Priority areas include privacy tools, open infrastructure, self-sovereign and local-first systems, encrypted communications, open silicon, secure hardware, verifiable software stacks across finance, governance, operating systems, and blockchain infrastructure, using techniques such as zero-knowledge proofs and fully homomorphic encryption.

No detailed project allocations were disclosed, underscoring a broad, mission-led push to strengthen open, trustless public goods.



Vitalik Buterin





Asia's #1 Open Source Conference

Total attendees
7081

Speakers
85+

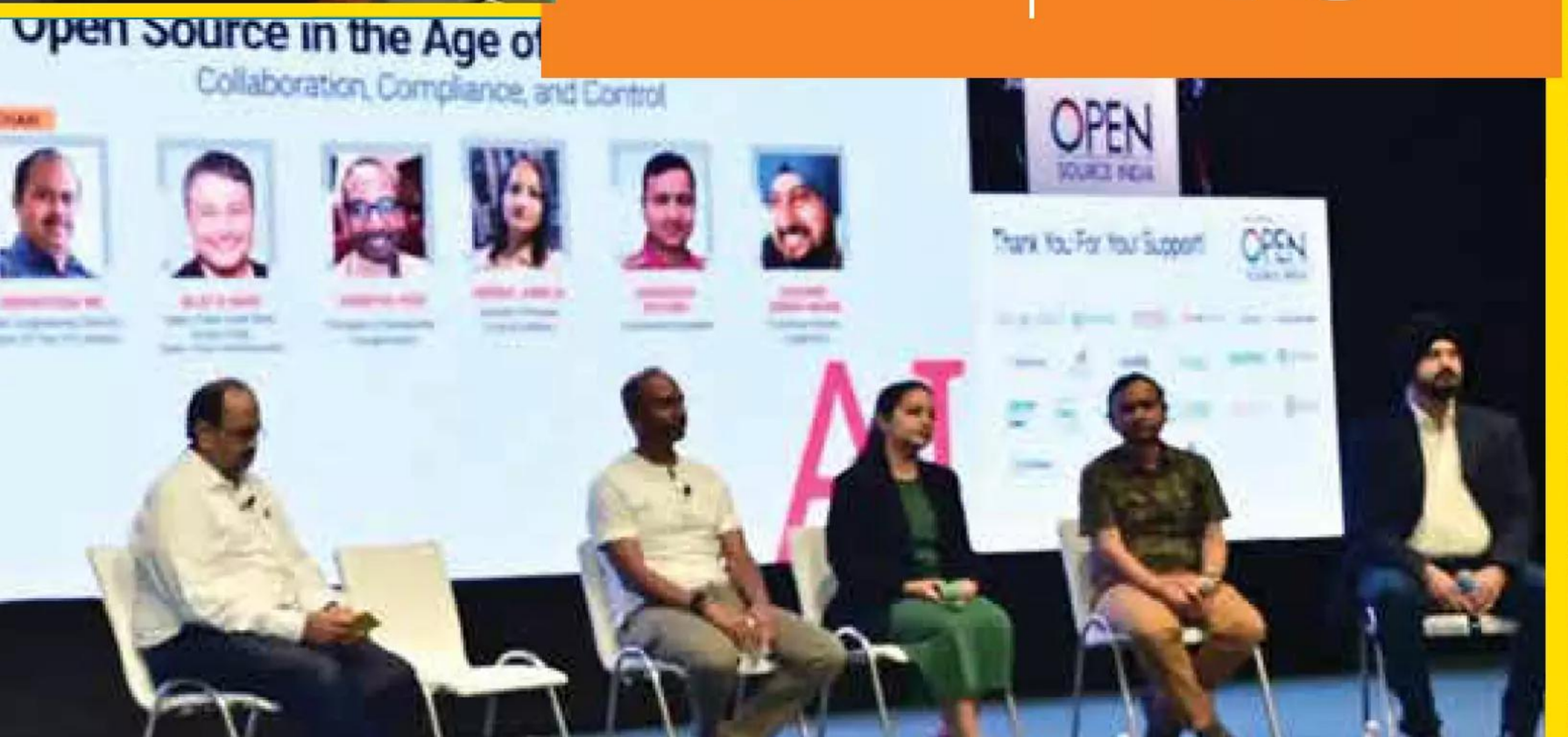
Sessions
45+

Workshops
10











www.OpenSourceIndia.in

An EFYGROUP EVENT

22nd Edition

OPEN SOURCE INDIA

Thank You

VISITORS, for making the event a huge success

SPEAKERS, for contributing your valuable thoughts

PARTNERS, for your support

Google Cloud · snowflake · NBBL NPCI Bharat BillPay Ltd. · PURESTORAGE · neo4j · AUTODESK · PERCONA · MariaDB Foundation · sysdig · Falco · SIEMENS · axonops · SAP · DELL · neon cloud · TuxCare We Secure Open Source · E-LAGORI Prototype Your Dreams · OpenSourceDB Your Trusted Data Partner · Mydbops Scaling Databases · Linux Professional Institute · vazu · mt-b

For more details, call on 98111 55335 or email us at info@opensourceindia.in

OpenClaw's creator joins OpenAI

In a move that reinforces its commitment to open development rather than acquisition, OpenAI has brought Peter Steinberger, founder of OpenClaw, into the company while placing the popular AI agent under a foundation structure to ensure it remains open source.

OpenAI will continue to support and fund the project instead of absorbing or closing it, creating a hybrid model that combines corporate backing with community governance.

CEO Sam Altman confirmed the decision, stating: "Peter Steinberger is joining OpenAI to drive the next generation of personal agents," and "OpenClaw will live in a foundation as an open source project that OpenAI will continue to support."

Previously known as Clawdbot or Moltbot, OpenClaw functions as a personal AI assistant capable of managing emails, dealing with insurers, checking in for flights, and handling multiple administrative tasks autonomously.

The project has seen rapid adoption, crossing 100,000 stars on GitHub and drawing two million visitors in a single week since its debut in November last year.

Steinberger reiterated the project's philosophy: "It's always been important to me that OpenClaw stays open source and given the freedom to flourish. Ultimately, I felt OpenAI was the best place to continue pushing on my vision and expand its reach."

The announcement was shared on X.



The mandate carries global weight. With nearly one billion internet users and more than 500 million social media users, India is a critical market for Google, Meta and X, and could set moderation benchmarks worldwide.

Current detection relies heavily on C2PA provenance metadata, yet gaps remain: metadata can be stripped, interoperability is weak, labels are subtle, and many AI tools — especially open source models — lack built-in provenance. This creates compliance risks for community projects, self-hosted systems and smaller platforms, pushing developers towards watermarking, signed provenance chains and open standards.

Civil liberties advocates, including Internet Freedom Foundation, warned the three-hour deadline may trigger 'rapid-fire' removals and excessive automation. Officials added provenance systems need only be implemented where "technically feasible."

NetActuate expands India footprint with open source-based ONE infrastructure

NetActuate has enhanced its Mumbai cloud platform using its Open Network Edge (ONE) Infrastructure-as-a-Service platform, built on open source software and open standards, positioning open technology as the core foundation of its latest regional expansion.



The upgrade expands services at the Mumbai data centre to include public cloud, private cloud, virtual private cloud (VPC) and hybrid cloud offerings. The objective is to meet rising enterprise demand for predictable performance, regulatory compliance, low latency and tighter control over data residency in India.

The company's existing presence in Chennai complements the Mumbai region, improving regional reach, route diversity, resilient deployments and international connectivity. Together, the two locations enable distributed and highly available architectures for business-critical workloads.

Delivered through the ONE platform, the infrastructure integrates pre-configured operating systems, monitoring and orchestration tools, and customer portal management to support faster provisioning and consistent multi-location deployments. The open source base is designed to enable standardisation, portability and cost control across hybrid environments.

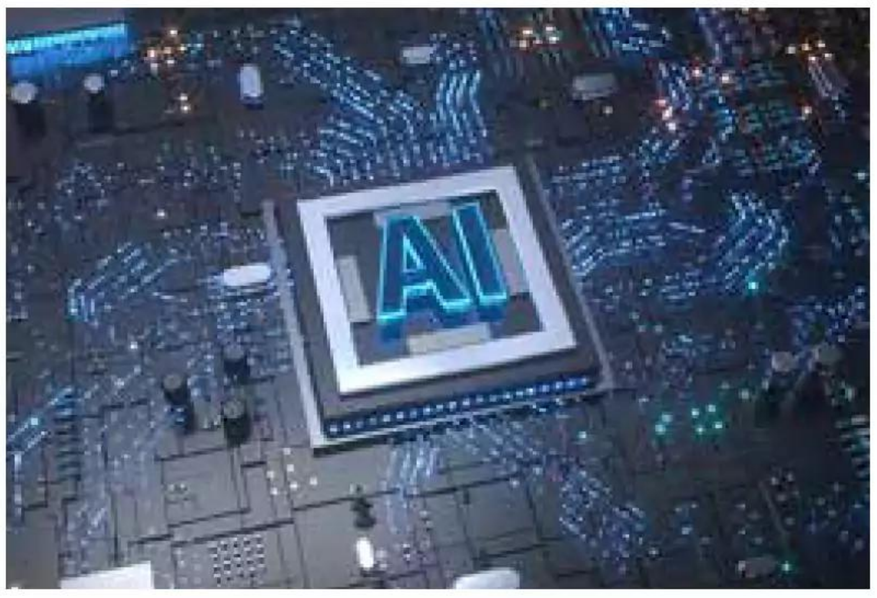
The expansion also aligns with Mumbai's growing role as a finance and digital gateway hub, supported by subsea cable landings and new international routes.

"NetActuate has had a presence in India for 10+ years and we've experienced continuous growth with our local and international partners," said Mark Mahle, CEO at NetActuate. "By enhancing our cloud platform in Mumbai with ONE capabilities and supporting customers with our Chennai PoP, we are giving enterprises and service providers the flexibility to choose the right mix of public, private, and hybrid cloud infrastructure while maintaining control over performance, cost, and data residency."

GLM5 brings open source parity to closed AI giants

Open source artificial intelligence has gained a heavyweight contender. Z.AI has launched GLM5, a 744-billion-parameter model released under the permissive MIT licence and hosted on Hugging Face, positioning community-led development directly against proprietary large language models.

Designed for autonomous coding and agentic engineering, GLM5 targets real-world software workflows rather than general chat. The system supports debugging, refactoring, repository navigation, and complex workflow automation, aided by 40 billion active parameters per task, training on 28.5 trillion tokens, and a 200,000-token context window that enables sustained reasoning across large codebases.



Benchmark results suggest the open source approach is closing the performance gap. GLM5 scored 77.8% on Software Engineering Bench and 56.2% on Terminal Bench 2.0, leading open source peers. It ranked highest on Vending Bench 2 for long-term operational capability and achieved 50.4% on Humanity's Last Exam, reportedly competing with or surpassing proprietary systems from OpenAI and Anthropic, as well as Gemini 3 Pro–class models.

The release signals a broader shift towards accessible, lower-cost, community-built AI that rivals closed alternatives in capability and scale.

Ant Group open sources two frontier AI models

Ant Group has open sourced two trillion-parameter frontier AI models — Ling-2.5-1T and Ring-2.5-1T — placing advanced large language and reasoning systems directly in the hands of developers and researchers worldwide. Both models are released under open licences and are publicly downloadable via Hugging Face and ModelScope, reinforcing the company's push towards an open AI ecosystem.



Ling-2.5-1T serves as the flagship general-purpose model. Built with one trillion parameters, it focuses on higher reasoning efficiency, fine-grained preference alignment and native agent interaction. It supports context windows of up to one million tokens while reducing compute overhead. On AIME 2026, the model matched frontier thinking systems using roughly 5,890 tokens, compared with the typical 15,000–23,000 tokens.

Ring-2.5-1T targets reasoning-heavy workloads and is positioned as the world's first hybrid linear-architecture thinking model. Designed for advanced logical and mathematical tasks, it achieved gold-tier scores on academic benchmarks, including 35/42 on IMO 2025 and 105/126 on CMO 2025.

Mozilla pushes for countering Big Tech's grip on AI

Mozilla is positioning itself as an open source counterweight to Big Tech's growing control over artificial intelligence, outlining plans to build what it calls a "rebel alliance" to reshape how AI is developed and governed.

The organisation wants AI to be trustworthy, human-centred, decentralised, and open rather than locked inside proprietary ecosystems.

"What if a rebel alliance of sorts—developers, activists, researchers, founders, investors—joined forces to build something different?" Mozilla asks, adding the coalition would "push AI in a direction that is trustworthy, human-centered and shaped by all of us."

Global banks are deploying Ethereum's Layer-2 stack

Banks are standardising on Ethereum's open source stack as production financial infrastructure, shifting from experimental pilots and proprietary blockchains to live Layer-2 networks for tokenised deposits, interbank payments, and cross-border settlement.

More than 30 institutions — including Bank of America, Citi, TD Bank Group and Wells Fargo — are working with SWIFT on an Ethereum-based platform for international transactions. Regulatory clarity is accelerating the move. The GENIUS Act has established rules for stablecoins, while additional market standards are under consideration, giving banks greater confidence to deploy blockchain rails at scale.

Use cases are already operational. JPMorgan's JPMCoin runs on Base, an Ethereum Layer-2. Citi, Vantage Bank and Custodia Bank issue tokenised, bank-backed US dollar deposits on Ethereum, with the Texas Bankers Association endorsing the Vantage–Custodia platform.

"It's very, very standard, very, very resilient, as we've seen across the ecosystem outside, but we run it on a very private basis. And because of that standardised nature, we think it gives us optionality to extend it and integrate it into other areas if possible," said Bis Chatterjee, global head of partnerships and innovation at Citi.

Despite alternatives such as Solana and Hyperledger, banks are converging on Ethereum, trusting open source, battle-tested code over building from scratch.



GitHub weighs turning off pull requests as AI slop floods projects

GitHub has formally acknowledged that AI-generated 'slop' is overwhelming open source projects, forcing maintainers to sift through poor pull requests (PRs), abandoned submissions and guideline violations — and is now considering restricting or even disabling pull requests, the core mechanism of open collaboration.

The Microsoft-owned platform is evaluating drastic measures, including turning off pull requests entirely, limiting them to trusted collaborators, deleting unwanted PRs from view, adding granular permissions, deploying AI triage tools, and introducing attribution to signal AI use — steps that could fundamentally reshape how open source functions.

Camilla Moraes, product manager at GitHub, called it "a critical issue affecting the open source community: the increasing volume of low-quality contributions that is creating significant operational challenges for maintainers." She added that GitHub is "actively investigating this problem and developing both immediate and longer-term strategic solutions."

The scale is stark. Xavier Portilla Edo, head of cloud infrastructure at Voiceflow, said: "1 out of 10 PRs created with AI is legitimate and meets the standards required to open that PR."



Firefox sits at the centre of that strategy. The browser will integrate AI strictly on an opt-in basis. Planned for 2026 are an AI Window and a centralised AI Controls dashboard that "give users one central place to manage AI features, even disable them completely. Don't want AI? Turn it off."



Beyond the browser, Mozilla's roadmap commits to investing in Firefox and Thunderbird, backing community initiatives and responsible startups, building open source AI developer tools, and improving the open source AI stack.

FFmpeg developers publicly call out AMD over a suspected AI-generated patch

Open source governance took centre stage after FFmpeg maintainers accused AMD of submitting unchecked, AI-generated code to the widely used multimedia project, publicly criticising the contribution as "AI slop" and rejecting the patch.



AMD had proposed changes to FFmpeg, one of the longest-running open source video and audio processing tools that underpins Chrome, Handbrake, OBS Studio and VLC Media Player. Developers reviewing the submission said the code appeared AI-generated and insufficiently vetted by a human.

Zhao Zilli, an FFmpeg developer, wrote: "Please ensure a thorough manual review of all AI-generated code before submission." Zilli added, "AI-generated code tends to be verbose; it requires the developer's own judgment and cleanup."

Maintainers flagged unnecessary extra comments, verbose structures typical of AI output, non-standard coding techniques, weak optimisation and signs of missing manual review.

Apple opens Xcode with Model Context Protocol to let any compatible agent plug in

Apple has recast Xcode as an open AI development hub. Xcode 26.3, released as a candidate build, integrates Anthropic's Claude Agent and OpenAI's Codex directly into the IDE, enabling autonomous or 'agentic coding'. The agents can write code, analyse project structure, access documentation, compile, run tests, capture screenshots for visual verification and iteratively fix errors with minimal oversight.



Unlike earlier autocomplete-style assistance, the agents now have full IDE-level control. They can build, detect failures, rebuild and validate results, while automatic checkpoints allow developers to roll back changes.

The bigger shift is architectural and strategic. Apple has standardised the integration on Model Context Protocol (MCP), an open standard created by Anthropic. Any MCP-compatible agent, not just Claude or Codex, can connect to Xcode, even from outside the app. The move marks a rare departure from Apple's historically closed ecosystem and opens the door to third-party and open source AI tools, positioning Xcode as a vendor-neutral agent platform.

Apple has also worked with Anthropic and OpenAI to reduce token costs and improve tool-calling efficiency. Agents install in one click and auto-update.

Limits remain. There is no direct MCP debugging tool, runtime issues still require manual intervention, and multiple agents cannot run simultaneously on the same project. Security concerns persist. David Mytton, founder and CEO of Arcjet, warned AI-developed software "into production will lead to catastrophic problems… There's going to be some big explosions coming!"

LibreOffice 26.2 comes with native Markdown support

LibreOffice 26.2 has been released by The Document Foundation, strengthening its position as a fully free and open source office suite for Windows, macOS, and Linux, with support for more than 120 languages. The community-driven update focuses on open standards, performance, compatibility, and privacy, offering organisations a credible alternative to proprietary productivity software.

The headline feature is native Markdown support. Users can now directly import Markdown files via file selection or clipboard and export documents back to the format, improving interoperability for technical documentation, developer workflows, and content management systems.



Performance has improved across the suite. EPUB export is faster and now includes a progress bar. Calc delivers smoother scrolling in sheets with many hidden columns and accelerates large tables, duplicate removal, and tracked-change rejection. Optimisations also cover SVG exports, 3D chart rendering, and Linux pattern fills.

Compatibility with Microsoft Office formats has been strengthened. Writer enhances DOCX export through Floating Table splitting and better Change Tracking preservation. Calc supports Excel's Biff12 clipboard format and defaults to Excel 2010+ XLSX, easing data exchange with proprietary environments.

Security receives a boost through ODF Wholesome Encryption, combining AES-GCM authenticated encryption with Argon2id key derivation to improve brute-force resistance and reduce metadata leakage.

LibreOffice 26.2 will be supported until November 30, 2026, with certified L3 providers offering extended commercial support for enterprises, schools, and governments.

"LibreOffice 26.2 shows what happens when software is built around users, not business models, and how open source software can deliver a modern, mature productivity suite without compromising user freedom," said Florian Effenberger, executive director of The Document Foundation.

For more news, visit *www.opensourceforu.com*

Microsoft's chemistry-focused QDK is open source

Microsoft has released an open source Quantum Development Kit (QDK) focused on quantum chemistry and error correction, expanding its Azure Quantum developer ecosystem. Announced on January 22, the tools are available now, with full availability expected through a phased rollout concluding in 2026.

A key differentiator of the release is cross-framework compatibility. Developers can work seamlessly across Q#, OpenQASM, IBM Qiskit, and Google Cirq within a single environment, eliminating the need to switch tools or rewrite programs. By embracing Qiskit rather than competing against it, Microsoft positions itself as a framework-agnostic platform, directly challenging ecosystem lock-in.

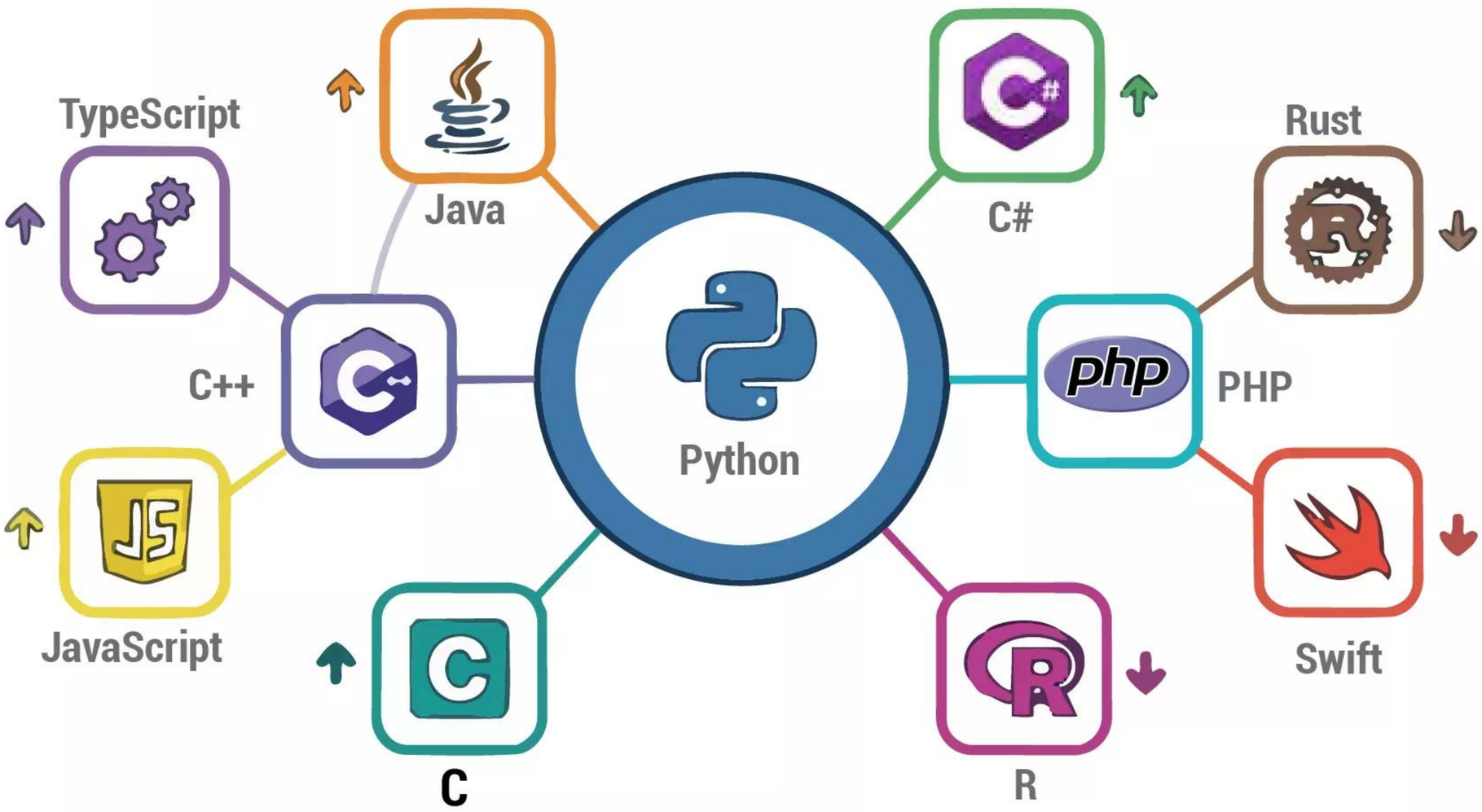
The QDK integrates deeply with existing developer workflows through native VS Code support, including Python and Jupyter integration, circuit rendering, IntelliSense, breakpoint debugging, local simulators, and resource estimation. GitHub Copilot further accelerates code generation, testing, and job submission.

"With GitHub Copilot and the QDK, programming tasks such as code generation, unit tests, and job submissions are faster and easier than ever before," said Matthias Troyer, technical fellow and corporate vice president of Quantum at Microsoft.

Industry validation came from Algorithmiq, a Finnish quantum computing startup specialising in computational life sciences. "We believe the modular design of the platform will be a true game changer for quantum chemistry," said Guillermo Garcia-Perez, chief scientific officer at Algorithmiq.

Revisiting Popular Programming Languages: The 2026 Edition

As always, this ranking of the Top 10 popular open source programming languages is based on four other rankings. A new entrant this year is Rust.



Over the past decade, I have contributed three articles to *Open Source For You (OSFY)* focusing on the ranking of popular programming languages. The first article on this subject, titled '10 Most Popular Programming Languages', was published in 2017. It is a difficult as well as controversial task to rank programming languages for popularity or usefulness. So, rather than depending on a single ranking, I considered the four most popular rankings of programming languages: the RedMonk programming language ranking, the PYPL (PopularitY of Programming Language Index) ranking, the IEEE Spectrum ranking of programming languages, and the TIOBE index. The article is available online on the *OSFY* portal at *https://www.opensourceforu.com/2017/03/most-popular-programming-languages/*. My top 10 list of programming languages that year was C, C#, C++, Go, Java, JavaScript, PHP, Python, Ruby, and Swift (in alphabetical order, not in the order of popularity).

After a gap of exactly two years, in February 2019, I wrote another article titled 'Yet Another Top Ten List of Popular Programming Languages', available at *https://www.opensourceforu.com/2019/03/yet-another-top-ten-list-of-popular-programming-languages/*. I used the same criteria as in 2017 to compile the Top 10 list, and the programming languages picked this time were C, C#, C++, Java, JavaScript, Objective-C, PHP, Python, R, and Swift. Objective-C and R entered the Top 10 list, replacing Go and Ruby. The languages that looked especially promising in 2019 were Python and JavaScript, while Ruby and Perl showed a noticeable decline in popularity.

The article concluded with a section on new and emerging programming languages that had been first released within the previous decade (2009 or later). This list included seven languages: Swift, Go, Rust, Kotlin, Julia, TypeScript, and Dart.

The last article in this series was published in February 2021. The article, titled 'The Latest Trends in the Programming World', is available at *https://www.opensourceforu.*

com/2021/03/the-latest-trends-in-the-programming-world/. The same criteria were used to compile the Top 10 list, and the programming languages were: C, C#, C++, Java, JavaScript, PHP, Python, R, Swift, and TypeScript. In this edition, Objective-C was replaced by TypeScript. Notably, eight languages consistently appeared in the Top 10 across all three of my rankings from 2017 onwards: C, C#, C++, Java, JavaScript, PHP, Python, and Swift. This article also identified the promising programming languages. In addition to Python and JavaScript, which were highlighted in 2019, R and TypeScript showed strong promise for rapid adoption in the programming world. Like the earlier edition, a list of languages in decline was compiled, featuring Perl (also present in the 2019 list) and Objective-C, whereas Ruby showed improvement and was removed from the list.

The list of promising young programming languages in 2021 remained largely the same as in 2019, with the addition of an eighth language, Solidity, which is used for blockchain-based applications. The final list therefore included Swift, TypeScript, Go, Rust, Kotlin, Dart, Julia, and Solidity.

Now that five years have passed since the last article with this theme was published, it is time to analyse the latest trends in the programming world. Our analysis will be based on the same four rankings used in the previous three articles mentioned earlier. This article is particularly relevant in light of two major developments since the last edition of the Top 10 programming languages: the emergence of ChatGPT, Gemini, and other large language models (LLMs), and significant research breakthroughs in quantum computing. It is high time to analyse how these impactful technologies have affected the world of programming.

The Top 10 list

We now examine how recent developments in computer science have shaped trends in the programming world. As a starting point, we present a Top 10 list of programming languages. The ranking in this article is obtained by aggregating the latest results from the four rankings introduced earlier. Drawing on multiple indices helps mitigate the biases inherent in any single ranking and offers a more balanced assessment. Six languages—Python, Java, JavaScript, C, C++, and C#—appear consistently across all four ranking schemes. These languages were also permanent fixtures in our previous rankings and continue to retain their positions in 2026, underscoring their enduring relevance and influence in modern computing.

The next two languages included in our Top 10 list are PHP and R. Each has recorded two Top 10 finishes and four Top 20 finishes across the rankings we considered. PHP had consistently appeared in the Top 10 in our previous three editions; however, its current positions—12th in the IEEE Spectrum programming languages ranking and 15th in the TIOBE index—suggest a decline in its usage over the past five years. R, by contrast, ranks 12th in the RedMonk ranking and 11th in the IEEE Spectrum ranking, placing it slightly ahead of PHP in overall popularity.

However, the final two spots in the Top 10 list are closely contested among TypeScript (2 Top 10 finishes and 3 Top 20 finishes), Rust (1 Top 10 finish and 4 Top 20 finishes), Swift (1 Top 10 finish and 3 Top 20 finishes), Go (1 Top 10 finish and 3 Top 20 finishes), and Kotlin (4 Top 20 finishes). To resolve this tie, I selected the two languages with the best average ranking across the four rankings considered: RedMonk, PYPL, IEEE Spectrum, and the TIOBE index. Rust is ranked 19, 7, 14, and 13 respectively, yielding the strongest average among the candidates at 13.25. Swift follows with rankings of 11, 6, 16, and 22, resulting in an average of 13.75. The remaining contenders trail slightly behind: TypeScript averages 14.25 (6, 12, 7, 32), Go averages 14.75 (12, 21, 10, 16), and Kotlin averages 16 (14, 17, 13, 20). Based on these aggregate rankings across independent measures, Rust and Swift emerge as the most consistent performers and therefore complete the final two positions in the Top 10 list.

In the previous three rankings, we only considered whether a language appeared in the top 10 or not, without assigning a numeric order within that group. In this article, however, we apply the same averaging method used earlier to break ties and produce a relative popularity ranking of the ten selected languages, arranging them from most to least popular. Thus, the final ranking based on average positions across the four indices is as follows:

1. Python
2. Java
3. C++
4. JavaScript
5. C
6. C#
7. R
8. PHP
9. Rust
10. Swift

Before drawing conclusions from the inclusions and omissions in the list, it is useful to compare the four ranking schemes and the biases inherent in each. We begin with the RedMonk ranking, which reports only the top 20 programming languages. Its methodology relies on two parameters: GitHub activity and Stack Overflow discussions. As a result, this ranking serves as a strong indicator of developer engagement and community activity. However, it tends to favour programming languages with vibrant open source ecosystems while under-representing programming languages used primarily in closed enterprise environments. Notably, this is the only ranking in which Python is placed second, narrowly trailing JavaScript.

The PYPL ranking, which reports only the top 30 programming languages, is based on how frequently programming language tutorials are searched on Google. It therefore reflects learning demand rather than direct industry usage. This approach naturally favours beginner-friendly languages and those widely taught in academic institutions. The relatively high position of C illustrates this tendency.

The IEEE Spectrum ranking is the most comprehensive among the schemes considered, incorporating a wide range of indicators such as Google search trends, GitHub activity, Stack Overflow data, job postings, academic publications, and social media signals. By combining these diverse metrics, it attempts to balance perspectives from academia, industry, and the developer community. As a result, the IEEE Spectrum ranking is less biased towards any single measure and serves as a strong indicator of overall ecosystem relevance. Among the four ranking schemes, it is also the most closely aligned with industry demand. The relatively high ranking of Kotlin illustrates this tendency.

Finally, we consider the TIOBE index. Although it lists the largest number of programming languages, its rankings can sometimes be confusing and potentially misleading. The TIOBE index is based on measures of visibility, including search engine hits, indexed web pages, books, tutorials, courses, vendor materials, and job mentions. Its primary limitation is that it reflects online presence rather than actual industry usage. Therefore, older programming languages often appear much higher in the rankings than languages currently favoured in modern software development. For example, Perl is ranked 11th, Fortran 12th, and Ada 18th. I have seen a lot of things in my life, but I have yet to see a professional Ada programmer in person. Programming languages like Fortran and Ada have significant historical importance and remain relevant in specialised domains, but their high rankings do not necessarily reflect mainstream industry adoption.

We now turn to the trends emerging from our ranking. The most noticeable shift is the relatively weaker performance of PHP. In all previous editions, PHP consistently appeared in the top 10 across all four rankings considered. This time, however, it is placed in the top 10 in only two of them. At present, it is too early to draw firm conclusions about PHP's future. This may represent a short-term fluctuation, or it could signal a gradual decline in the language's popularity.

Another surprising finding is the absence of TypeScript from the Top 10 list. This outcome is largely driven by its low ranking—32nd—in the TIOBE index. In contrast, the IEEE Spectrum ranking places TypeScript in 7th position, just below JavaScript, highlighting its strong relevance in modern software development. The comparatively weak showing in the TIOBE index may be partly explained by TypeScript's relative youth (its first version was released in 2012) and by the likelihood that many TypeScript-related searches are counted under JavaScript.



Figure 1: The logo of Rust

Rust's first entry into the Top 10 list is another important trend. Given that it was released only in 2012, this represents a significant achievement for the language. Figure 1 shows the logo of Rust.

Winners, losers, and the under-17 contenders in the programming language race

Now let us identify the languages that are gaining momentum. As in our previous rankings, the clear winner is Python. Its continued dominance is not surprising, even amid the rapid expansion of AI and quantum computing. In fact, these very trends reinforce Python's position. Python remains the primary language for AI development, and leading quantum computing toolkits such as Qiskit are built around it. Rather than slowing Python's progress, the explosive growth of AI applications and quantum simulations has accelerated its momentum and strengthened its leadership.

Another important trend is that, unlike 2019 or 2021, there are no clear losers. Back in 2019 I thought that Ruby and Perl might be doomed. But in just two years I had to retract my words about Ruby. My 2021 list had Perl and Objective-C as languages destined for oblivion. However, after five years I am retracting my prediction about Perl. Not only is Perl still around, it is also surviving even after Perl 6 was officially renamed Raku and is considered a programming language independent of Perl (Perl 5). Similarly, Objective-C has 1 Top 10 finish, 3 Top 20 finishes, and ranks 29th in the TIOBE index — not the signs of a language in decline. So, I have an empty list of languages that are losing the battle for dominance in the programming world.

Finally, we turn to the under-17 contenders in the race for the programming language championship. The original 2019 edition focused on languages first released in 2009 or later. Today, these languages have matured, yet even the oldest among them is no more than 17 years old. The list compiled in 2019 included Swift, TypeScript, Go, Rust, Kotlin, Dart, and Julia. With the addition of Solidity in 2021, it expanded to eight promising programming languages.

What trends can we infer for 2026 from the performance of these languages? Swift, TypeScript, Go, Rust, Kotlin, and Dart each record at least three Top 20 finishes, indicating strong and sustained visibility across rankings. This consistency suggests that they have secured firm positions

in both industry and academia. Based on the available data, it is reasonable to expect that some of these languages will remain permanent fixtures in the Top 10, while the others are likely to hold stable positions within the Top 20.

Julia presents a less straightforward case. It appears only once in the Top 20, and that too at the 20th position in the PYPL ranking. It does not appear in the RedMonk Top 20 list, ranks 31st in the IEEE Spectrum ranking, and 28th in the TIOBE index. Despite this modest showing, Julia is still a relatively young language, with its first release in 2012. Given its strengths in scientific and high-performance computing, there remains reason for optimism, and it deserves to be retained among the promising programming languages to watch in 2026.

Solidity, the programming language I added in 2021 to the 2019 list of seven promising languages, entered the discussion at a time when blockchain technology was at the peak of its popularity. Designed for implementing smart contracts on blockchain platforms, Solidity was widely viewed as a potential breakout language. However, its recent performance suggests that it may not become the next major force in the programming world after all. As blockchain technology has moved beyond its hype phase into a more mature phase, expectations surrounding Solidity have also become more grounded. With a 25th rank in the IEEE Spectrum ranking and a 41st rank in the TIOBE index, Solidity no longer appears to be an imminent breakout star, particularly given its domain-specific nature. Reluctantly, I therefore retract my earlier prediction and remove Solidity from the list of promising programming languages.

But should we include any new language in our list of promising contenders? I must admit I hesitated, especially after having to retract my earlier optimism about Solidity. However, after careful research, I have decided to add one more language to the list: Zig, a systems programming language designed as a modern, general-purpose improvement over C. Zig is free and open source software released under the MIT License. Since it was first released in 2016, it is still very much in its formative years.

How does Zig perform in the rankings we have considered? It is currently placed at the 24th position in the PYPL ranking and appears within the top 100 of the TIOBE index. While these positions do not yet signal dominance, they indicate steady visibility and growing interest. That alone is encouraging enough to justify watching Zig closely. I plan to revise this list in a couple of years, and it will be interesting to see how far Zig has progressed by then. Figure 2 shows the logo of Zig.



Figure 2: The logo of Zig

With Zig's inclusion, the final list of promising programming languages now consists of Swift, TypeScript, Go, Rust, Kotlin, Dart, Julia, and Zig.

What the Stack Overflow 2025 Developer Survey says

Now let us look at some insights from the Stack Overflow 2025 Developer Survey, one of the most comprehensive global surveys on programming and developer trends. After the United States and Germany, India ranks third in the number of respondents, highlighting the country's growing technology adoption and expanding developer community.

Some results from the survey help explain Rust's first-time entry into the Top 10 list. The survey distinguishes between desired and admired languages, and Rust ranks as the most admired programming language at 72%. Now consider Cargo, Rust's build system and package manager -- according to the survey, Cargo is the most admired cloud development and infrastructure tool in 2025. The survey also highlights the popularity of uv, an extremely fast Python package and project manager written in Rust, further reflecting Rust's growing influence in modern programming.

The survey also supports my decision to include Zig in the list of promising programming languages. Zig records a strong admiration score of 64%, making it the fourth most admired programming language. Perhaps this time the prediction will hold more firmly than my earlier optimism about Solidity. The survey also reinforces Python's global dominance.

I hope this analysis proves useful to students of computer science as well as working professionals, since understanding programming language rankings can help you evaluate whether your current area of expertise remains relevant and which languages may be worth learning next. At the same time, we must ask a deeper question: will programming look the same a decade from now, given the rapid rise of advanced AI tools? Could we enter a phase where many traditional programming roles are replaced by prompt engineering? It is a possibility — but it is equally certain that programming itself will not disappear. The tools may evolve, but skilled programmers will continue to shape how technology is built and used.

With that thought, I bid you farewell until we meet again to discuss the next wave of trends in the programming world in a few years' time. END

By: Dr Deepu Benson

The author is a free software enthusiast, and his area of interest is theoretical computer science. He is an assistant professor (senior scale) at Manipal Institute of Technology, Bengaluru, and maintains a technical blog.

EFY interactive

Asia's #1 **Digital Marketing Agency** for Electronics and Tech Industry

WE SPECIALISE IN B2B + TECH MARKETING

Digital Marketing | Events | Publications | Databases



An EFY GROUP DIVISION

BRANDS WE WORK WITH...

infineon | MOUSER ELECTRONICS | Altium | KEYSIGHT

power integrations | RG | AVNET | MICROCHIP

DigiKey | VISHAY | ROHM | TAIYO YUDEN

VITZROCELL | nichicon | MAXIM Power Corp | MAXIM SMT

Google | Henkel | HETZNER | HARTING

SIEMENS | ANALOG DEVICES | Fibocom | Chroma

HARWIN | muRata INNOVATOR IN ELECTRONICS | Littelfuse | TEME Electronic Components

ecia Connect. Influence. Optimize | Anritsu Advancing beyond | DELVI TECH | onsemi

For more information:

+91-98111-55335 | growmybiz@efy.in



The Path to Cybersecurity in the Quantum Era

The rise of quantum computing will be accompanied by a failure of conventional cryptography. Post-quantum cryptography and advanced threat detection methodologies, among other techniques, are being evolved to counter security threats in the quantum era.

Digital security is undergoing a profound transformation as quantum computing transitions from theoretical promise to practical reality. For security professionals, researchers, and IT leaders, understanding quantum security has become essential. Quantum computers, with their immense computational power, threaten to undermine the foundations of classical cryptography, making previously unbreakable encryption susceptible to rapid decryption.

Quantum security algorithms are poised to fundamentally reshape the way we think about safeguarding information in the digital age. Unlike traditional cryptographic methods, which often depend on the immense difficulty of solving mathematical puzzles using classical computers, quantum-resistant algorithms are designed with the express intent of surviving the powerful capabilities of quantum machines. This distinction is crucial, as quantum computers threaten to render many established encryption techniques obsolete by solving problems once considered infeasible in a fraction of the time.

What sets quantum security algorithms apart is their proactive approach to anticipating the evolving threat landscape. Rather than reacting to emerging risks, these algorithms are conceived to withstand the unique and powerful attack vectors unleashed by quantum computing. By embedding resilience into the very foundation of cryptographic protocols, they help ensure that confidential information,

critical communications, and essential services remain protected, regardless of technological advances. As organisations and governments move towards adopting these standards, quantum security algorithms are becoming an indispensable cornerstone for the future of data protection, inspiring confidence in the continued safety of digital interactions and transactions.

Post-Quantum Cryptography (PQC) represents a pivotal advancement in the ongoing evolution of cryptographic security, emerging in direct response to the accelerating development of quantum computing technologies. The primary aim of PQC is to develop and standardise cryptographic algorithms that can withstand the unprecedented computational capabilities of quantum machines. As traditional public key cryptosystems, which currently underpin secure communications and digital trust, face obsolescence in the quantum era, the adoption of robust PQC algorithms is essential for maintaining the confidentiality, authenticity, and integrity of sensitive information.

Quantum security algorithms — the foundation of data protection in the future

Lattice-based cryptography algorithms: These draw upon advanced branches of mathematics to create formidable barriers against even the most sophisticated quantum attacks. Lattice-based cryptography leverages the inherent complexity of lattice problems, which quantum computers find exceptionally challenging to break. Similarly, hash-based cryptography relies on the irreversible nature of cryptographic hash functions, making it difficult for adversaries to reconstruct original data. Code-based and multivariate polynomial-based schemes further expand this repertoire, offering a variety of mechanisms that collectively strengthen the protective framework around sensitive digital assets.

Shor's algorithm: The critical security threat posed by quantum computing arises from Shor's algorithm, a groundbreaking quantum algorithm devised by mathematician Peter Shor. Shor's algorithm efficiently factors large integers and computes discrete logarithms, tasks that are prohibitively time-consuming for classical computers. This efficiency directly undermines the security basis of widely used public key systems such as RSA, which relies on the difficulty of factoring large numbers, and Elliptic Curve Cryptography (ECC), which depends on the hardness of the discrete logarithm problem over elliptic curves. In theory, a sufficiently powerful quantum computer running Shor's algorithm could break these systems in a matter of hours, rendering encrypted communications, digital signatures, and key exchanges vulnerable to interception and forgery. This imminent risk has accelerated the global search for quantum-resistant alternatives to current cryptographic standards.

CRYSTALS-Kyber algorithm: PQC algorithms based on lattice problems are believed to be resistant to both classical and quantum attacks. Among these, CRYSTALS-Kyber has gained significant attention as a key encapsulation mechanism (KEM) designed for secure key exchange. This algorithm utilises the hardness of the Learning With Errors (LWE) problem, a mathematical challenge for which no efficient quantum solution is known. In practice, when two parties wish to establish a shared secret over an insecure channel, CRYSTALS-Kyber enables them to exchange messages in such a way that an eavesdropper, even with access to a quantum computer, cannot feasibly compute the secret key. For instance, in a secure messaging application, Alice can use Kyber to generate a public key and share it with Bob. Bob then uses this public key to encapsulate a secret, sending the resulting ciphertext back to Alice. Only Alice, with her private key, can decapsulate the ciphertext and retrieve the shared secret, ensuring both confidentiality and forward secrecy in the presence of quantum threats.

CRYSTALS-Dilithium algorithm: Complementing key exchange mechanisms, secure digital signatures are indispensable for authentication and non-repudiation in digital transactions. The CRYSTALS-Dilithium algorithm, also founded on lattice-based cryptography, establishes a quantum-resistant digital signature scheme. Built on the Module-LWE and Module-SIS problems, it offers strong security guarantees and efficient implementation. In a typical scenario, an organisation may use Dilithium to sign software updates before distribution. The developer generates a signature using their private key, which is then attached to the update package. Recipients, such as end users or automated systems, can verify the authenticity of the update using the corresponding public key, confident that even a quantum-enabled adversary cannot forge signatures or impersonate the sender. This ensures the integrity and provenance of critical software, protecting against tampering and supply chain attacks.

As quantum computing continues to advance, the necessity for robust post-quantum cryptography solutions becomes ever more apparent. Understanding the distinguishing features, strengths, and typical applications of various algorithms is essential for organisations seeking to safeguard their digital assets against future quantum-enabled threats (see Table).

Key algorithms shaping the post-quantum cryptography landscape

Algorithm	Features	Common use cases
Lattice-based cryptography algorithms	Built on the mathematical difficulty of lattice problems, these algorithms are designed to be resistant to both classical and quantum cyber attacks. They offer strong security guarantees and can be efficiently implemented, making them one of the most promising approaches for future cryptographic standards.	Securing digital communications, protecting confidential data, and serving as the foundation for advanced encryption and digital signature schemes in quantum-resilient systems.
Shor's algorithm	Unlike the others, Shor's algorithm is a quantum algorithm that can efficiently factor large integers and solve discrete logarithms. Its primary significance lies in its ability to break widely-used classical encryption techniques, such as RSA and ECC, if run on a sufficiently powerful quantum computer.	Primarily a threat model for existing cryptographic systems, Shor's algorithm is not used for securing data but serves as the catalyst for the development and adoption of quantum-resistant cryptography.
CRYSTALS-Kyber algorithm	This is a lattice-based key encapsulation mechanism that utilises the Learning With Errors (LWE) problem, making it highly resistant to quantum attacks. It is known for efficient key exchange and strong security even in the presence of quantum adversaries.	Ideal for establishing secure shared secrets over insecure channels, such as in encrypted messaging platforms and secure online transactions, ensuring that eavesdroppers cannot derive the exchanged keys.
CRYSTALS-Dilithium algorithm	Also founded on lattice-based constructs, Dilithium is a digital signature scheme relying on Module-LWE and Module-SIS problems. It offers a blend of robust security and practical performance, enabling swift signature generation and verification.	Widely used for authenticating software updates, verifying transactions, and ensuring the integrity and origin of digital communications, especially in environments where resistance to quantum attacks is paramount.

Lattice-based cryptography stands out for its flexibility and strong quantum resistance, making it a cornerstone for next-generation encryption and signature mechanisms. In contrast, Shor's algorithm represents a pivotal breakthrough in quantum computing, serving as a reminder of the vulnerabilities in current cryptographic standards and driving the urgent transition to post-quantum solutions. CRYSTALS-Kyber offers a practical and secure method for exchanging keys, ensuring confidentiality in the face of both classical and quantum threats. Meanwhile, CRYSTALS-Dilithium guarantees authenticity and non-repudiation for digital signatures, addressing the growing need for trust in software distribution and electronic transactions.

These algorithms are being actively standardised and integrated into security protocols, providing a path for organisations to transition away from vulnerable systems and embrace quantum-resilient architectures.

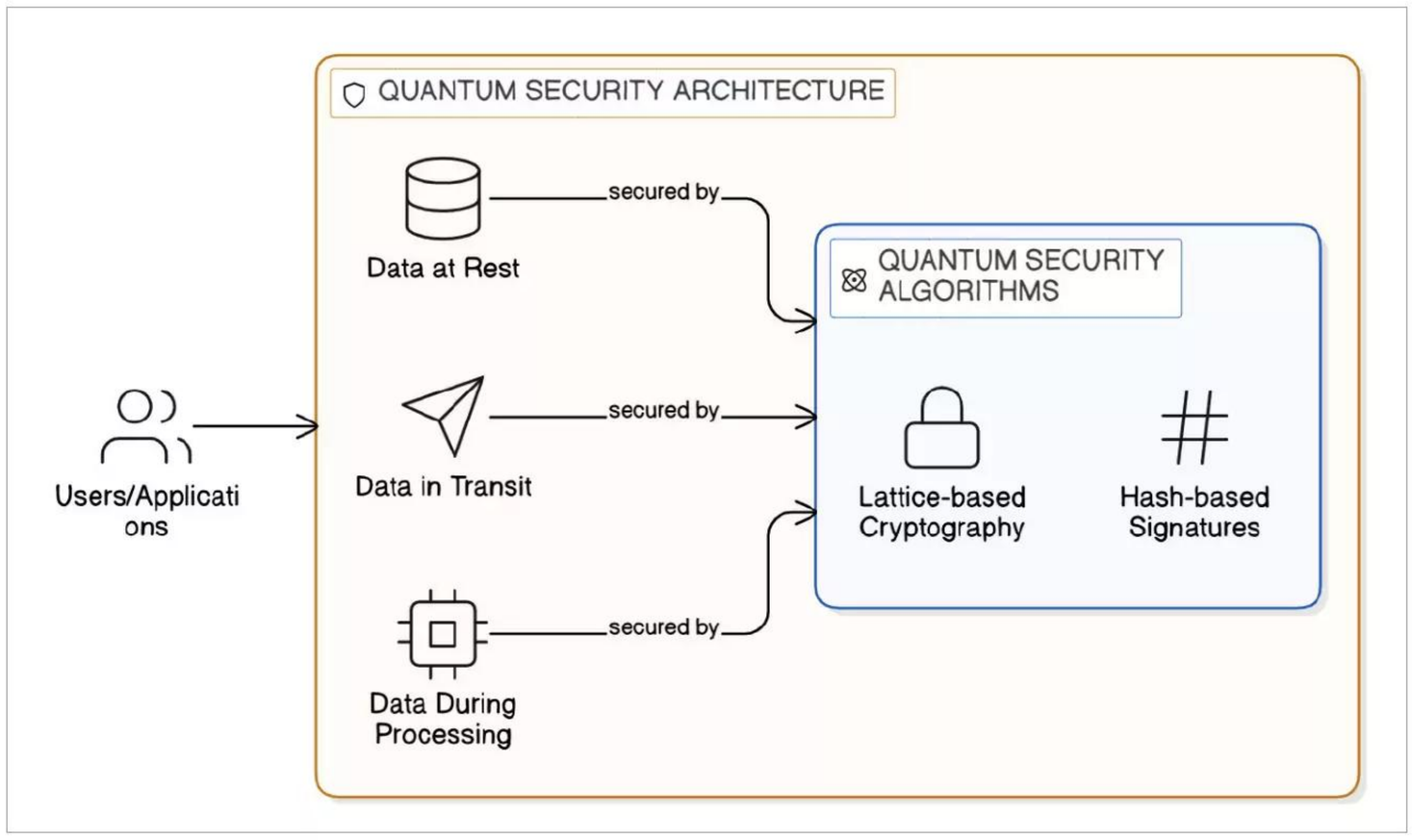


Figure 1: Quantum security reference architecture

Post-quantum readiness assessment

As quantum threats loom on the horizon, assessing organisational readiness for post-quantum security becomes imperative. A thorough readiness assessment involves reviewing existing cryptographic assets, identifying vulnerable systems, and estimating the exposure to quantum-enabled attacks. This evaluation extends to strategic planning, resource allocation, and workforce training, ensuring that organisations are equipped to

implement quantum-resistant solutions. The importance of such assessments cannot be overstated, as proactive preparation is key to mitigating risks and maintaining operational resilience. Effective readiness assessments enable security leaders to construct roadmaps for migration, prioritise critical assets, and align business objectives with emerging technological demands.

Threat detection and attack modelling using knowledge graphs

Threat detection and attack modelling in the age of quantum computing require innovative strategies that go beyond traditional security approaches. One such advancement is the use of knowledge graphs, which provide a sophisticated means to map out and understand the intricate web of relationships between entities, events, and vulnerabilities within an organisation's digital environment. Knowledge graphs offer a structured framework, enabling security teams to integrate and analyse vast amounts of data from multiple sources. This interconnected representation allows for the identification of nuanced patterns and hidden correlations that might otherwise remain undetected, especially those associated with quantum-enabled threats.

By leveraging knowledge graphs, organisations can simulate potential attack vectors, including those that exploit the unique capabilities of quantum technology. These models help security professionals anticipate how quantum adversaries might target critical assets, exposing gaps that need to be addressed before they can be exploited. The dynamic nature of attack modelling using knowledge graphs supports continuous refinement as new threats emerge and as the organisational landscape evolves. As a result, security teams gain a holistic perspective that enhances their situational awareness, enabling them to respond more effectively to emerging risks. In essence, the adoption of knowledge graph-based methodologies elevates the organisation's ability to detect threats early, model complex attack scenarios, and reinforce the overall security posture in preparation for the quantum era.

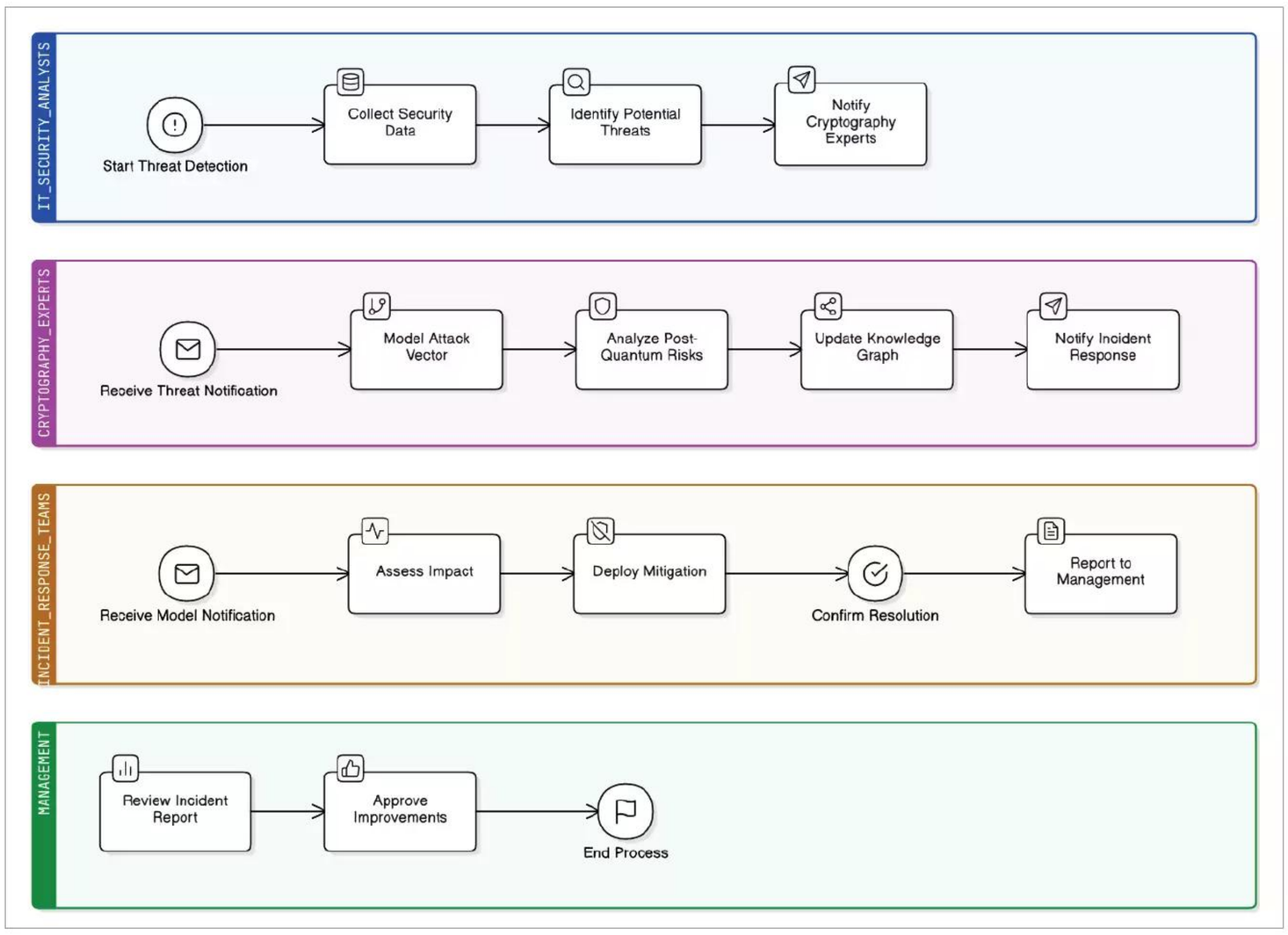


Figure 2: Business process diagram of thread detection and attack modelling

Continuous monitoring for quantum threats and vulnerabilities

Quantum continuous monitoring is driven primarily by the 'harvest-now, decrypt-later' cyberthreat – adversaries collect encrypted data today and decrypt it in the future when quantum systems are mature enough to do so. Thus, monitoring needs to be threat-informed for quantum capability signals and cryptography-informed for tracking wherever breakable or weak cryptography is still in use.

Quantum threat intelligence and horizon tracking: A revamped programme regularly fetches quantum-aligned and ecosystem readiness-related indicators: vendor roadmaps, error-correction achievements, logical qubit forays, and standardisation updates. This stream feeds a Quantum Risk Register which is reviewed as vulnerability exposure.

Cryptographic asset discovery and 'Y2Q' telemetry: Organisations need constant visibility of where RSA/ECC, weak parameter sets and non-agile crypto are deployed. This is done by Crypto Bill of Materials (CBOM) and automated crypto scanners for code, containers, firmware, PKI, and SaaS configurations.

Hybrid deployment and monitoring that is safe to rollback: Hybrid key exchange (classical + PQC KEM) is about the only thing that's standard while migrating to de-risk adoption. Monitoring must validate:

- Proper negotiation (no silent downgrade), matching sets of parameters, and library versions.
- Behaviour and failure scenarios (handshake latency, fragmentation, retransmissions) with respect to identifying operational regressions that could also be exploited by attackers.

Handling vulnerabilities in PQC implementations: Continuous monitoring should include:

- Side-channel exposure analysis (timing/cache), RNG monitoring, constant-time-testing, and dependency CVE analysis in PQC-

enabled TLS/VPN stacks.
- Key encapsulation/decapsulation error handling, logging hygiene, and key lifecycle security controls.

Automation of response and governance: Organisations must tie detections to playbooks, auto-ticket crypto debt, quarantine non-compliant services, enforce crypto-agile baselines, and produce audit-ready evidence that migration is measurable, continuous and resistant against quantum-enabled adversaries.

Explainable security metrics in PQC

Explainable security metrics are essential in post-quantum cryptography (PQC) because the migration is both technical and organisational. Leaders must understand why an algorithm choice, parameter set, or hybrid mode meaningfully reduces quantum risk. In practice, explainability means metrics that are traceable to standards, grounded in threat models, and actionable for engineers.

Standard-aligned strength metrics (what level of protection?): PQC deployments should report security using NIST security categories and the specific primitive role:

- Key establishment (e.g., lattice-based KEMs such as CRYSTALS-Kyber, standardised by NIST as ML-KEM).
- Digital signatures (e.g., CRYSTALS-Dilithium, standardised as ML-DSA).

Metrics should clearly state target category, parameter set, and whether the configuration is hybrid (classical + PQC) or PQC-only.

Attack-surface explainability (why is this secure?): Replace vague 'quantum-safe' labels with evidence-oriented indicators:

- Assumption mapping (e.g., LWE/Module-LWE hardness), known reduction notes, and what is not assumed (no reliance on factoring/discrete log).
- Implementation posture: constant-time status, side-channel hardening level, and dependency integrity (library provenance, reproducible builds, SBOM/CBOM coverage).

Migration readiness metrics (how exposed are we today?): To counter harvest-now, decrypt-later exposure, organisations should quantify:

- *Quantum exposure window:* Proportion of sensitive traffic/data encrypted with vulnerable algorithms and the retention duration.
- *Crypto agility score:* Ability to rotate algorithms/parameters without outages (coverage across TLS, VPN, PKI, IAM, HSMs).
- *PQC coverage:* Percentage of endpoints/services successfully negotiating PQC or hybrid handshakes, with downgrade-resistance verification.

Operational metrics (will it work at scale?): Explainable metrics must also address performance and reliability trade-offs:

- Handshake latency deltas, CPU/memory overhead, failure rates, and interoperability exceptions linked to Kubernetes services and remediation steps.

Humanised reporting: dashboards that justify decisions: The output should read like an engineering narrative: 'Asset X stores data for 7 years → RSA exposure is unacceptable → move to hybrid ML-KEM + classical today, ML-DSA for signing next → residual risks are implementation CVEs and downgrade attempts, monitored continuously'. This closes the loop between cryptographic theory, operational reality, and accountable governance.

Quantum security stands at the forefront of the next wave of technological advancement, demanding a proactive, informed approach from security professionals and leaders. The deployment of quantum-resistant algorithms, the adoption of post-quantum cryptography, and the rigorous assessment of organisational readiness are foundational steps in safeguarding data and infrastructure. Advanced threat detection methodologies, continuous monitoring, and transparent security metrics further reinforce defences, enabling organisations to navigate the uncertainties of the quantum era. As quantum computing continues to evolve, the commitment to quantum security will determine the resilience and trustworthiness of digital ecosystems for years to come. END



References

- *https://www.nomios.com/resources/what-is-quantum-security/*
- *https://thequantuminsider.com/2023/07/17/quantum-security/*
- *https://thequantuminsider.com/2020/02/07/quantum-programming-101-shors-algorithm/*
- *https://medium.com/identity-beyond-borders/crystals-kyber-the-key-to-post-quantum-encryption-3154b305e7bd*
- *https://www.ibm.com/docs/en/zos/2.5.0?topic=cryptography-crystals-dilithium-digital-signature-algorithm*
- *https://www.kyndryl.com/ca/en/perspectives/articles/2025/10/quantum-safe-networks*

By: Dr Magesh Kasthuri and Dr Anand Nayyar

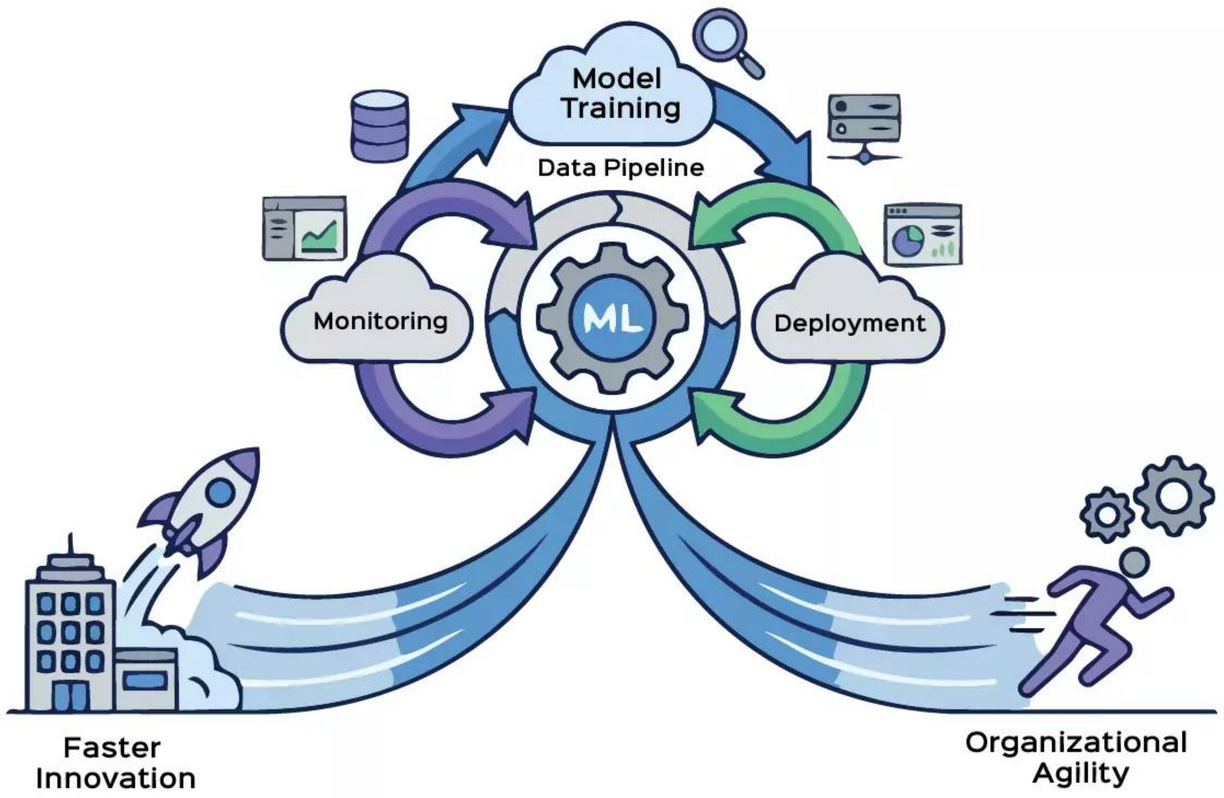
Dr Magesh Kasthuri is a PhD in artificial intelligence and the genetic algorithm. He currently works as a distinguished member of the technical staff (master) and chief architect at Wipro Ltd. This article expresses his views and not of the organisation he works in.

Dr Anand Nayyar is a PhD in wireless sensor networks and swarm intelligence. He works at Duy Tan University, Vietnam, and loves to explore open source technologies, IoT, cloud computing, deep learning, and cyber security.

FOCUS

Artificial Intelligence

MLOps for Enterprise Agility: An Overview



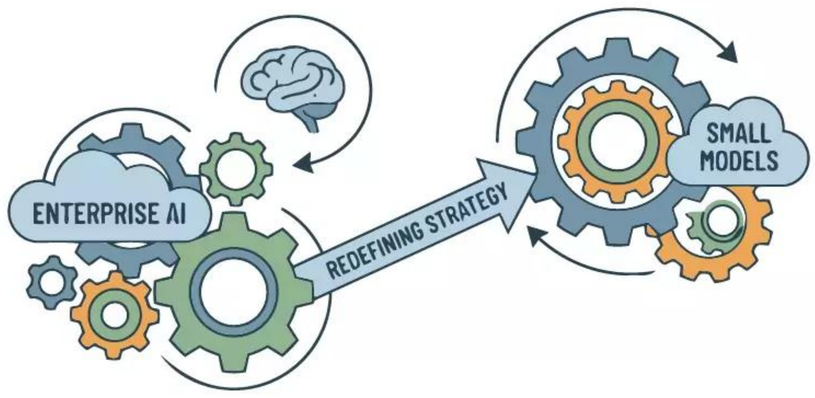
MLOps helps enterprises develop, deploy, and manage machine learning systems that are reliable, scalable, and production ready... *Read more on page.....32*

Moving Towards New Frontiers with Generative AI



Generative AI is making its mark in diverse fields, but it must be nurtured with care to ensure it's free of bias and preserves human values.*Read more on page.....45*

The Quiet Revolution: How Small Language Models are Redefining Enterprise AI Strategy



Small language models are proving to be game changers in the AI world. Deploying these models early on can give organisations an edge over the competition.

Read more on page.....50

The Transformation of Prompts into Source Code

Prompts are no longer simple text. As generative AI evolves, they are turning into engineering assets, and more specifically, code that needs to be designed, tested, versioned, and maintained. *Read more on page.....60*

How to Fine-Tune LLMs for Domain-Specific Adaptation

In this first part of a two-part series, discover why we need to fine-tune large language models, and get introduced to the core concepts and techniques used for their domain-specific adaptation.

Read more on page.....63

The Relevance of Rubber Duck Debugging in the Age of AI

Discover why rubber duck debugging is a powerful process today. There's also a step-by-step guide on how to use it in the age of artificial intelligence.

Read more on page.....66

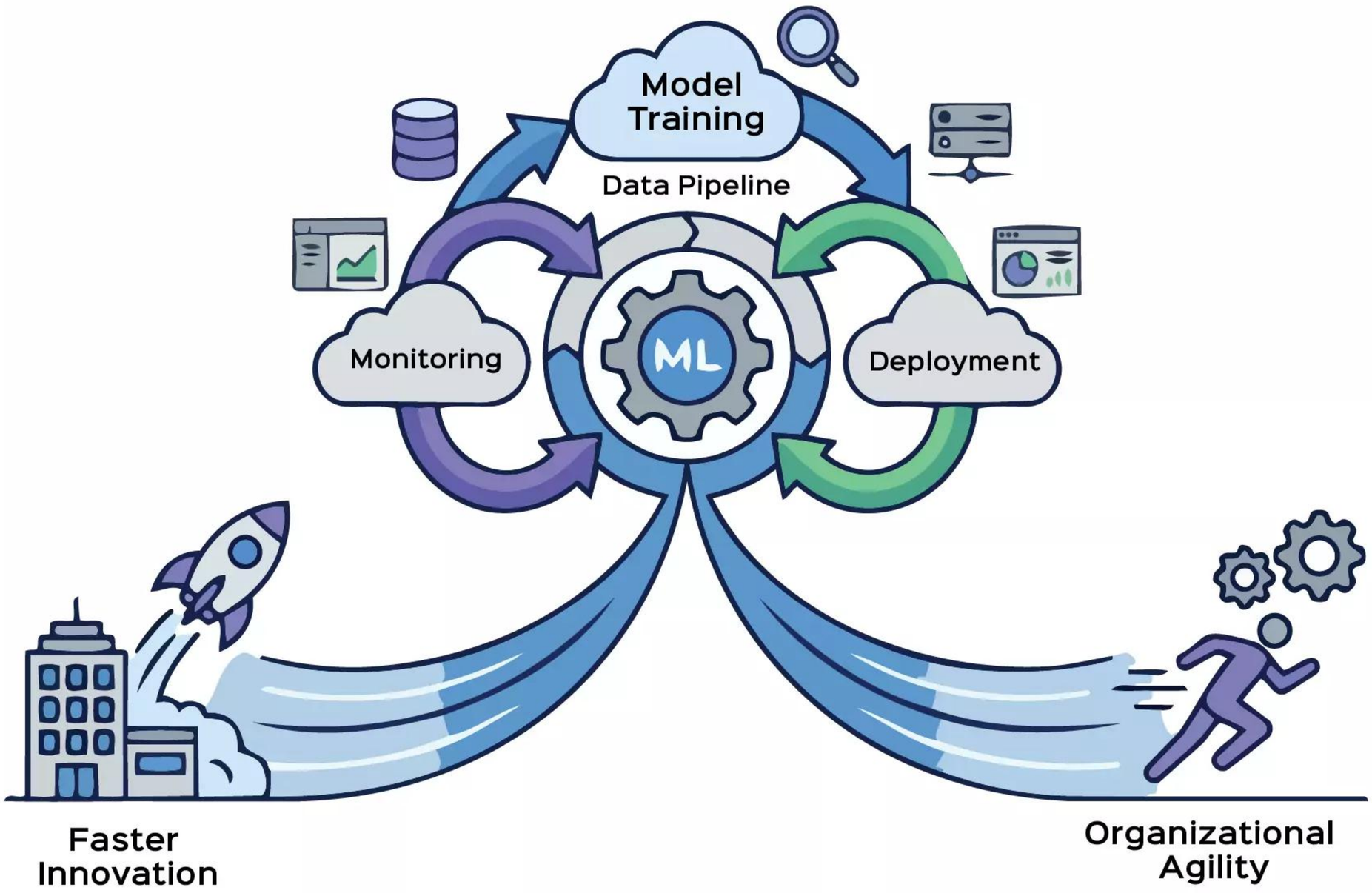
Why Open Source Large Language Models are Popular

Open source large language models mark a pivotal moment in the evolution of generative AI. By lowering barriers to entry and fostering collaborative innovation, these models are enabling a broader spectrum of organisations to benefit from AI.

Read more on page.....68

MLOps for Enterprise Agility: An Overview

MLOps helps enterprises develop, deploy, and manage machine learning systems that are reliable, scalable, and production ready.



Enterprises have moved past the era of siloed AI experiments and into a phase of true operational reliance. The goal is to automate and stabilise the entire lifecycle of a machine learning model and leverage autonomous agents that drive revenue, safety, and customer trust.

According to a recent study from *Innovatics*, nearly 87% of machine learning models fail during deployment. They never reach production. Manual monitoring leads to undetected model drift and risk. Also, only 54% of AI projects advance from pilot to production at best. MLOps tools and technologies address these failures. Enterprises are investing billions trying to solve the ML model deployment crisis.

MLOps is the standardised framework that unifies data science, DevOps, and governance to ensure models are not just built, but are reliable, compliant, and cost-effective at scale. It bridges the gap between the people who build the models and those who run the business. Its pillars are:

Machine learning: Focused on model architecture, feature engineering, and hyperparameter tuning.

DevOps*:* Focused on CI/CD (continuous integration/continuous deployment), infrastructure-as-code, and system uptime.

Data engineering: This focuses on the 'plumbing' — it addresses ingesting, cleaning, and versioning the massive datasets required to train models.

According to *Grant View Research*, the global MLOps market is projected to grow from US$ 2.19 billion (2024) to US$ 16.6 billion by 2030 at a CAGR of 40.5%. As per a report from *Innovatics*, MLOps spending in the US grew from nearly zero to over US$ 2 billion in 2024, and is projected to reach US$ 17-40 billion by 2030.

Core principles of MLOps

MLOps enables automation across the ML lifecycle, reducing errors and accelerating delivery, while aligning data scientists, ML engineers, data engineers, and DevOps teams around shared processes and tools.

Four continuous practices form the backbone of MLOps.

Continuous integration (CI) in MLOps*:* This automates the merging

of code and models into a shared repository while maintaining strict version control. CI covers:

- *Automated code and model integration:* Merges code changes and ML models into a shared repository seamlessly.
- *Version control for code, data and models:* Utilises tools like Git, DVC or MLflow to track changes.
- *Automated testing of ML pipelines*: Implements unit tests, integration tests and validation tests for data and models.
- *Build reproducibility:* Ensures that builds can be replicated consistently across environments.
- *Enhanced collaboration*: Facilitates teamwork between data scientists, engineers and stakeholders.

Continuous deployment (CD) in MLOps: This moves models safely into production using strategies like blue-green or canary deployments to minimise risk. CD covers:

- *Automated model deployment:* Deploys machine learning models to production environments automatically.
- *Infrastructure as Code (IaC):* Uses Terraform or Kubernetes for scalable, reproducible deployments.
- *Blue-green and canary deployments:* Implements strategies for safe and incremental model releases.
- *Rollback mechanisms:* Enables quick rollback to previous model versions in case of issues.
- *Seamless integration with CI pipelines:* Ensures smooth transition from integration to deployment phases.

Continuous training (CT) in MLOps: This regularly updates models with fresh data to maintain their predictive power automatically. It covers:

- *Automated retraining of pipelines:* Schedules and triggers model retraining based on data updates or performance metrics.
- *Data versioning and management:* Tracks and manages datasets using tools like DVC or Delta Lake.
- *Hyperparameter tuning and experimentation:* Automates optimal parameter search with frameworks like Optuna or Hyperopt.
- *Scalable training infrastructure:* Leverages cloud resources or distributed computing for efficient training.
- *Model validation and testing:* Ensures new models meet performance and quality standards before deployment.

Continuous monitoring (CM) in MLOps: This tracks real-time metrics such as accuracy and latency using tools to identify issues immediately. It covers:

- *Real-time model performance monitoring:* Tracks metrics using tools like Prometheus or Grafana.
- *Data and model drift detection:* Identifies changes in data distribution or model performance.
- *Automated alerts and notifications:* Sets up alerts for anomalies or performance degradation.
- *Logging and auditing:* Maintains logs for compliance and troubleshooting.
- *Feedback loops for model improvement:* Uses monitoring insights for retraining and deployment cycles.

Table 1: DevOps and MLOps – a comparison

Parameter	DevOps	MLOps
Purpose	General software development and IT operations	Machine learning model development, deployment and monitoring
Collaboration	Developers, IT operations and security teams	Data scientists, ML engineers, DevOps engineers and analysts
Roles	DevOps engineers, system admins, software developers	MLOps engineers, data scientists, machine learning engineers
Key tools	Jenkins, Docker, Kubernetes, Ansible, Terraform	ML flow, DVC, Kubeflow, TFX, Airflow
Main component	Code	Code + Data + Model
Testing	Unit and integration tests	Back testing, drift detection, and bias audits
Lifecycle management	Focuses on app code lifecycle and infrastructure	Focuses on data, model lifecycle and continuous retraining
Primary focus	Code delivery, system automation and reliability	Data versioning, model reproducibility and performance
Deployment	Single software version	A/B testing and shadow deployments
Maintenance	Bug fixes	Retraining and fine-tuning
Performance monitoring	Primarily monitors system and application performance	Monitors model drift, accuracy and impact on real world data
Scalability	Infrastructure scaling for application needs	Scaling ML pipelines and model deployment

MLOps lifecycle

The process of building and maintaining an ML model is divided into four key functional areas.

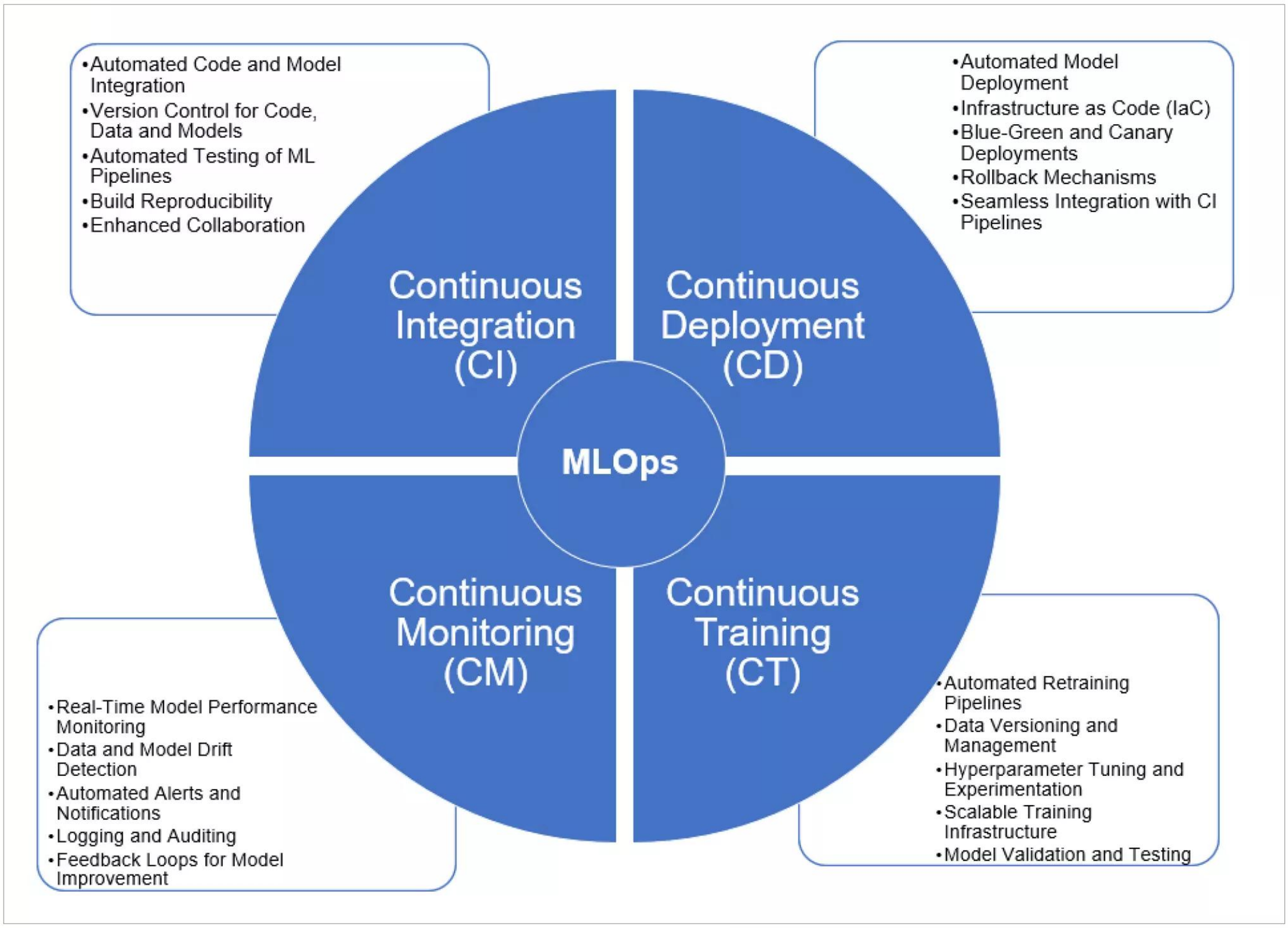


Figure 1: Principles of MLOps

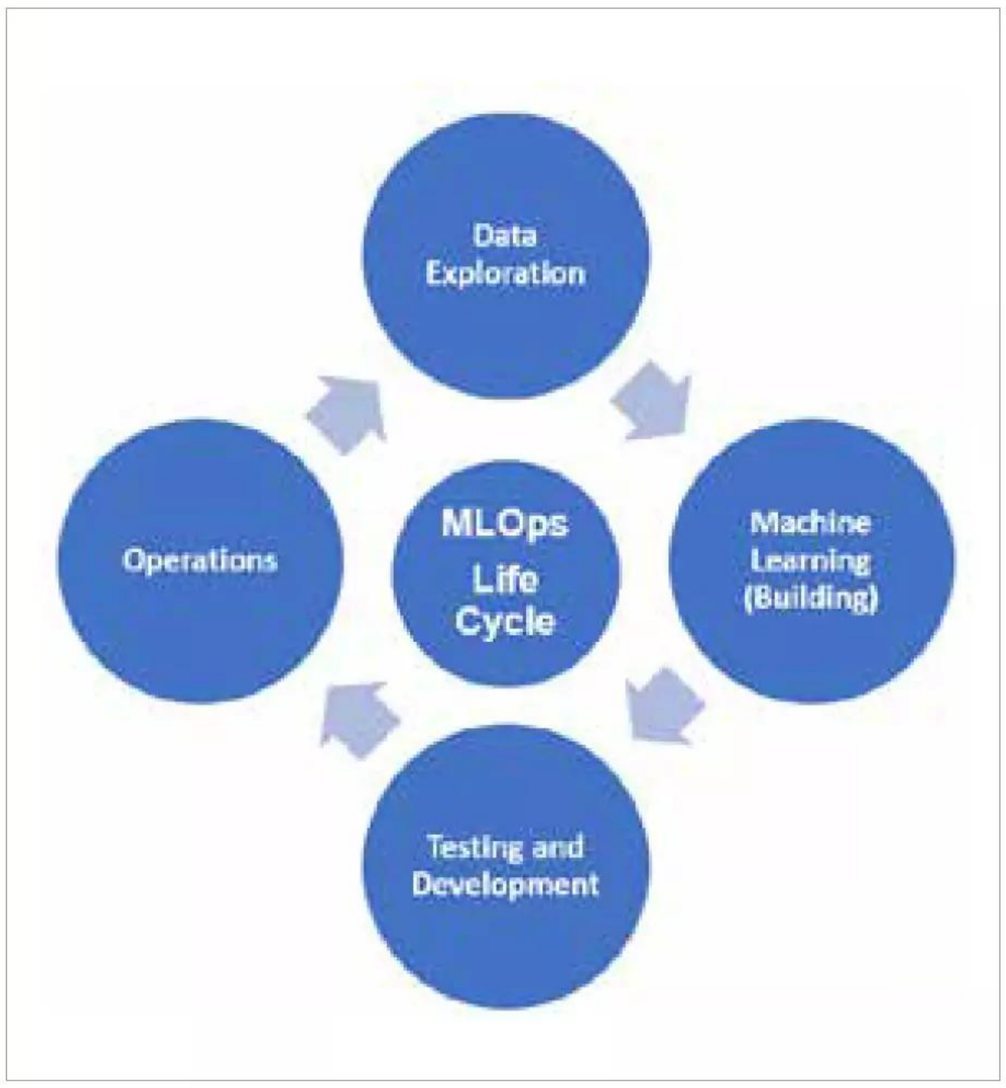


Figure 2: MLOps lifecycle

Data exploration: This initial phase occupies 30-40% of the project time. It involves collecting, transforming, and validating data to ensure it is ready for training.

Machine learning (building): This involves selecting the right algorithms and optimising model performance for the specific business use case.

Testing and development: Rigorous validation is done in this phase using techniques like cross-validation to ensure the model solves the problem without creating new ones.

Operations: This involves deploying the model as a service and monitoring its performance in a production environment.

MLOps framework for enterprise adoption

The MLOps framework is designed to bridge the gap between model development and production. It focuses on automation, monitoring, and standardising the ML lifecycle. The steps involved are:

Business case: In this step, the usage of ML and how it should be leveraged to solve the business problem is clearly articulated.

Data collection*:* Raw data is identified and gathered from all relevant sources as part of this step – for example, EHR data, lab results,

prior admissions, demographics, claims, etc.

Data ingestion: In this step, data is moved from source systems into ML platforms like data lakes, warehouses, or feature stores. An example is batch ingestion of nightly ETL jobs from operational databases into a lakehouse.

Data transformation and cleaning: Here, raw and messy data is turned into structured, usable data (such as removing duplicates of the same patient visits recorded twice).

Data analysis: This involves exploratory data analysis (EDA) to understand distributions, relationships, and issues. An example is a correlation analysis between features and the target.

Data visualisation: Visual storytelling makes data patterns and issues understandable to stakeholders – for example, visualising customer churn by tenure and product usage.

Feature engineering: In this step, meaningful input variables (features) are created from raw data. An example is the number of patient admissions in a hospital in the last 12 months.

Model exploration: This involves trying different model families and approaches to see what fits best – for example, LSTMs for time-series and Transformers for text in deep learning.

Model building: The chosen model(s) are formalised into reproducible code and pipelines, such as implementing a training script that reads from the feature store, trains, and logs metrics.

Model training: The training process is run at scale, often multiple times, in this step. An example is training on millions of patient records to predict readmission.

Model evaluation: This assesses how well the model performs and whether it's fit for its purpose – for example, fairness and performance across age, gender, region, or other sensitive groups.

Model deployment: The model is made available to real users and systems in this step. An example is a REST endpoint that returns readmission risk when given patient data in API deployment.

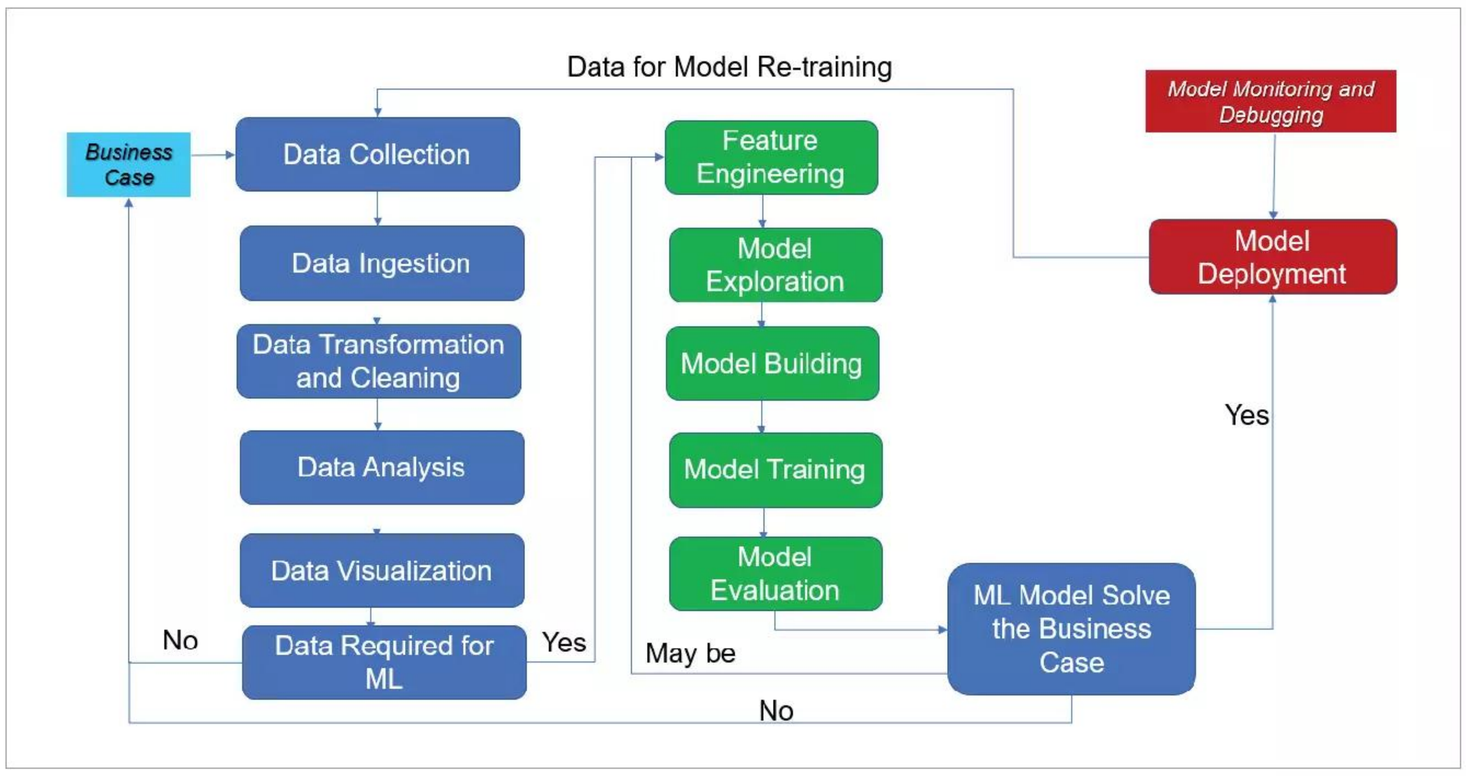


Figure 3: MLOps framework

***Model monitoring and debugging*:** This involves ongoing oversight of the model in production such as monitoring prediction drift.

Role of an MLOps engineer

MLOps engineers acts as the bridge between development and operations. Their roles focus on:

Foundational DevOps skills: Managing the operation and deployment aspects of the ML workflow using traditional software engineering practices.

Standardised model development: Implementing industry best practices to ensure models are reproducible, scalable, and compliant with organisational standards.

Pipeline management: Designing and managing end-to-end pipelines from data ingestion to automated model deployment, so that data scientists can focus on higher-level tasks.

Open source MLOps platforms

A modern MLOps stack combines traditional DevOps tools with specialised machine learning platforms (Table 2).

Here's a brief description of these tools.

Apache NiFi: This visual data-flow tool is designed for moving and transforming data between systems. It supports drag-and-drop pipelines, real-time streaming, and strong data provenance. NiFi excels in regulated environments because every data movement is tracked and auditable.

MLOps framework steps	Tools used
Data ingestion	Apache NiFi, Airbyte, Slinger, Kafka
Orchestration	Apache Airflow, Kubeflow Pipelines
Data processing	Apache Spark, dbt Core, Great Expectations
Analytics and exploration	JupyterLab, Apache Superset
Model development	PyTorch, TensorFlow, Scikit-Learn
CI/CD and packaging	Jenkins, Docker
Model registry	MLflow Model Registry
Observability	Prometheus + Grafana, ELK Stack
Feedback loop	Kafka

Table 2: Open source tools used in the different steps of the MLOps framework

Airbyte: Airbyte provides open source ELT connectors for databases, SaaS apps, and APIs. It offers 300+ ready-to-use connectors and a simple YAML-based configuration model. Its modular design makes it easy to extend and integrate into modern data platforms.

Singer (Taps & Targets): This tool defines a simple, open standard for data extraction and loading. Taps pull data from sources, and Targets load it into destinations. It's lightweight, scriptable, and ideal for custom or niche integrations.

Apache Kafka: Kafka is a distributed event-streaming platform used for real-time data ingestion and messaging. It provides durable logs, high throughput, and replayable streams. It's foundational for event-driven architectures and ML feedback loops.

Apache Airflow: This workflow management tool is used to schedule complex data pipelines from ingestion to training. Airflow orchestrates complex workflows using Python-based DAGs. It handles scheduling, retries, dependencies, and lineage and is widely used for ETL, ML pipelines, and cross-system automation.

Apache Spark: This distributed compute engine for large-scale data processing supports SQL, machine learning, streaming, and graph analytics. It's in-memory execution makes it significantly faster than traditional MapReduce.

dbt Core: This tool enables SQL-based data transformations with software-engineering best practices. It brings versioning, testing, documentation, and lineage to analytics pipelines. It's ideal for building curated datasets and ML-ready tables.

Great Expectations: Great Expectations is a data quality and validation framework. It lets teams define 'expectations' for data and automatically test them. It generates human-readable reports and integrates seamlessly with pipelines.

JupyterLab: This is an interactive development environment for notebooks, code, and data exploration. It supports multiple kernels (Python, R, Julia) and rich visualisations. It's the default workspace for data scientists.

Apache Superset: Superset is an open source BI and dashboarding platform. It offers SQL exploration, charting, and interactive dashboards. It's lightweight, cloud-friendly, and ideal for analytics teams.

Jenkins: This widely used open source CI/CD automation server supports pipelines-as-code, plugins, and integrations with any tool. It's often used to automate model packaging, testing, and deployment.

Docker: Docker packages ML code, dependencies, and models into portable containers. It ensures reproducibility across dev, test, and production. It's essential for Kubernetes-based ML platforms.

Kubeflow Pipelines: This orchestrates end-to-end ML workflows on Kubernetes. It provides reusable components, metadata tracking, and experiment management, and is ideal for scalable, cloud-native ML training.

PyTorch/TensorFlow/scikit-learn: These are the core open source ML frameworks.

- *scikit-learn:* Classical ML, fast prototyping
- *PyTorch:* Flexible deep learning, research-friendly
- *TensorFlow:* Production-grade deep learning with strong ecosystem support

MLflow Model Registry: Model Registry stores, versions, and manages ML models. It supports lifecycle stages (staging → production), lineage, and approvals. It integrates with MLflow tracking and serving tools.

Seldon Core: Seldon Core deploys and serves ML models on Kubernetes. It supports REST/gRPC endpoints, canary releases, A/B tests, and explainers. It's highly scalable and production ready.

Prometheus + Grafana: Used for monitoring, these visualise performance metrics and identify anomalies. Prometheus collects time-series metrics, and Grafana visualises them. Together, they provide observability for ML services (latency, throughput, errors).

ELK Stack (Elasticsearch, Logstash, Kibana): The ELK stack handles centralised logging and search. Logstash ingests logs, Elasticsearch indexes them, and Kibana visualises them. ELK is widely used for debugging ML pipelines and services.

Kafka (Feedback Loop): Kafka captures predictions, outcomes, and user interactions for retraining. It enables event-driven ML and continuous learning.

LLMOps: The future of MLOps

Generative AI models have become mainstream in enterprise development. Organisations are now shifting toward LLMOps, or large language model operations, to manage autonomous agentic workflows and enforce safety and governance. LLMOps spans prompt management, vector-store orchestration, fine-tuning pipelines, predictive modelling, integrated safety layers, and continuous evaluation, enabling reliable, governed, and cost-efficient scaling of genAI systems. This evolution requires traditional MLOps teams to expand their skills and operate as LLMOps practitioners to ensure generative outputs remain secure, aligned, and trustworthy.

The core capabilities of LLMOps are:

Prompt management and versioning: Track, test, and optimise prompts as first-class artifacts.

RAG (retrieval-augmented generation) operations: Manage vector databases, embeddings, chunking strategies, and retrieval pipelines.

Model lifecycle management: Handle fine-tuning, model selection, model switching, and rollback.

Evaluation and benchmarking: Continuous testing for hallucinations, toxicity, bias, and factual accuracy.

Observability and monitoring: Track latency, cost, token usage, drift, and user feedback loops.

Safety, guardrails and policy enforcement: Apply content filters, grounding, red-teaming, and enterprise-grade safety layers.

Cost optimisation: Manage model routing, caching, batching, and hybrid model strategies.

Some of the open source LLMOps tools/ frameworks are: LangChain/LangGraph, LlamaIndex, Haystack, Agenta, Langfuse, Chroma, Milvus/ Weaviate, and Arize Phoenix.

MLOps has progressed from 'model operations' to 'intelligence operations', supporting AI systems that learn continuously, act autonomously, and operate safely at enterprise scale. CIOs need to treat MLOps as a core enterprise capability, not a data science accessory. MLOps tools and frameworks need to become the norm, especially for enterprises with sensitive data. They must invest in cross-skilling data engineers, ML engineers, and DevOps teams.

As generative AI models become mainstream, the future seems to be the emergence of LLMOps. However, managing LLMs introduces new challenges like prompt management, fine-tuning pipelines, vector stores, and safety layers. MLOps teams must evolve into LLMOps teams, integrating agentic workflows and safety governance. END

Acknowledgements

The authors would like to thank Tricon Solutions LLC and Gspann Technologies, Inc., for giving the required time and support in many ways when this article was being written.

By: Dr Gopala Krishna Behara and Narumanchi Sree Keerthi

Dr Gopala Krishna Behara is an enterprise architect at Tricon IT Solutions. He has around 28 years of experience in the IT industry.

Narumanchi Sree Keerthi is a senior software engineer at Gspann Technologies, Inc. She has around 5 years of experience in the IT industry.

Disclaimer: *The views expressed in this article are those of the authors. Tricon Solutions LLC and Gspann Technologies, Inc., do not subscribe to the substance, veracity or truthfulness of the said opinion.*

An "AWESOME" Development Board from India?

- **Super Small:** 3 centimeters
- **Super Smart:** 3 processors
  (Xtensa® Microprocessor + 2 ULP RISC-V Coprocessor)
- **Super Sensitive:** 5 Sensors
- **Super Connected:** Wifi & 10+ Interfaces
- **Super Flexible:** 30 GPIOs
- **Super Easy:** FPC + Dedicated Battery Connector

Introducing the

IndusBoard COIN

Priced Shockingly at ₹2,359 (Inclusive of GST)



SPECIAL INTRO PRICE

₹1,749 (Inclusive of GST)

KNOW MORE / BUY NOW: indus.electronicsforu.com

Developed by the Electronics For You team, for you.

DEALERS: • Om Technology Centre, Bengaluru (Mob: +919845400864)
• Bharathi Electronics: Coimbatore (Mob: +919952228618)

Building Production-Ready Retrieval Systems

This comprehensive and practical guide will help you successfully design, build, and run retrieval-augmented generation (RAG) systems that are viable in production environments.



Retrieval-augmented generation (RAG) combines information retrieval with large language model (LLM) capabilities to provide answers that are based on current, validated sources rather than only on previously accessed information. By using more contextual data added to the answer at the time of response, RAG helps produce 'grounded' (i.e., based on evidence), real-time, valid answers.

LLMs are incredibly useful but have many limitations, the most serious one being that they can generate answers based only on their training data. As a result, they are not suitable for providing accurate responses where the answers require current information, knowledge of a specific domain, or proprietary information.

RAG provides the solution to these problems by integrating two separate systems:

- Retriever that retrieves the relevant data from an external data source.
- Generator (LLM) that generates answers based on the data returned from the retrieved context.

Rather than asking the model to retrieve everything, RAG gives it access to retrieve information as needed while processing requests. Therefore, it is particularly useful in the following areas:

- Enterprise knowledge base
- Product documentations and APIs
- Internal tools (i.e., HR, finance, support)
- Regulated industries (i.e., healthcare, legal, finance)

Most RAG examples found online are demo-grade systems. They work well for tutorials but fail under real-world constraints.

Table 1: The difference between demo RAG and production RAG systems

Dimension	Demo RAG	Production RAG
Data size	Few documents	Millions of chunks
Updates	Manual, static	Continuous ingestion
Retrieval	Simple vector search	Hybrid + re-ranking
Latency	Ignored	Static SLAs
Evaluation	Manual inspections	Automated metrics
Reliability	Best effort	Fault-tolerant

RAG architecture

To create a production-level RAG system, we use a structured pipeline design that separates the knowledge retrieval process from the language generation process. The operating flow of the RAG system has a defined path to follow during runtime. First, the system analyses and rewrites a user query in order to improve the effectiveness of its retrieval process. Next, the system uses a 'retriever' to find matching vector, or hybrid, database entries for the query, returning a set of candidate text 'chunks' (potential pieces of text).

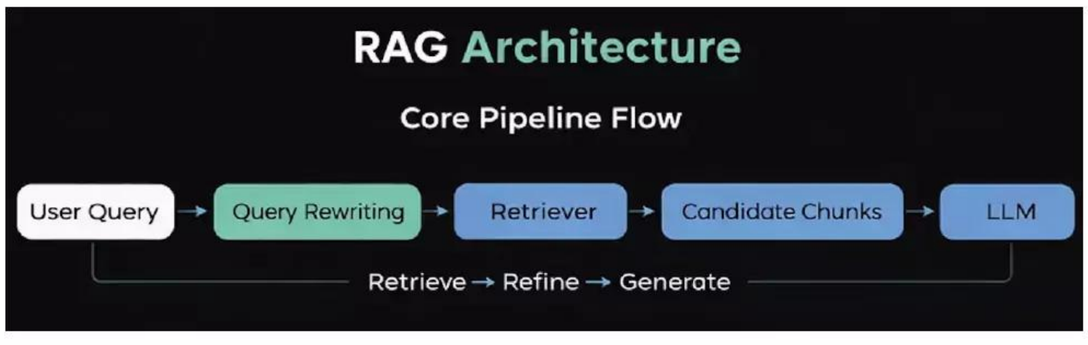


Figure 1: RAG architecture

Finally, the retrieved chunks are refined and assembled into a bounded context which can be provided to the LLM

as a basis for generating the final response. This process of retrieve-refine-generate, as shown in the architectural flow diagram (Figure 1), provides assurance that the LLM's answer is built on externally verified source material rather than solely on its internally trained data.

There are two pipelines in RAG architecture: offline and online. Offline pipelines create knowledge before the user sends queries by reading source documents, cleaning and standardising text, cutting content into pieces with clear meanings, adding more information about the item, and creating and storing embeddings of the item in a vector database. This type of processing is done with an emphasis on the rate of processing (throughput), not the speed at which it is processed. Therefore, offline processing can happen using a batch.

Online pipelines respond to the request of an end user in real-time and have strict guidelines regarding latency. The online pipeline includes query rewriting, retrieval of related items, re-ranking the retrieved items, constructing the context, and running inference with an LLM.

By separating the offline and online pipelines, the overall system can scale without affecting the performance of real-time queries due to large and costly indexing.

How does a production-grade RAG architecture work?

Production-ready RAG architectures are heavily influenced by the latency and cost constraints presented on them. As an example, every stage of your online pipeline contributes to your overall latency budget, and LLM inference is typically the most latency- and cost-contributing stage. Given that LLM inference is often the most expensive stage in your production-ready RAG pipeline, retrieval and re-ranking must both be fast and efficient, ensuring that a limited number of high-quality chunks are fed into your model.

In addition, from a cost perspective, architecture must be designed such that system resources are progressively used. This means that retrieval operations (which are typically low-cost) must be performed prior to higher-cost operations like re-ranking and LLM generation, in order to identify the context needed to perform them. This layer-oriented design approach ensures predictable performance, controlled token usage, and the ability to scale as both data volume and query traffic increase.

Data ingestion: Preparing knowledge for retrieval

The process of data ingestion is a crucial element of a production-ready RAG system — it changes raw enterprise data into knowledge that can be retrieved. In the production environment, ingestion is an ongoing process whereby an automated pipeline is created to maintain data quality, currency, and traceability.

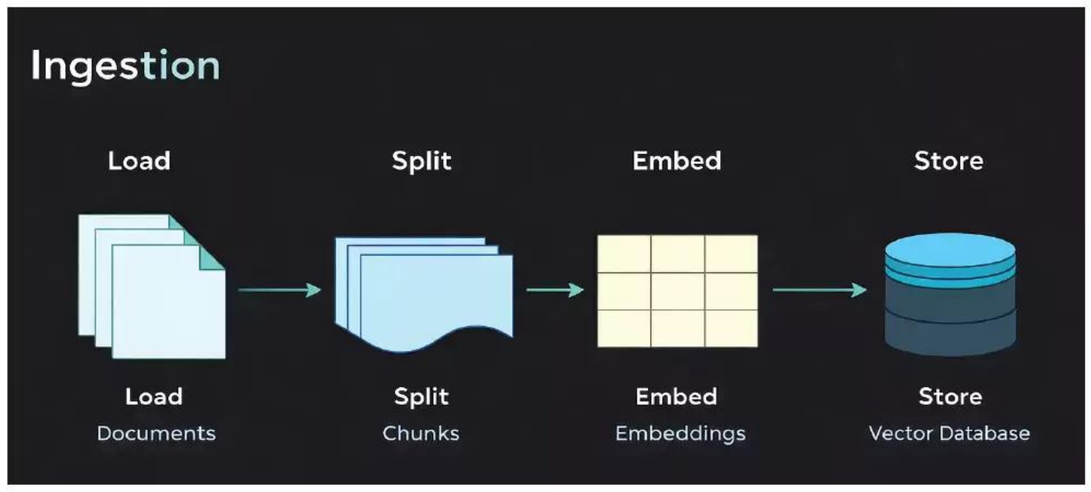


Figure 2: Data ingestion

Step 1. Source types (Load): RAG systems can ingest data from several different sources (e.g., PDFs, HTML pages, Markdown files, databases, APIs, knowledge bases, and other internal systems). Each type of source requires a specific type of loader to extract the content properly and maintain the document boundaries, timestamps, and access control, all of which are necessary for secure and auditable retrieval.

Step 2. Cleaning and normalisation (Split): Once loaded into the RAG system, documents undergo a cleaning process to remove noise such as headers, footers, navigation elements, and duplicate content. During this process, the text is normalised (ensuring that all the text uses the same encoding and formatting) and documents are split into semantically meaningful size chunks. This ensures that low-quality text is not embedded in the RAG system, which would then degrade the precision of subsequent retrievals.

Step 3. Metadata enrichment (Embed): Each chunk of data is enriched with structured metadata (i.e., source, document type, language, version, and date of update). The metadata enables filtered, hybrid and context-enabled retrievals in production systems. The chunks of data are then processed to generate dense vector embeddings of the data. The processing of the chunks is typically performed using a batched approach, which allows for lower cost and higher throughput.

Step 4. Update handling (Store): Storing all embeddings and their associated metadata occurs through a vector database. Incremental updates occur in production environments by only re-embedding any new or changed documents for production use and by deprecating old content with versioning and soft deletes. By using these processes for deprecation of old or outdated content, the accuracy of retrieval from the vector database will not require the entire corpus to be reprocessed.

Chunking and the types of chunking strategies

The process of breaking down raw documents into smaller units of knowledge for use in embedding and retrieval processes is called chunking. The design of the chunks directly impacts the retrieval accuracy, context relevance, and ability to control hallucination in RAG systems. Poorly

designed chunks may yield insufficient or noisy context sent back by the retriever (due to either the design of the chunks or poor quality of the embeddings), resulting in the LLM generating incomplete or inaccurate answers, even when it has high-quality embeddings as a source.

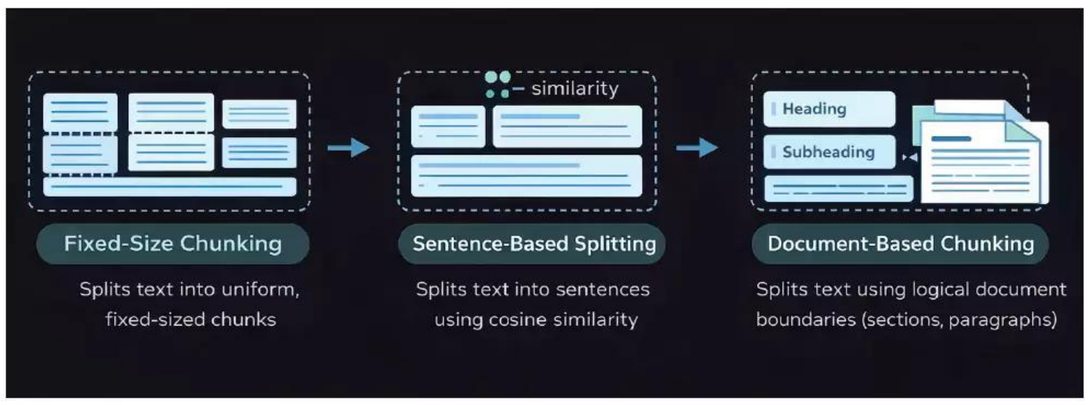


Figure 3: Chunking

Chunks that are too small lose any connected semantics and relationships, which may affect performance, whereas chunks that are too large introduce irrelevant context and waste tokens. A well-designed chunking strategy will allow each chunk to be retrievable, meaningful, and self-containing.

Fixed size chunking: The document is broken up into chunks that are characterised by the presence of tokens (single units of language), for example, in fixed size chunks. The advantage of this method is speed and ease of implementation, but the disadvantage is that it often breaks apart logical boundaries in a way that creates discontinuities in the represented data.

```
from typing import List
import re

# Split the text into units (words, in this case)
def word_splitter(source_text: str) -> List[str]:
   source_text = re.sub("\s+", " ", source_text)  # Replace
multiple whitespces
   return re.split("\s", source_text)  # Split by single
whitespace

def get_chunks_fixed_size_with_overlap(text: str, chunk_size:
int, overlap_fraction: float = 0.2) -> List[str]:
   text_words = word_splitter(text)
   overlap_int = int(chunk_size * overlap_fraction)
   chunks = []
   for i in range(0, len(text_words), chunk_size):
       chunk_words = text_words[max(i - overlap_int, 0): i +
chunk_size]
       chunk = " ".join(chunk_words)
       chunks.append(chunk)
   return chunks
```

Sentence-based chunking: The sentences in the document are used as the points at which to divide it up into chunks. The option of merging chunks into single ones based upon their similarity may be considered as well when implementing this strategy. This method provides better structural support than the previous method of chunking at fixed sizes; however, it can still be impeded by chunks created from conceptually multi-sentence pieces or lengthy explanations.

```
from typing import List
def recursive_chunking(text: str, max_chunk_size: int = 1000)
-> List[str]
   # Base case: if text is small enough, return as single
chunk
   if len(text) <= max_chunk_size:
       return [text.strip()] if text.strip() else []
     # Try separators in priority order
   separators = ["\n\n", "\n", ". ", " "]

   for separator in separators:
       if separator in text:
           parts = text.split(separator)
           chunks = []
           current_chunk = ""

           for part in parts:
               # Check if adding this part would exceed the
limit
               test_chunk = current_chunk + separator + part
if current_chunk else part

               if len(test_chunk) <= max_chunk_size:
                   current_chunk = test_chunk
               else:
                   # Save current chunk and start new one
                   if current_chunk:
                       chunks.append(current_chunk.strip())
                   current_chunk = part

           # Add the final chunk
           if current_chunk:
               chunks.append(current_chunk.strip())

           # Recursively process any chunks that are still
too large
           final_chunks = []
           for chunk in chunks:
               if len(chunk) > max_chunk_size:
                   final_chunks.extend(recursive_
chunking(chunk, max_chunk_size))
               else:
                   final_chunks.append(chunk)

           return [chunk for chunk in final_chunks if chunk]
```

```
    # Fallback: split by character limit if no separators work
    return [text[i:i + max_chunk_size] for i in range(0,
len(text), max_chunk_size)]
```

Document-based chunking: Chunks are created from chunks based upon logical document boundaries (headings, subheadings, paragraphs, and sections), allowing the content's intended meaning as well as its structure to be retained, making this option the optimal method of chunking for any production RAG systems, particularly for technical, legal, or knowledge-based documents.

```
from typing import List
import re
def markdown_document_chunking(text: str) -> List[str]:
    # Split by markdown headers (# ## ### etc.)
    header_pattern = r'^#{1,6}\s+.+$'
    lines = text.split('\n')

    chunks = []
    current_chunk = []

    for line in lines:
        # Check if this line is a header
        if re.match(header_pattern, line, re.MULTILINE):
            # Save previous chunk if it has content
            if current_chunk:
                chunk_text = '\n'.join(current_chunk).strip()
                if chunk_text:
                    chunks.append(chunk_text)

            # Start new chunk with this header
            current_chunk = [line]
        else:
            # Add line to current chunk
            current_chunk.append(line)

    # Add final chunk
    if current_chunk:
        chunk_text = '\n'.join(current_chunk).strip()
        if chunk_text:
            chunks.append(chunk_text)

    return chunks
```

Embeddings and vector storage power semantic retrieval

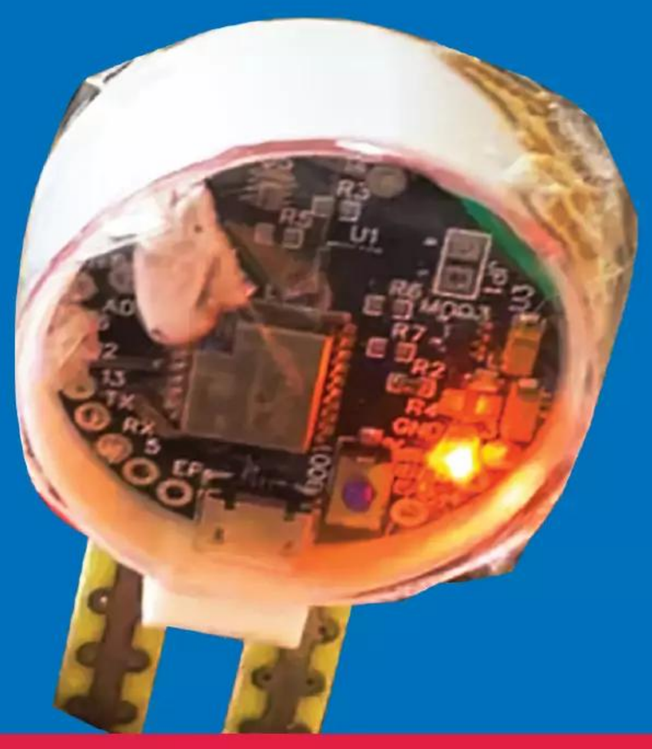
Embeddings take content (such as text and audio data) and convert it into numeric arrays that represent the meaning of the data. Figure 4 illustrates that semantically related sentences create embeddings that are closer together than disparate sentences. Once you select your embedding model,

DIY KIT: Agriculture IoT & AI

Want to create exciting IoT projects for agriculture use cases or agri-IoT projects?

This kit provides everything you need to create typical IoT for agriculture (aka agri IoT) projects.

KIT INCLUDES:
- Soil Sensor
- GSM Module
- Relay Module
- *IndusBoard Coin*



POWERED BY



IndusBoard COIN

Super small sized development board that enables you to create solutions for many use-cases where small size is super important.

SPECIAL INTRO PRICE

₹2,149 (Inclusive of GST)

KNOW MORE / BUY NOW: efy.co.in/agriiot

Developed by the Electronics For You team, for you.

DEALERS: • Om Technology Centre, Bengaluru (Mob: +919845400864)
• Bharathi Electronics: Coimbatore (Mob: +919952228618)

you will be balancing between quality, latency, and cost (the larger your model, the more semantic information will be created, whereas the smaller the model, the more likely that it will meet your specific case). Within a production system, embeddings should be normalised and batched together, and to further boost retrieval, an additional layer of multilingual embeddings can support cross-lingual retrieval within the same pipeline.

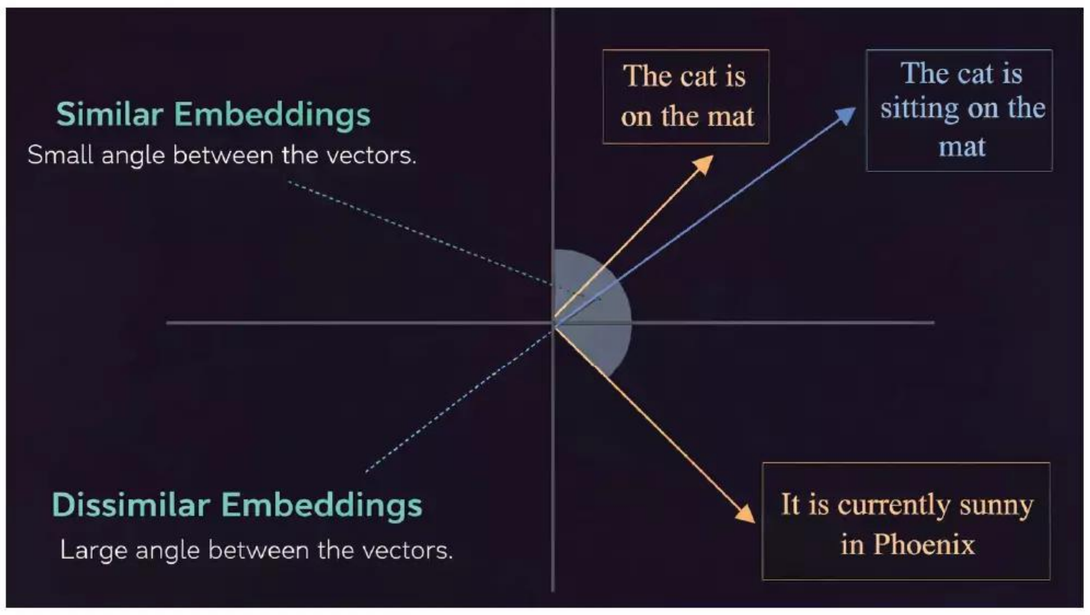


Figure 4: Embeddings

The embeddings are stored in a vector database that has been designed to operate using similarity search techniques. Because of this, there are several different index types available (some index types trade off speed for accuracy and others scale differently). Metadata filters also exist that can be used to filter through data based on the source, language, and section. Also, hybrid forms of retrieval exist that combine both the results from the similarity search and either keyword-based filtering or rule-based filtering. This will also provide a more precise and dependable answer in retrieval-augmented generators.

The way retrieval logic and context construction work

The query is embedded in a vector database for similarity-based retrieval so that there can be a large number (n = ?) of candidate chunks returned in order to maximise recall. Top-K tuning manages how many chunks are carried forward based upon the trade-offs between relevance, latency, and cost. To ensure there is no redundancy in the chunk candidates, a method called MMR Retrieval, which uses diversity to find relevant chunks, is often applied. Finally, chunks are reordered based on deeper levels of relevance through a re-ranking model (based on query–chunk or query–tag matches), so that the best chunks of context reach the LLM. Query rewriting may also be used to clarify intended meaning and to increase retrieval accuracy.

From the re-ranked results, only the top-K chunks are selected to fit within the LLM (limited tokens). The chunks will be ordered according to relevance and injected into

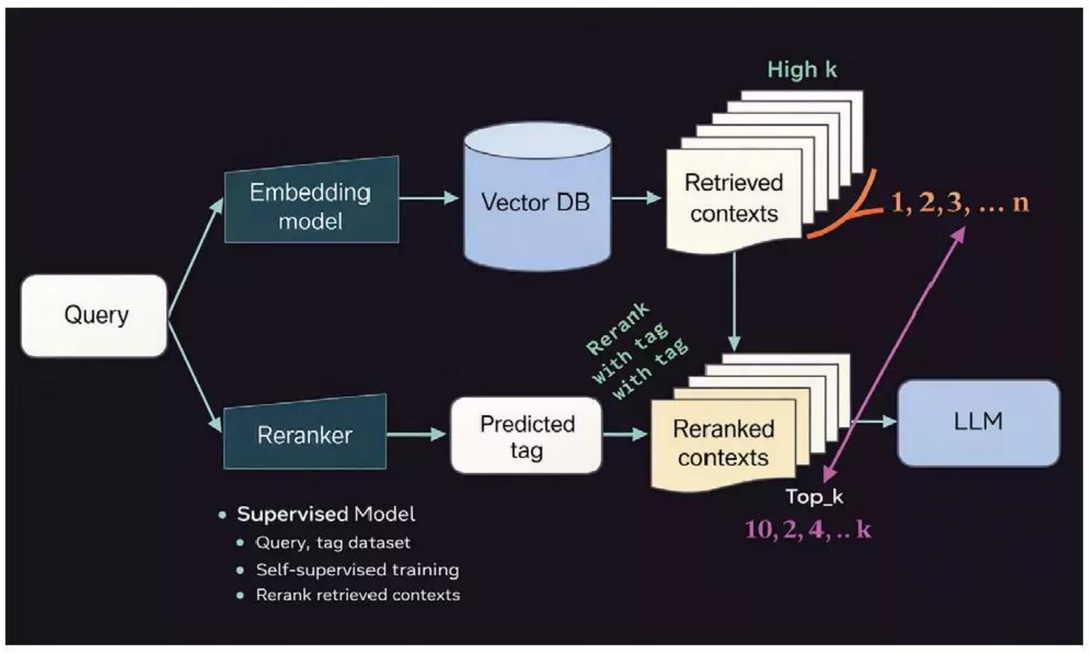


Figure 5: Retrieval logic and context construction

a structured prompt so that the instruction, context, and query are clearly separated. By controlling tight context and relevance filtering, we can minimise hallucinatory data so that responses are based on retrieved data.

Why evaluation and feedback loops are essential for production RAG

RAG evaluation assesses full cycle validation between queries, context retrieved and response generated. Retrieval metrics measure how well the retrieved context matches the query intent, i.e., do the pieces of content retrieved correspond to the intended query? The answer's quality is then measured by how well grounded the answer is, and whether it has relevance to the context from which it is being generated. These two sets of metrics ultimately contribute signals to help continuously improve query rewriting, retrieval ranking and selection of chunks.

Many pitfalls in production create direct breaks to the above-mentioned loop. For instance, over-chunking fragments the meaning and leads to reduced groundedness, whereas neglecting to use metadata results in irrelevant context returned. Sending too much context returned (retrieval inflation) wastes tokens and dilutes relevance, while failure to have fallbacks produces ungrounded answers when no high-quality context exists. To maintain RAG systems that produce reliable, hallucination-resistant outcomes, corrective measures must be taken.

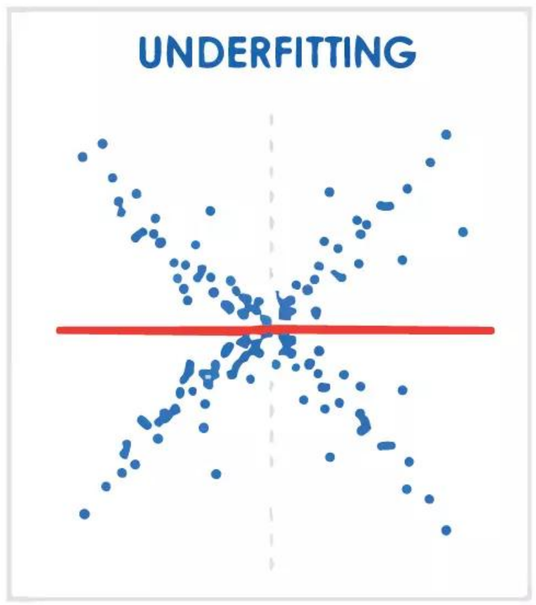
You should now be able to understand the working of an end-to-end production-ready RAG system. Once you have this solid foundation, the building of RAG from the ground upwards will be methodical rather than overwhelming. Using these rules step by step will allow you to successfully transition from an experimental phase to an operationally sustainable, real-world RAG pipeline. END

By: Raj Patel

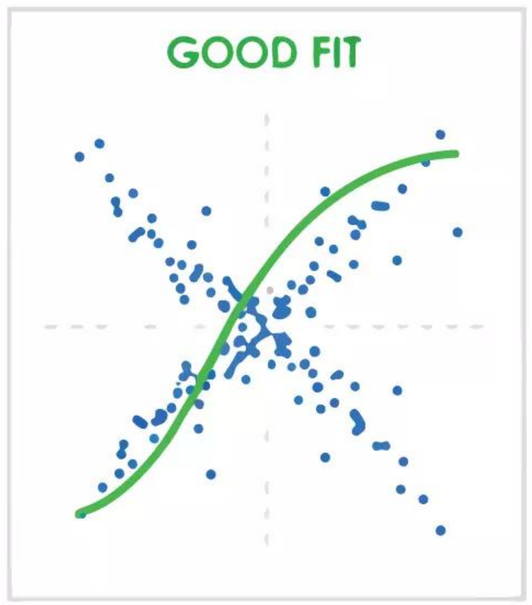
The author is a SaaS enthusiast and technical content creator.

Understanding Underfitting and Overfitting in Machine Learning

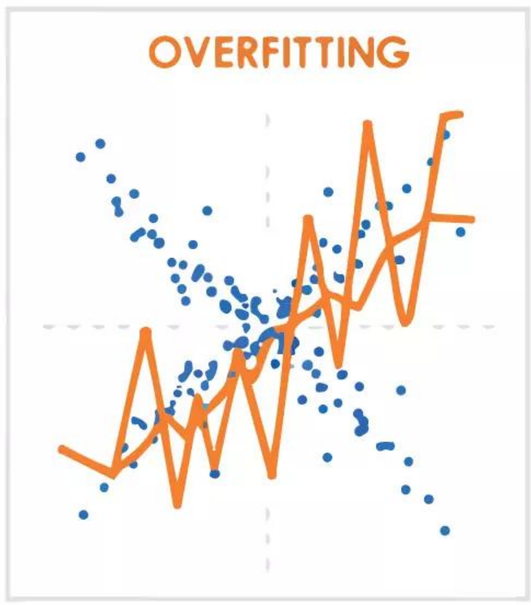
Machine learning models must balance the concepts of underfitting and overfitting.



Model too simple, high error on training and test data



Just right, low error on training and test data



Model too complex, low error on training, high error on test data

In machine learning, a model's goal is not just to fit the given data, but to generalise well to unseen data. Two common failure modes prevent this:

- Underfitting – the model is too simple to capture the true pattern.
- Overfitting – the model is too complex and memorises the training data.

I will demonstrate both concepts using simple, executable Python code, and a dataset generated from a known mathematical formula. This makes it easy to compare predicted values with true values.

Dataset with known ground truth

To clearly understand model behaviour, we generate data using a known formula:

$$y = x^2 + 2x + 1$$

This quadratic relationship allows us to check:

- How well a model fits training data
- How it behaves on unseen values

```
import numpy as np
X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9],
[10]])
y = np.array([x[0]**2 + 2*x[0] + 1 for x in X])
```

Underfitting example: Linear regression

Linear regression assumes a straight-line relationship between input and output. Since our data is quadratic, this assumption is incorrect. Here's the code:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

# Train
model.fit(X, y)

# Predict
print("Underfitting (Linear Regression)")
print("x | predicted | true")
print("--+-----------+------")

print("Seen (training range)")
for v in [1, 3, 5, 7, 9]:
    y_pred = model.predict([[v]])[0]
    y_true = v*v + 2*v + 1
    print(f"{v:2d} | {y_pred:9.2f} | {y_true:4d}")

print("\nUnseen (outside training range)")
```

```
for v in [11, 12, 15]:
    y_pred = model.predict([[v]])[0]
    y_true = v*v + 2*v + 1
    print(f"{v:2d} | {y_pred:9.2f} | {y_true:4d}")
```

The output is:

```
x | predicted | true
--+-----------+------
Seen (training range)
 1 |     -8.00 |    4
 3 |     18.00 |   16
 5 |     44.00 |   36
 7 |     70.00 |   64
 9 |     96.00 |  100

Unseen (outside training range)
11 |    122.00 |  144
12 |    135.00 |  169
15 |    174.00 |  256
```

This is what is observed:

- The model makes noticeable errors even on training data.
- Errors increase smoothly for unseen values.
- The model cannot capture curvature.

This is a classic case of underfitting:

- The model is too simple.
- It fails to learn the true data pattern.

Overfitting example: K-Nearest Neighbors (K = 1)

K-Nearest Neighbors (KNN) makes predictions based on nearby data points. With K = 1, the model simply copies the nearest training value. The code is:

```
from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n_neighbors=1)

# Train
model.fit(X, y)

# Predict
print("Overfitting (KNN(N=1)")
print("x | predicted | true")
print("--+-----------+------")

print("Seen (training range)")
for v in [1, 3, 5, 7, 9]:
    y_pred = model.predict([[v]])[0]
    y_true = v*v + 2*v + 1
    print(f"{v:2d} | {y_pred:9.2f} | {y_true:4d}")

print("\nUnseen (outside training range)")
for v in [11, 12, 15]:
    y_pred = model.predict([[v]])[0]
    y_true = v*v + 2*v + 1
    print(f"{v:2d} | {y_pred:9.2f} | {y_true:4d}")
```

The output is:

```
x | predicted | true
--+-----------+------
Seen (training range)
 1 |      4.00 |    4
 3 |     16.00 |   16
 5 |     36.00 |   36
 7 |     64.00 |   64
 9 |    100.00 |  100

Unseen (outside training range)
11 |    121.00 |  144
12 |    121.00 |  169
15 |    121.00 |  256
```

It can be observed that:

- Predictions are perfect on training data.
- For unseen values, the model repeats the nearest known output.
- It completely fails to follow the quadratic growth.

This demonstrates overfitting:

- The model memorises training data.
- It does not learn the underlying function.

The bias–variance interpretation is:

- Underfitting corresponds to high bias and low variance. The model makes strong assumptions and is consistently wrong.
- Overfitting corresponds to low bias and high variance. The model fits training data perfectly but is unstable on new data.

The goal of machine learning is to find a balance between these two extremes as the table below indicates.

Model	Training error	Test error	Behaviour
Linear regression	High	High	Underfit
KNN (K=1)	Zero	Very High	Overfit
Ideal	Balanced	Low	Good

To conclude, underfitting means not learning enough while overfitting means learning too much. Good models learn the pattern, not the data points. END

By: Supriyo Ganguly

The author is a technical manager at the Electronics Corporation of India Limited, Hyderabad.

Moving Towards New Frontiers with Generative AI

Generative AI is making its mark in diverse fields, but it must be nurtured with care to ensure it's free of bias and preserves human values.



Generative AI is a branch of artificial intelligence focused on producing new data that resembles its training examples. Unlike discriminative models that classify or detect patterns, generative models create original outputs such as text, images, music, or video. By learning the structure of data, these systems can produce realistic and novel content, making them influential across creative industries, research, and automation.

Recent advances in model scale and performance have accelerated interest in generative AI. Large language models can draft articles or hold conversations, image systems generate visuals from text prompts, and audio models synthesize speech or music. This progress marks a shift in AI's role from analytical assistance to creative collaboration.

However, these capabilities also introduce risks. Deepfakes, factual inaccuracies, and biased outputs highlight the need for responsible development and oversight. Hence, understanding generative AI requires both technical knowledge and ethical awareness.

The journey so far

Research on generative models spans several decades in machine learning. Early work relied on probabilistic approaches such as Gaussian mixture models and hidden Markov models, which established the basic principles of modelling data distributions. The deep learning era introduced more powerful frameworks, most notably Generative Adversarial Networks (GANs) by Goodfellow *et al* and Variational Autoencoders (VAEs) by Kingma and Welling, which renewed widespread interest in generative modelling (see References at the end of this article).

Subsequent developments included flow-based models with exact likelihood training and autoregressive approaches like PixelCNN and WaveNet that generated data sequentially. In recent years, diffusion models and Transformer based architectures have achieved major breakthroughs. Diffusion methods now lead in image and audio synthesis, while Transformers originally designed for sequence tasks have become central to large scale language and multimodal generation.

The rise of large foundation models, which can be adapted to many tasks after pre-training, has further unified progress in the field. Prior surveys have compared technical trade-offs and discussed ethical implications such as deepfakes and bias. Building on this literature, our work provides an updated overview that connects recent technical advances with practical applications and ongoing challenges.

Technical foundations of generative AI

Generative AI is built on machine learning models that learn the structure of data and create new samples from it. The four major model families behind modern systems are Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), diffusion models, and Transformer-based models. We briefly describe the intuition behind each approach and highlight their main strengths and limitations. Figure 1 presents a GAN architecture, while Table 1 compares these paradigms.

Generative Adversarial Networks (GANs): Generative Adversarial Networks (GANs) model learning as a competition between two neural networks: a generator and a discriminator. Proposed by Goodfellow *et al*, the generator *G* produces synthetic samples from random noise, while the discriminator *D* attempts to distinguish real data from generated outputs. Both

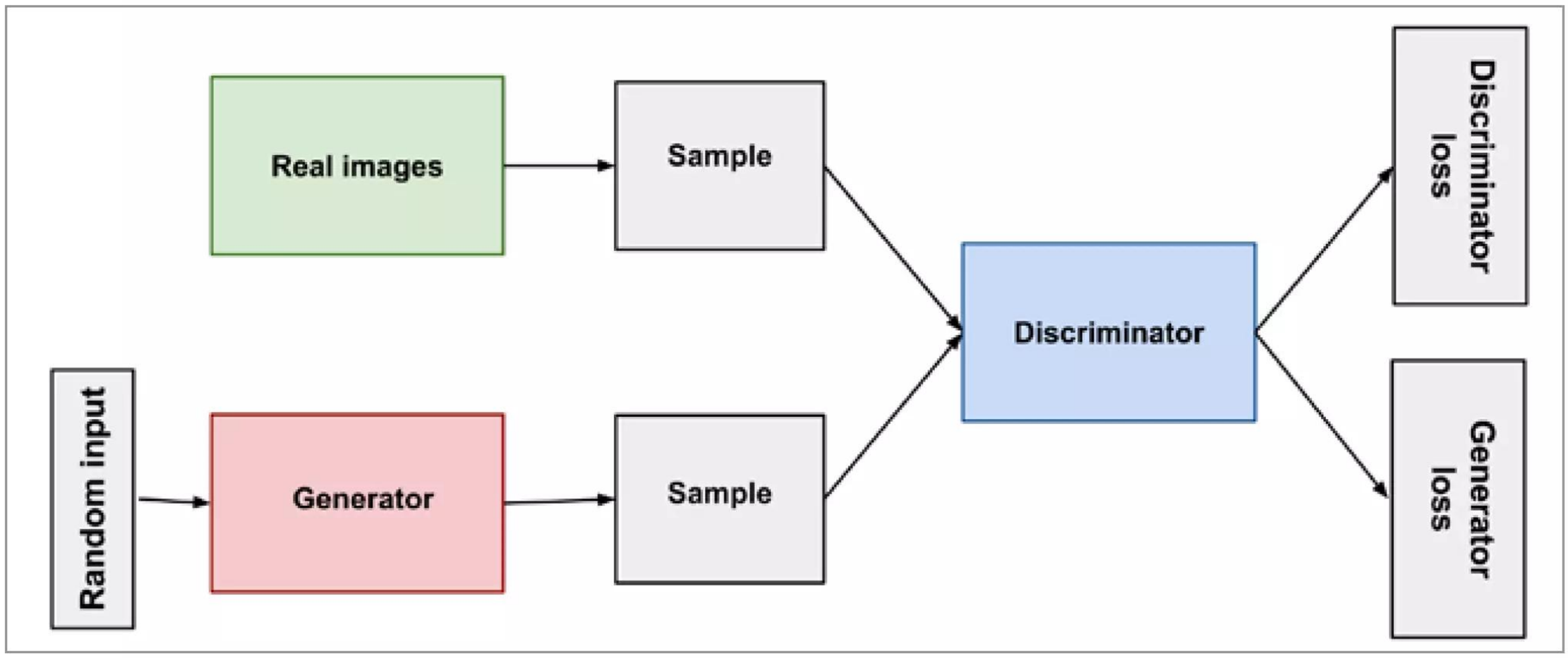


Figure 1: Architecture of GAN

networks are trained together in an adversarial setup, forming a minimax optimisation problem in which *G* tries to fool *D* and *D* learns to detect fakes.

Figure 1 illustrates this interaction, which can be viewed as a two-player game. As training progresses, the generator becomes better at producing realistic data while the discriminator sharpens its detection ability. Successful training allows GANs to approximate the true data distribution, leading to highly realistic outputs. Models such as StyleGAN demonstrated this capability through convincing face and scene synthesis.

GANs are especially known for producing sharp, detailed images, making them effective for tasks like image synthesis and super-resolution. However, they are difficult to train and prone to issues such as mode collapse or unstable convergence. They also do not provide explicit likelihood estimates. Despite these drawbacks, GANs remain widely used in computer vision due to their strong visual quality.

Variational Autoencoders (VAEs): Variational Autoencoders (VAEs) take a probabilistic approach to generative modelling. Introduced by Kingma and Welling in 2013, VAEs extend the traditional autoencoder by representing each input as a distribution in latent space rather than a fixed vector. The encoder outputs parameters of this distribution, usually a Gaussian, while the decoder samples from it to reconstruct data. Training optimises

Model type	Year	Key idea	Strengths	Challenges
Generative Adversarial Network (GAN)	2014 (Goodfellow *et al*)	Uses two neural networks (generator and discriminator) trained in a minimax adversarial setup	Generates high-fidelity, sharp samples; very fast inference	Difficult to train due to instability; suffers from mode collapse; lacks explicit likelihood function
Variational Autoencoder (VAE)	2013 (Kingma & Welling)	Learns probabilistic latent encodings using KL-divergence regularisation	Stable training process; interpretable latent space; enables likelihood estimation	Outputs can appear blurry; requires careful balance between reconstruction loss and regularisation
Diffusion model	2015/2020 (Sohl-Dickstein *et al*; Ho *et al*)	Recovers data from noise by reversing a diffusion process through iterative denoising	Achieves state-of-the-art generation quality; avoids mode collapse	Sampling is slow; computationally expensive; requires many denoising steps
Transformer-based models (e.g., GPT)	2017/2020 (Vaswani *et al*; Brown *et al*)	Uses self-attention mechanisms to learn long-range dependencies in an autoregressive manner	Highly effective for language tasks; processes long contexts; flexible across domains	Requires significant computational resources; may hallucinate facts; interpretability remains difficult

Table 1: Comparison of major generative model types in deep learning

the evidence lower bound (ELBO), balancing reconstruction accuracy with a regularisation term that shapes a smooth latent space.

VAEs are generally stable and easier to train than GANs, and they provide explicit likelihood estimates that support tasks such as interpolation and anomaly detection. Their main limitation is output sharpness: generated images often appear blurrier compared to GAN results because of the trade-off between fidelity and regularisation. Even so, VAEs remain valuable where structured latent representations or probabilistic reasoning are important, particularly in scientific and hybrid generative systems.

Diffusion models: Diffusion models generate data by gradually denoising random noise, drawing inspiration from physical diffusion processes. They use two stages: a fixed forward process that incrementally adds noise to real data, and a learned reverse process that removes this noise step by step until structure reappears.

Formally, starting from a data sample $\mathbf{x}_0$, the forward diffusion produces:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T,$$

where $\mathbf{x}_T$ approaches a Gaussian noise distribution. A neural network is trained to reverse this corruption by predicting the clean sample from a noisy input at time step *t*. To synthesize new data, the model begins with:

$x_T \sim N(0, I)$ and applies $x_T \rightarrow x_{T-1} \rightarrow \cdots \rightarrow x_0$.

Because each denoising step is small and stable, diffusion models avoid the instability of adversarial training while producing highly detailed outputs. They currently achieve state-of-the-art results in image generation, often exceeding GANs in visual fidelity.

In practice, these models learn to transform pure noise into realistic samples, leading to strong diversity and quality, especially for images and audio. Training is typically stable but computationally intensive. Their main drawback is slow sampling, since many iterative denoising steps are required, and they do not provide a clear low-dimensional latent space. Even so, diffusion models have become a central technique in modern generative AI due to their reliability and output quality.

Figure 2 illustrates a diffusion model that generates an image through iterative denoising. Step 1 starts with a random noise image. Steps 2 and 3 are intermediate steps, where the model removes noise and begins to reveal structure (in this toy example, a simple shape).

Transformer-based generative models: The Transformer is a deep learning architecture that reshaped generative modelling, particularly in natural language processing. Originally built for sequence tasks, its defining feature is the self-attention mechanism, which allows the model to consider relationships across an entire sequence at once, improving parallelization and long-range coherence.

Transformers power large language models such as GPT-3 and later systems. These models generate text autoregressively by predicting one token at a time while referencing the full prior context, enabling fluent and consistent long-form output. Beyond text, Transformers are also used in image, music, and code generation, often combined with other techniques such as diffusion conditioning or discrete token modelling.

Transformer-based models excel at capturing global context and adapting to multiple tasks through prompting or fine tuning, exemplifying the ‘foundation model’ paradigm. However, they are resource intensive to train and run, and despite their fluency they can still produce inaccurate information. Their scale and complexity also make them difficult to interpret, which raises challenges for transparency and trust.

Applications of generative AI

Generative AI is being applied across many domains, from language and visual media to audio and video. These systems demonstrate strong potential in automating creative work, personalising digital content, and supporting business and research activities.

Text generation and language applications: Natural language generation is one of the most mature areas of generative AI. Large language models such as GPT-3, GPT-4, and Google’s LaMDA can produce coherent and context aware text from short prompts. This capability supports several practical uses:

- *Content creation:* Assisting with articles, marketing copy, reports, and brainstorming drafts.
- *Conversational agents:* Powering chatbots and virtual assistants that answer questions and hold interactive dialogues.
- *Translation and summarisation:* Generating translations or concise summaries of long documents within

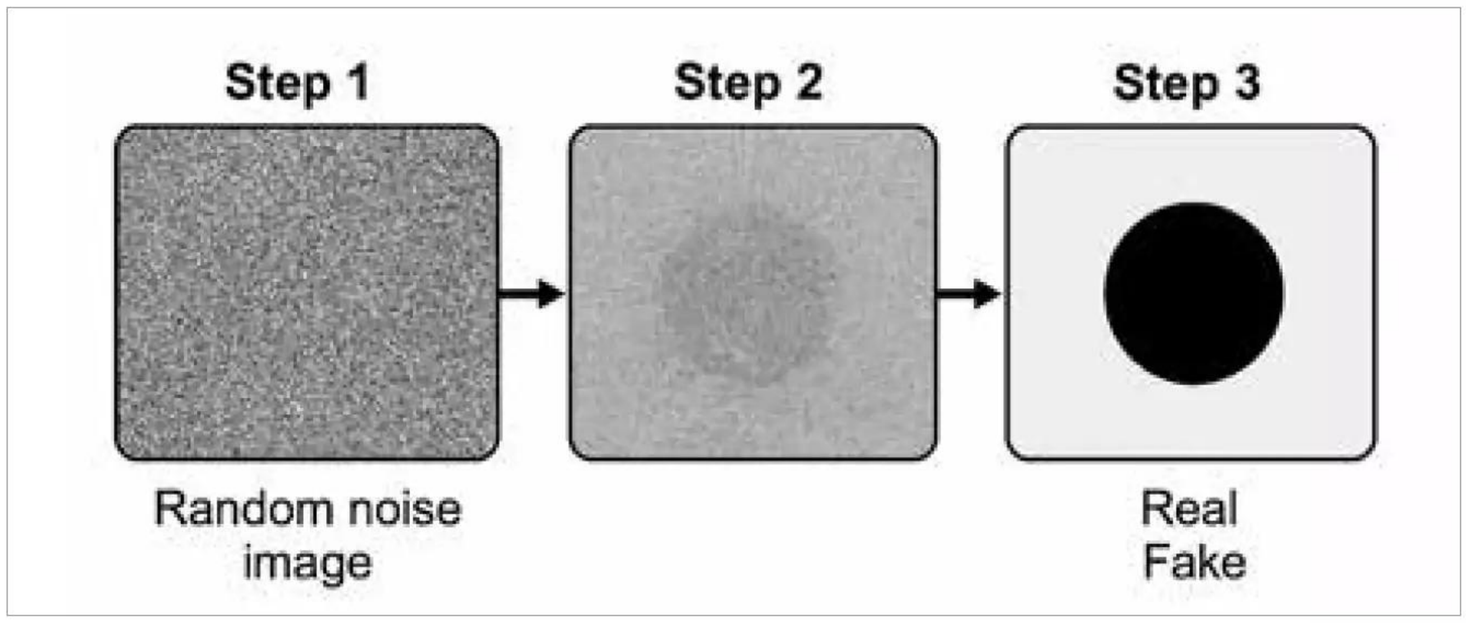


Figure 2: Illustration of a diffusion model

a unified framework.

- *Programming assistance:* Producing code snippets or auto completing functions from natural language descriptions.

These tools are increasingly used in customer service, education, and document drafting. However, human oversight remains important because generated text may occasionally be inaccurate or biased.

Image generation and creative design: Image generation has gained significant attention through models such as DALL·E, Stable Diffusion, and Midjourney, which create visuals from textual prompts. Built on GANs and diffusion techniques, these systems are widely used in:

- *Art and design:* Rapid concept exploration and creative experimentation.
- *Image editing:* Inpainting, style transfer, and super-resolution enhancement.
- *Marketing and advertising:* Producing customised visuals at scale without traditional photo shoots.
- *Product and architectural design:* Generating early-stage prototypes and design variations.

Industries such as e-commerce, gaming, and film-making increasingly adopt these tools, though professional workflows typically involve human review and refinement.

Audio and speech synthesis: Generative AI is also reshaping audio production, particularly in speech and music:

- *Speech generation:* Text-to-speech systems now produce natural sounding voices and enable controlled voice cloning.
- *Music composition:* Models can generate melodies, accompaniments, or adaptive background music.
- *Sound effects:* Dynamic generation of ambient noises and effects for games or virtual environments.

While these technologies enable personalisation and scalability, they also introduce ethical concerns related to voice imitation, copyright, and misuse.

Video generation and animation: Video generation is an emerging frontier for generative AI because maintaining temporal consistency across frames is difficult. Despite this challenge, steady progress is visible:

- *Deepfakes (face swapping):* Generative models can realistically replace faces in videos. While sometimes used for entertainment, deepfakes also enable misinformation and identity misuse, raising serious ethical concerns.
- *Video prediction and generation:* Recent research prototypes demonstrate short video creation from text prompts or initial frames, showing early but promising capabilities.
- *Animation and gaming:* AI tools assist in character animation, physics simulation, and content variation, helping reduce manual design effort in games and media production.

Although fully AI-generated films are still rare, industry use is growing in advertising variations, film restoration, frame interpolation, and virtual backgrounds in conferencing tools. Because humans are sensitive to visual artifacts and motion errors, most current applications focus on short or controlled clips, with quality expected to improve as models and hardware advance.

Ethical and societal considerations

Alongside its technical achievements, generative AI introduces significant ethical and societal challenges. Key concerns include bias, misinformation through deepfakes, hallucinated outputs, intellectual property questions, and broader social impacts. Addressing these issues is essential for responsible deployment and long-term trust.

Bias and fairness: Generative models learn from real-world data that may contain social or cultural biases, which can appear in generated text or images. Such outputs risk reinforcing stereotypes or unequal representation when used at scale. Mitigation requires better dataset curation, bias detection methods, and targeted fine tuning. Ensuring fairness remains an active research and industry priority, particularly in sensitive domains such as recruitment or public services.

Misinformation and deepfakes: Generative AI can produce highly realistic fake media, including images, audio, and video that may be used to mislead or manipulate audiences. This threatens public trust by making it harder to distinguish authentic evidence from fabricated content. In response, researchers are developing detection tools and provenance tracking, while policymakers explore legal safeguards. The dynamics between generation and detection continue to evolve as both technologies improve.

Hallucinations and reliability: Language and multimodal models sometimes produce confident but incorrect information, a phenomenon known as hallucination. This occurs because models predict patterns rather than verify factual truth. In high stakes settings such as healthcare or law, such errors can be harmful. Current mitigation strategies include retrieval augmented generation, grounding outputs in trusted data sources, and human oversight. Improving reliability and explainability is therefore central to building trustworthy generative AI systems.

Intellectual property and ownership: Generative AI complicates traditional ideas of authorship and copyright. When a model trained on large collections of artwork produces a new image, it is unclear who owns the result—the user, the model provider, or the original artists whose styles influenced the output. This uncertainty has already led to legal disputes and ongoing policy debates. While some users view AI outputs as original creations, others argue they may be derivative of training data. Companies are beginning to clarify usage rights, but the legal landscape remains unsettled.

Beyond copyright, ethical concerns arise when AI imitates the style or voice of living artists without consent or compensation.

***Societal and economic impact*:** Generative AI's ability to automate creative tasks raises questions about employment and content quality. Professions such as design, writing, and media production may experience disruption, although new roles centered on supervising or collaborating with AI are also emerging. Education and professional training will likely need to adapt as these tools become common in workflows. Another concern is information overload: when content can be generated at scale, distinguishing reliable and high-quality material becomes more difficult, increasing the risk of spam and misinformation.

In summary, generative AI offers remarkable opportunities but challenges existing norms of creativity, truth, and ownership. Responsible development requires guidelines for fairness, transparency, and accountability. Technical safeguards such as bias auditing and content filtering, along with collaboration between technologists, policymakers, and social scientists, are essential to balance innovation with societal well-being.

Challenges and future directions

Despite rapid progress, generative AI faces several technical and ethical challenges.

***Alignment with human values*:** Ensuring that generative systems behave in ways consistent with human intentions and social norms is a central concern. Techniques such as reinforcement learning from human feedback (RLHF) and content moderation help reduce harmful outputs, but long-term alignment remains an open research problem. Future approaches may involve richer feedback loops, multi-agent critique systems, and broader stakeholder participation in defining acceptable behaviour.

Generalisation and robustness: Generative models often perform well within their training distribution but struggle with unfamiliar scenarios. Improving generalisation may involve combining neural approaches with reasoning modules or enabling rapid adaptation through fewshot learning. Robustness testing, including adversarial evaluation, is increasingly important to ensure reliability across varied inputs and conditions.

Explainability and interpretability: Large generative networks typically function as black boxes, making it difficult to understand why a particular output was produced. This lack of transparency is problematic in sensitive domains. Ongoing research explores attention visualisation, probing techniques, and modular system designs to provide clearer rationales and increase user trust.

Efficiency and scalability: Modern generative models demand significant computational resources for both training and inference. Improving efficiency through model compression, quantization, and more optimised architectures is a key direction. Reducing energy consumption and hardware dependence will be crucial for broader accessibility and sustainability.

Multimodal and general-purpose AI: A clear trend is the development of models capable of handling multiple data types: text, images, audio, and video within unified systems. Such multimodal assistants could coordinate diverse generative tasks, but they also introduce new challenges in training complexity, evaluation, and alignment across domains.

***Governance and regulation*:** As generative AI becomes widespread, governance and policy frameworks will play a larger role. Industry standards for transparency, certification mechanisms, and regulatory safeguards for high-risk applications are likely to emerge. Technical safety measures, combined with interdisciplinary collaboration, will help ensure responsible deployment.

In summary, the future of generative AI is promising but demands careful management of risks and limitations. Progress in alignment, efficiency, and transparency alongside thoughtful governance will determine how effectively these systems augment human creativity and problem solving.

Generative AI has moved in a short time from being a specialised research topic to a major force in modern artificial intelligence. It marks a shift from computers that only analyse information to systems that can actively create it. If developed thoughtfully, it can support human creativity, speed up innovation, and provide personalised solutions on a large scale. The future of generative AI will depend not only on technical breakthroughs, but also on how carefully and responsibly we choose to guide its growth. END



References

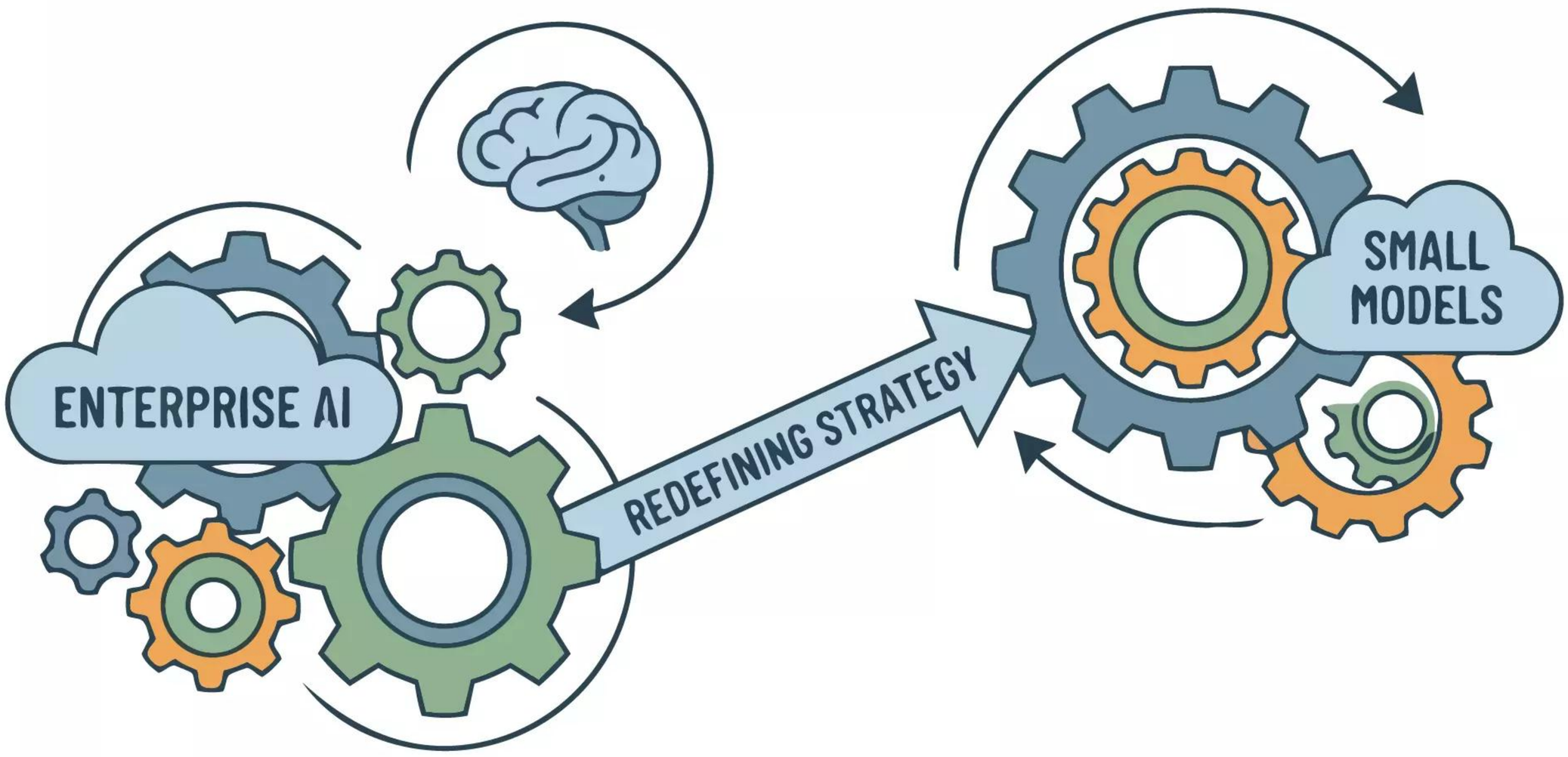
- Goodfellow, J. Pouget-Abadie, M. Mirza, et al, 'Generative Adversarial Nets', in Proc. of NeurIPS, 2014, pp. 2672–2680
- D.P. Kingma and M. Welling, 'Auto-Encoding Variational Bayes', arXiv preprint arXiv:1312.6114, 2013 (ICLR 2014)
- J. Ho, A. Jain, and P. Abbeel, 'Denoising Diffusion Probabilistic Models', in Proc. of NeurIPS, 2020
- A. Vaswani, N. Shazeer, et al, 'Attention Is All You Need', in Proc. of NeurIPS, 2017, pp. 5998–6008
- R. Bommasani, J.K. Wei, *et al,* 'On the Opportunities and Risks of Foundation Models', Center for Research on Foundation Models (CRFM), Stanford, White Paper, 2021

By: Rishi Patel and B. Thangaraju

The authors are associated with the Open Source Technology Lab at the International Institute of Information Technology, Bengaluru.

The Quiet Revolution: How Small Language Models are Redefining Enterprise AI Strategy

Small language models are proving to be game changers in the AI world. Deploying these models early on can give organisations an edge over the competition.



While the industry fixates on frontier large language AI models with hundreds of billions of parameters, a parallel transformation is underway. Small language models (SLMs)—typically ranging from 1B to 13B parameters—are proving that intelligent scale, not massive scale, drives business value. Organisations deploying SLMs are achieving 200-400% ROI within the first year, with deployment cycles measured in weeks rather than quarters.

This shift represents more than cost optimisation. It represents a fundamental rethinking of how AI creates competitive advantage through speed of deployment, precision of application, data sovereignty, and operational resilience. The question is no longer whether to adopt small models, but how quickly your organisation can architect around them before competitors do.

Real-world ROI

The conventional wisdom that larger models inherently deliver better outcomes is being challenged by empirical evidence across industries. Consider three representative scenarios that illustrate the strategic value of architectural precision over computational excess.

Customer service transformation: A Fortune 500 financial services firm replaced its third-party large model API with a fine-tuned 7B parameter model for tier-1 customer enquiries. The result: 89% accuracy on domain-specific queries, sub-100ms response times, and a 73% reduction in inference costs. More critically, they achieved full data residency compliance across all global markets—something not possible with API-based approaches. Time to full production: 11 weeks; first-year ROI: 340%.

Contract intelligence at scale: A legal technology provider implemented a specialised 3B parameter model for contract clause extraction and risk assessment. By training on their proprietary corpus of 2 million

contracts, they achieved a 94% F1 score on clause identification—outperforming general-purpose models 3X their size. The architecture processes 10,000 contracts daily at one-tenth the cost of their previous solution, with complete audit trails and explainability. Deployment time: 6 weeks; payback period: 4 months.

Manufacturing intelligence: An industrial equipment manufacturer deployed edge-based 1.5B parameter models across 47 manufacturing facilities for predictive maintenance and quality control. Running on local GPUs, these models process sensor data in real time, trigger maintenance workflows, and generate natural-language reports for operators. The system operates during network outages, meets strict data locality requirements, and reduces unplanned downtime by 41%. Implementation across all sites: 9 weeks; annual savings: $14.2M.

Why SLMs work well

While economic efficiency drives initial interest in SLMs, sustained competitive advantage emerges from four architectural properties.

Deployment velocity: Small models can be fine-tuned, validated, and deployed in weeks rather than quarters. This agility transforms AI from a strategic initiative into a tactical capability, enabling rapid response to market changes, regulatory requirements, or competitive threats.

Precision through specialisation: Domain-specific training creates models that deeply understand industry jargon, regulatory frameworks, and operational context. A 7B model trained on your data often outperforms a 70B generalist model on your specific tasks—while running 10X faster and consuming 90% less infrastructure.

Data sovereignty and risk mitigation: On-premises or VPC-deployed SLMs provide complete control over data residency, model behaviour, and compliance posture. This matters increasingly as regulations tighten globally and as organisations grapple with the liability implications of third-party AI decisions.

Operational resilience: Organisations running their own models avoid single-vendor dependencies, API throttling, unexpected pricing changes, and service degradation. Edge deployments enable critical operations to continue during network failures or cloud outages.

The architecture and deployment: Building for intelligent scale

Successful SLM deployments share a common architectural philosophy: modular, composable, and observable. Rather than monolithic AI platforms, organisations are building layered systems where each component can be independently optimised, scaled, and evolved.

Model foundation (Layer 1): The base model selection drives all downstream decisions. Modern SLM architectures leverage transformer variants optimised for efficiency: Mistral 7B and Mixtral 8x7B for general intelligence, Phi-3 family for reasoning tasks, Llama 3 variants for balanced performance, and Gemma models for Google ecosystem integration. The choice depends on three factors: task complexity, latency requirements, and available infrastructure.

- *Critical consideration:* Model licence compatibility with target deployment model. Apache 2.0 and MIT licences permit unrestricted commercial use; some licences impose restrictions on derivative works or competitive use.

Data and fine-tuning pipeline (Layer 2): The differentiator between commodity and competitive advantage. This layer encompasses data curation, quality assurance, synthetic data generation, and continuous training pipelines. Leading implementations use DVC for data versioning, Label Studio or Argilla for annotation workflows, and automated quality filters to maintain training data hygiene.

- *Critical consideration:* Fine-tuning strategies range from full fine-tuning for maximum performance to LoRA and QLoRA for parameter-efficient adaptation. Organisations typically start with instruction tuning on 1,000-10,000 examples, then iterate based on production feedback. The key is treating fine-tuning as a continuous process, not a one-time event.

Inference and serving infrastructure (Layer 3): Where architecture meets reality. The inference layer must balance throughput, latency, and cost across diverse deployment targets: cloud, edge, and hybrid configurations. Modern stacks use vLLM or TensorRT-LLM for GPU optimisation, delivering 10-30X throughput gains over naive implementations. For CPU-only environments, llama.cpp and GGUF quantization enable surprisingly capable inference on commodity hardware.

- *Critical consideration:* Production rollout includes model quantization strategy, batching policies, request-routing logic, failover behaviour, and cost allocation. Organisations achieving sub-50ms p95 latency typically employ sophisticated caching, speculative decoding, and continuous batching techniques.

Integration and orchestration (Layer 4): SLMs rarely operate in isolation. This layer provides retrieval-augmented generation, tool use capabilities, multi-step reasoning, and integration with enterprise systems. LangChain and LlamaIndex provide foundational abstractions; production systems typically extend these with custom components for error handling, retries, circuit breaking, and business logic.

- *Key architectural decision:* Local-first vs cloud-dependent retrieval. Organisations handling sensitive

data increasingly implement on-premises vector stores using Qdrant, Weaviate, or Milvus rather than managed services, ensuring end-to-end data sovereignty.

Observability and governance (Layer 5): Production AI systems require specialised observability beyond traditional application monitoring. This layer tracks model performance metrics, data drift, latency distributions, cost per inference, and business impact. Tools like Weights & Biases, MLflow, and Langfuse provide experiment tracking and production monitoring. Mature organisations implement real-time alerting on accuracy degradation, automated model rollback, and continuous evaluation against golden datasets.

- *Critical consideration:* Deployment speed determines competitive impact. Organisations achieving production deployment in under 90 days follow a disciplined progression from proof of concept to scaled operations.

It's important to identify processes where AI can reduce cycle time, improve accuracy, or enable previously impossible capabilities. Ideal first use cases have clear success metrics, accessible training data, and tolerance for 85-90% accuracy. Avoid starting with mission-critical systems or areas requiring 99%+ precision. Establish a cross-functional team including domain experts, data scientists, infrastructure engineers, and security stakeholders. Define success criteria, budget parameters, and governance framework.

Multiple candidate models must be deployed using Hugging Face, Ollama, or LM Studio for rapid experimentation. Test each model against representative tasks using your actual data. Measure task-specific accuracy, latency, and resource consumption. This phase should feel experimental—testing assumptions, discarding what doesn't work, and iterating daily. Conduct prompt engineering experiments before committing to fine-tuning; well-crafted prompts can bridge surprising performance gaps.

Curate 1,000-5,000 high-quality examples rather than 100,000 mediocre ones. Use domain experts to validate training data. Implement systematic quality checks: Are instructions clear? Are outputs consistent? Does the data reflect actual business requirements? Begin fine-tuning on cloud-based GPU resources (vast.ai, RunPod, or AWS/GCP/Azure) using parameter-efficient methods such as LoRA. This enables rapid iteration without massive infrastructure investment. Continuously evaluate against holdout test sets.

Deploy infrastructure starting with the inference server, monitoring, and basic orchestration. Use containerization (Docker) and orchestration (Kubernetes or simpler alternatives like Docker Compose for early stages) to enable consistent deployment across environments. Implement comprehensive logging, latency tracking, and cost monitoring from day one. Set up A/B testing infrastructure to compare model performance against baseline systems. Establish incident response procedures and rollback mechanisms before user exposure.

Monitor obsessively. Collect feedback systematically. Measure against the success criteria defined at the start of the design phase. Document lessons, celebrate wins, and prepare for scaled rollout. The goal is proof that the system creates measurable business value, not perfection.

Navigating the challenges

Successful deployments encounter predictable obstacles. Anticipating these challenges enables proactive mitigation rather than reactive crisis management.

The cold start problem: Start with synthetic data generation using larger models or commercial APIs to create initial training corpus. Use active learning to identify high-value examples for human annotation. Leverage transfer learning from similar domains. Consider data partnerships or licensed datasets for rapid bootstrapping. Many organisations achieve surprising results with just 500-1000 carefully curated examples.

Infrastructure complexity: Adopt managed inference platforms like Together AI, Replicate, or Modal for early-stage deployments. As scale increases, evaluate build vs buy decisions based on true total cost of ownership—including engineering time, opportunity cost, and strategic value of in-house capability. Many organisations reach optimal economics with hybrid approaches — managed

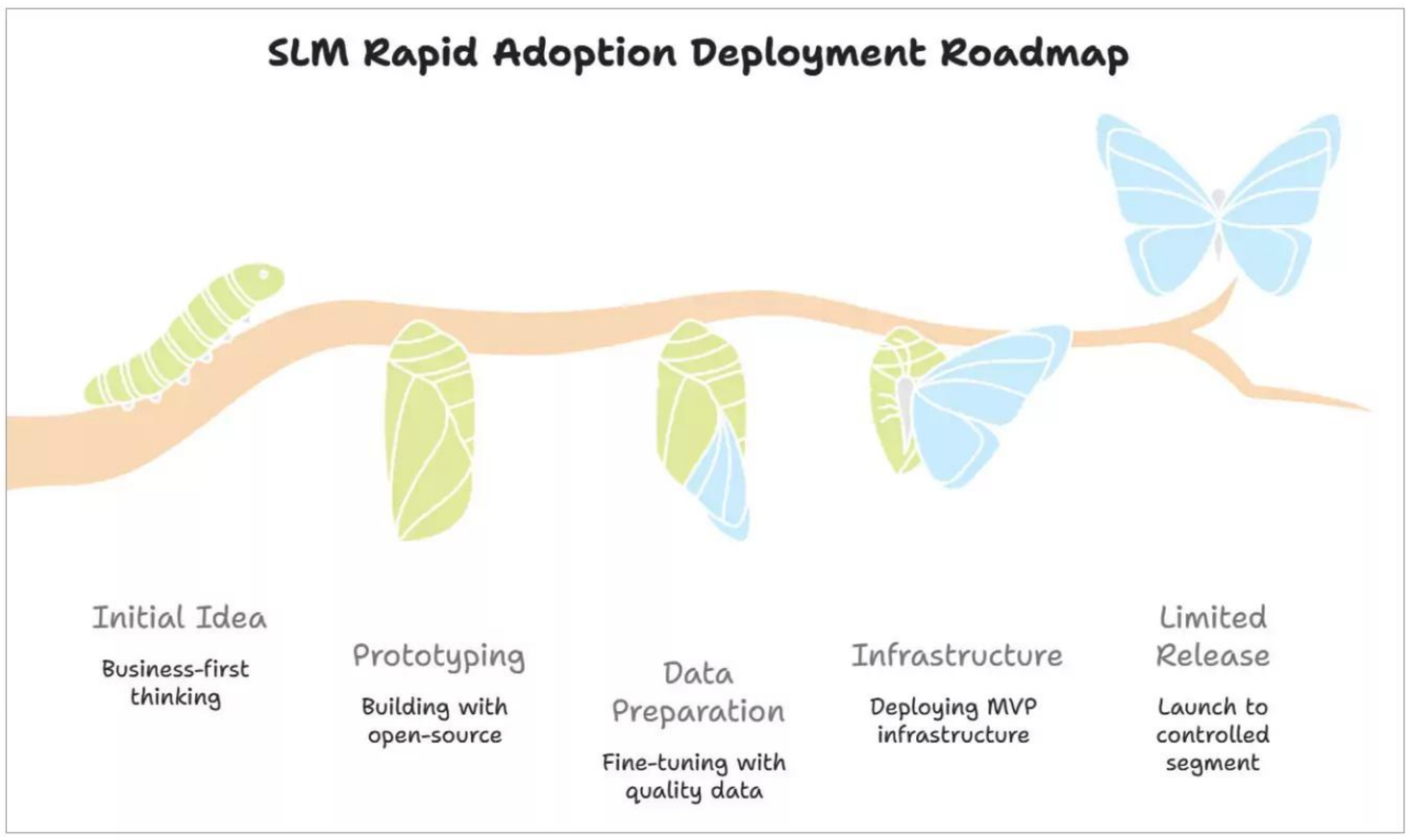


Figure 1: Small language model (SLM) rapid adoption deployment roadmap

services for experimentation and low-volume workloads, self-hosted for high-volume production.

Model performance degradation: Implement continuous monitoring with automated alerts on accuracy drops. Establish feedback loops capturing production errors for retraining. Schedule regular model refresh cycles—monthly or quarterly, depending on domain volatility. Maintain shadow deployments of candidate models to enable rapid switching when primary models degrade. Treat model operations as ongoing programmes, not one-time projects.

Organisational resistance: Design for human-in-the-loop workflows initially. Present AI outputs as recommendations requiring approval rather than automated actions. Provide transparency into model reasoning through attention visualisation or chain-of-thought outputs. Measure and communicate impact in business terms: time saved, revenue generated, errors prevented. Build trust through consistent performance and clear escalation paths when the system encounters edge cases.

Security and compliance concerns: Engage security and compliance stakeholders from day one. Deploy in secure environments with appropriate network isolation, encryption at rest and in transit, and access controls. Implement comprehensive audit logging. For sensitive data, use on-premises or VPC deployments with no data egress. Document data handling procedures, model training provenance, and decision-making processes. Many organisations find that local SLM deployments actually simplify compliance compared to third-party API usage.

Best practices for SLM deployment

Organisations operating SLMs at scale converge on similar operational patterns. These practices separate experimental deployments from business-critical systems.

Resist the temptation to build general-purpose AI platforms. Begin with a single, well-defined use case with clear success metrics. Prove value. Then expand systematically to adjacent use cases, leveraging shared infrastructure and learned expertise. Organisations attempting to boil the ocean with universal AI initiatives consistently underdeliver compared to those pursuing focused, sequential deployments.

The performance ceiling is determined by training data quality, not model size. Allocate more resources to data curation, cleaning, and validation than to infrastructure or compute. Establish clear data quality standards. Review training examples manually. Remove ambiguous or contradictory data. This unglamorous work drives outsized performance improvements.

Modern quantization techniques enable 4-bit or 8-bit inference with minimal accuracy loss, reducing memory requirements by 4-8x and increasing throughput proportionally. Every SLM deployment should evaluate quantized variants. The performance-efficiency tradeoff typically favors quantization except for the most demanding tasks. This single optimisation can eliminate 75% of infrastructure costs.

Automated evaluation against golden datasets should run on every code commit, model update, and production deployment. Maintain diverse test suites covering common cases, edge cases, adversarial inputs, and business-critical scenarios. Track metrics over time to detect gradual degradation. Production systems should continuously evaluate a sample of outputs against human judgment, feeding this data back into training pipelines.

Assume your current model will be replaced within 6-12 months. Abstract model interfaces to enable swapping base models without application

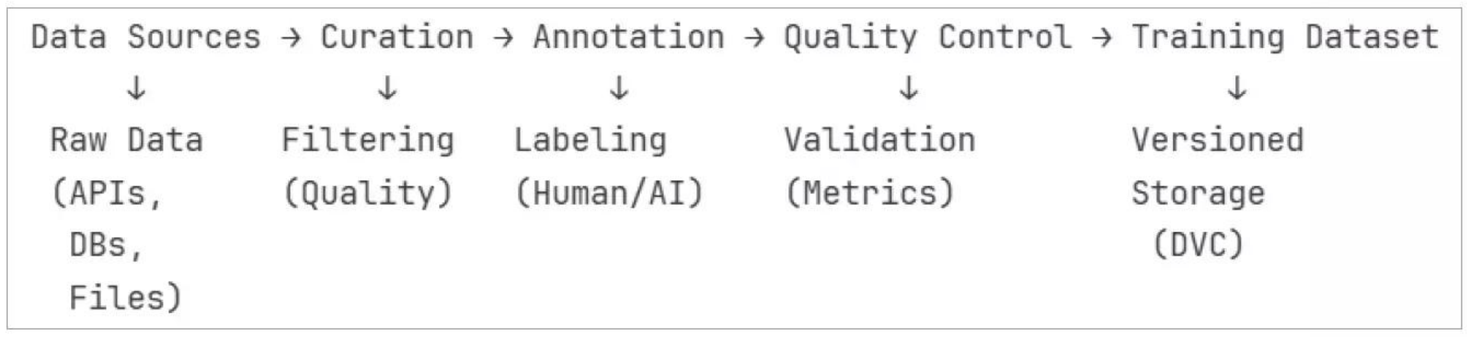


Figure 2: Data pipeline architecture

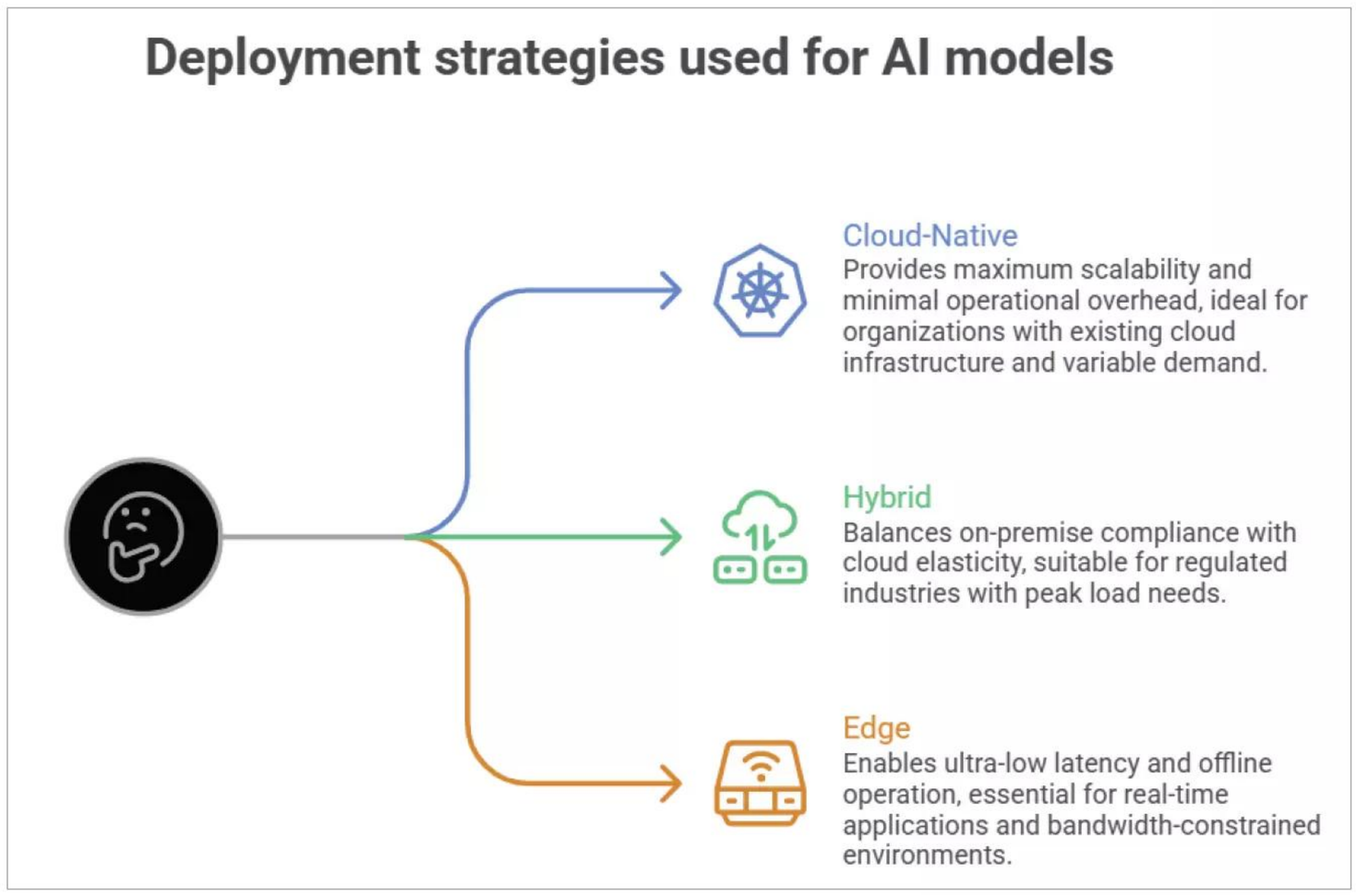


Figure 3: Integration patterns for enterprise environments

Layer	Component	Open source tools
Model foundation	Base models	Mistral 7B, Mixtral 8x7B, Llama 3.3 70B, Phi-4, Gemma 2 (2B-27B), Qwen2.5 (0.5B-72B)
	Model hub	Hugging Face Hub, Ollama
Data pipeline	Data versioning	DVC, Git LFS, LakeFS
	Annotation	Label Studio, Argilla, Prodigy
	Synthetic data	NeMo Curator, DataTrove, Lilac
Training and fine-tuning	Training framework	Hugging Face Transformers, Axolotl, LLaMA-Factory, PEFT
	Distributed training	DeepSpeed, Accelerate, PyTorch FSDP
	Experiment tracking	Weights & Biases, MLflow, TensorBoard
Inference infrastructure	GPU inference	vLLM, TensorRT-LLM, Text Generation Inference (TGI), LocalAI
	CPU inference	llama.cpp, GGML/GGUF, ONNX Runtime
	Model serving	Ray Serve, BentoML, KServe, Triton Inference Server
	Quantization	bitsandbytes, GPTQ, AWQ, AutoGPTQ
Orchestration	Application framework	LangChain, LlamaIndex, Haystack, Semantic Kernel
	Vector database	Qdrant, Weaviate, Milvus, Chroma, pgvector
	Agent framework	AutoGen, CrewAI, LangGraph, OpenAI Swarm
Deployment	Containerisation	Docker, Podman
	Orchestration	Kubernetes, Docker Compose, Nomad
	CI/CD	GitHub Actions, GitLab CI, Jenkins, ArgoCD
Observability	LLM monitoring	Langfuse, Phoenix, LangSmith (commercial with free tier)
	Infrastructure	Prometheus, Grafana, Loki, Jaeger
	Evaluation	DeepEval, RAGAS, TruLens, PromptFoo
Security and governance	Model security	NeMo Guardrails, Llama Guard, Prompt Armor
	Model registry	MLflow Model Registry, Hugging Face Hub (private repos)

rewrites. Maintain version-controlled model registries. Implement blue-green deployment patterns enabling zero-downtime upgrades. The AI landscape evolves rapidly; architectural flexibility provides lasting competitive advantage while specific model choices do not.

Technical metrics like perplexity and BLEU scores matter less than business outcomes. Define success in terms stakeholders care about: customer satisfaction scores, processing time reduction, error rate improvement, revenue impact, or cost savings. Connect AI system performance directly to business KPIs. This alignment ensures continued investment and organisational support.

Open source tech stack by SLM reference architecture layer

The stack in the table above enables progressive sophistication. Organisations can begin with a simple configuration: base model, basic inference, minimal monitoring, and systematically add capabilities as requirements evolve. Each layer remains independently swappable, preventing vendor lock-in and enabling continuous optimisation.

The strategic opportunity in small language models extends beyond current cost savings or operational improvements. Organisations building competency in SLM deployment are establishing capabilities that compound over time: proprietary training data, domain-specific model expertise, optimised inference infrastructure, and institutional knowledge of what works.

Early movers are already seeing network effects. Each production deployment generates training data for the next iteration. Each solved integration problem becomes reusable infrastructure. Each successful use case builds organisational confidence and accelerates subsequent deployments. The gap between AI leaders and

followers widens not through access to better models—those are freely available—but through their ability to deploy them systematically.

The question facing leadership teams is increasingly clear. As open source models approach or exceed proprietary model performance across narrow domains, and as deployment tools mature towards commodity status, competitive advantage accrues to organisations that move fastest from strategy to architecture to production at scale.

The quiet revolution in efficient AI is creating a window of opportunity measured in quarters, not years. Organisations that master the architecture, operations, and business integration of small language models are building sustainable advantages in an AI-enabled world. Those who wait for perfect clarity or mature platforms may find that the advantage has already been claimed by faster-moving competitors.

The infrastructure exists. The models are available. The case studies validate the approach. The remaining variable is organisational will and execution velocity. The leaders of 2026 will be defined not by who talks about AI strategy, but by who deploys it into production this quarter. END

References

- Hugging Face - Open-Source AI Platform and Model Hub: *https://huggingface.co*
- Mistral AI - Frontier Open Models: *https://mistral.ai*
- vLLM - High-Throughput Inference Engine: *https://github.com/vllm-project/vllm*
- LangChain - LLM Application Framework: *https://www.langchain.com*
- Ollama - Local Model Deployment Platform: *https://ollama.ai*
- Meta AI Research - Llama Model Family: *https://ai.meta.com/llama/*
- Weights & Biases - ML Experiment Tracking and Monitoring: *https://wandb.ai*
- Qdrant - Vector Database for AI Applications: *https://qdrant.tech*
- Axolotl - Fine-Tuning Framework: *https://github.com/OpenAccess-AI-Collective/axolotl*
- NVIDIA TensorRT-LLM - GPU Inference Optimization: *https://github.com/NVIDIA/TensorRT-LLM*

By: Bala Kalavala

The author is a seasoned technologist and an enthusiastic digital-transformation expert at a global consulting firm. He is a sought-after keynote speaker, evangelist, tech blogger, angel investor, and a serial entrepreneur.

Disclaimer: *The opinions expressed in this article are the author's and do not reflect the views of the organisation he works in.*

Statement about ownership and other particulars about
OPEN SOURCE FOR YOU
FORM IV (See Rule 8)

1. Place of publication	:	New Delhi
2. Periodicity of its publication	:	Monthly
3. Printer's Name	:	Ramesh Chopra
Nationality	:	Indian
Address	:	OPEN SOURCE FOR YOU D-87/1, Okhla Industrial Area, Phase-1, New Delhi 110020
4. Publisher's Name Nationality and Address	:	Same as (3) above
5. Names and addresses of individuals who own the newspaper & partners or shareholders holding more than 1% of the total capital	:	**EFY Enterprises Pvt Ltd.** D-87/1, Okhla Industrial Area, Phase-1, New Delhi 110020

I, Ramesh Chopra, hereby declare that the particulars given above are true to the best of my knowledge and belief.

Date: 28-2-2026

Ramesh Chopra
Publisher

Popular FOSS Tools for LLM Observability, Monitoring and Evaluation

This overview of popular tools for monitoring large language models also sheds light on how LLM-as-a-judge enhances their performance.



It's no secret that, these days, the corporate world as well as the government sector is using AI chatbots and large language models (LLMs) for data analytics and decision making. The key strengths of AI chatbots that use LLMs include multimodal search, content creation, programming, applications development, coding, scripting, multimedia creation, task automations, academic research, speech analytics, and data engineering.

In India, many MNCs as well as domestic IT companies are working on the development and deployment of AI chatbots for different applications. As per a report by the Bank of America, cited in *The Economic Times*,

Table 1: Popular AI chatbots for different applications

ChatGPT	Gemini
Chatbot	Claude
Perplexity	Microsoft Copilot
Meta AI	Grok
Pi	Qwen Chat
Mistral Chat	YouChat
Ernie Bot	HuggingChat
Character.AI	Jasper Chat
Baichuan Chat	Writesonic Chat
Bing Chat	DeepSeek Chat
TII Falcon Chat	Replika
Kimi	SenseChat
Suno AI Chat	Yi Chat
ChatGLM	ChatPDF



Figure 1: Key use cases of AI LLMs

Table 2: Key tools for performance evaluation and analytics of LLMs

Name	Core modules/ Capabilities	Key applications	URL
Comet Opik	Prompt/response logging, trace visualisation, experiment tracking, evaluations, OpenTelemetry support	LLM observability, evaluation and debugging	*comet.com/opik*
OpenLit	OpenTelemetry instrumentation, LLM spans, latency and token metrics, vendor-agnostic	LLM observability, tracing, prompt/response monitoring	*openlit.io*
OpenTelemetry	Traces, metrics, logs, exporters (Jaeger, Prometheus, console)	Unified observability backbone	*opentelemetry.io*
Helicone	Request proxying, latency analysis, prompt logs, error monitoring	LLM request monitoring and debugging	*helicone.ai*
Langfuse	Prompt versioning, trace trees, scoring, datasets, feedback loops	Prompt tracking, cost analysis, evaluations	*langfuse.com*
Promptfoo	Regression testing, prompt scoring, multi-model evals	Prompt testing and comparison	*promptfoo.dev*
Traceloop	OpenTelemetry-native tracing, framework integrations (LangChain, LlamaIndex)	LLM pipeline tracing	*traceloop.com*
Phoenix	Embedding drift, trace visualisation, evaluation dashboards	LLM quality and hallucination analysis	*phoenix.arize.com/*
Ragas	Faithfulness, relevance, context precision/recall	RAG and LLM output evaluation	*github.com/explodinggradients/ragas*
Evidently AI	Drift detection, data quality, evaluation reports	Data and model monitoring	*evidentlyai.com*
MLflow	Runs, metrics, artifacts, model registry	Experiment tracking (LLM-adaptable)	*mlflow.org*
DeepEval	Test cases, metrics, CI/CD-ready evaluation	Unit testing for LLMs	*github.com/confident-ai/deepeval*
Jaeger	End-to-end request tracing, span timelines	Trace visualisation	*jaegertracing.io*
Prometheus	Time-series metrics, alerting	Metrics monitoring	*prometheus.io*
Grafana	Traces, metrics, logs dashboards	Visualisation and dashboards	*grafana.com*
Ollama	Local model serving, OpenAI-compatible API	Offline/local LLM execution	*ollama.com*

India is a leader in the adoption and implementation of AI models.

LLM-as-a-judge: Observation, monitoring and analytics of LLM platforms

LLM-as-a-judge is a technology for evaluating the performance of AI LLMs in terms of effectiveness of prompts, tokens used, cost factor, and many other parameters that directly affect the execution of the deployment.

The LLM-as-a-judge service provides observation and analytics with use cases centered on prompt lifecycle management—including assessing prompt effectiveness, ensuring traceability and versioning, and analysing cost factors—alongside operational monitoring of latency and token accounting. Key advantages are robust governance and quality assurance, offering comprehensive response auditing, error diagnosis, and outcome visibility with capabilities for hallucination detection and regression testing. Furthermore, it enables

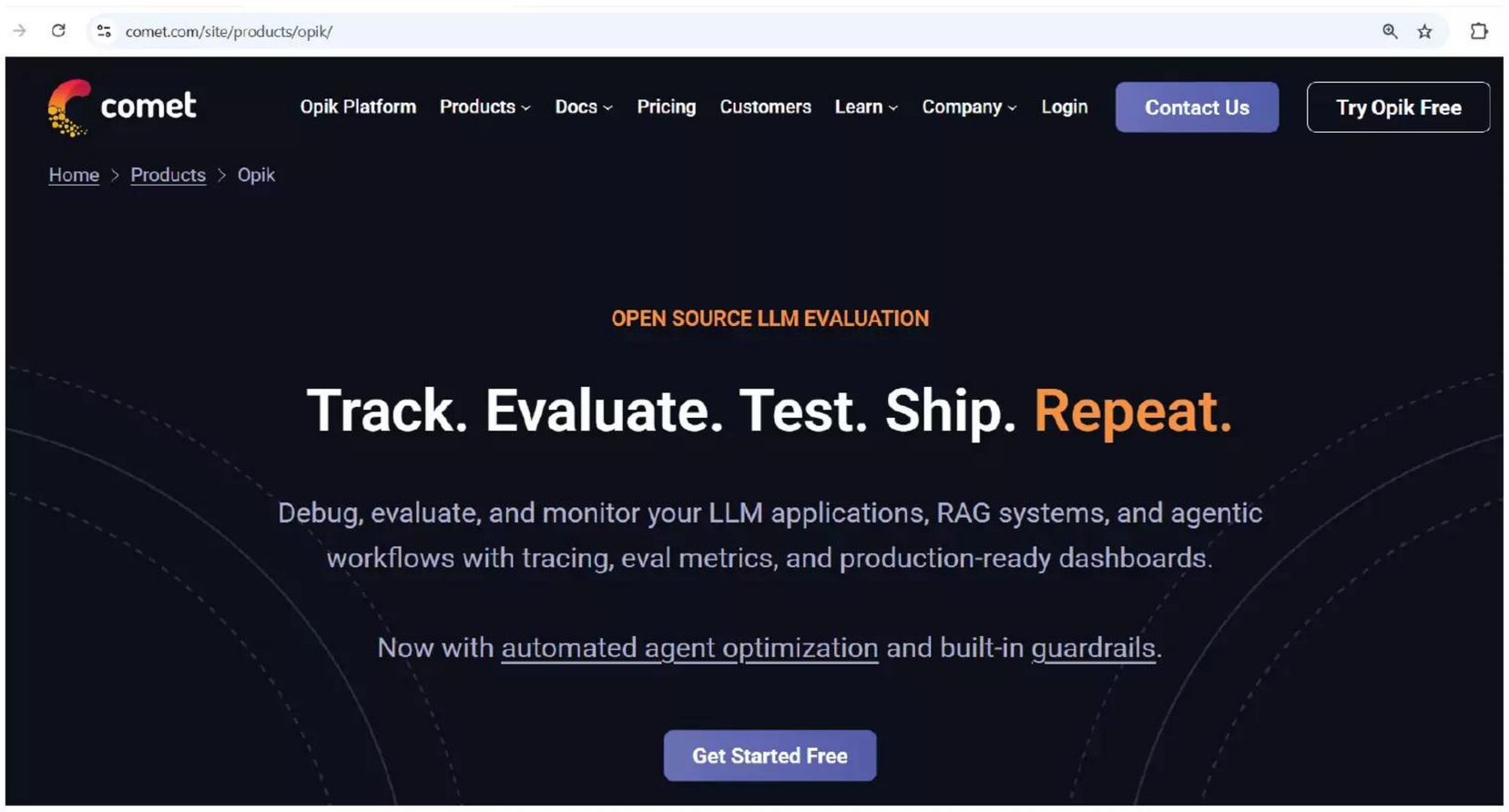


Figure 2: Comet Opik for LLM analytics

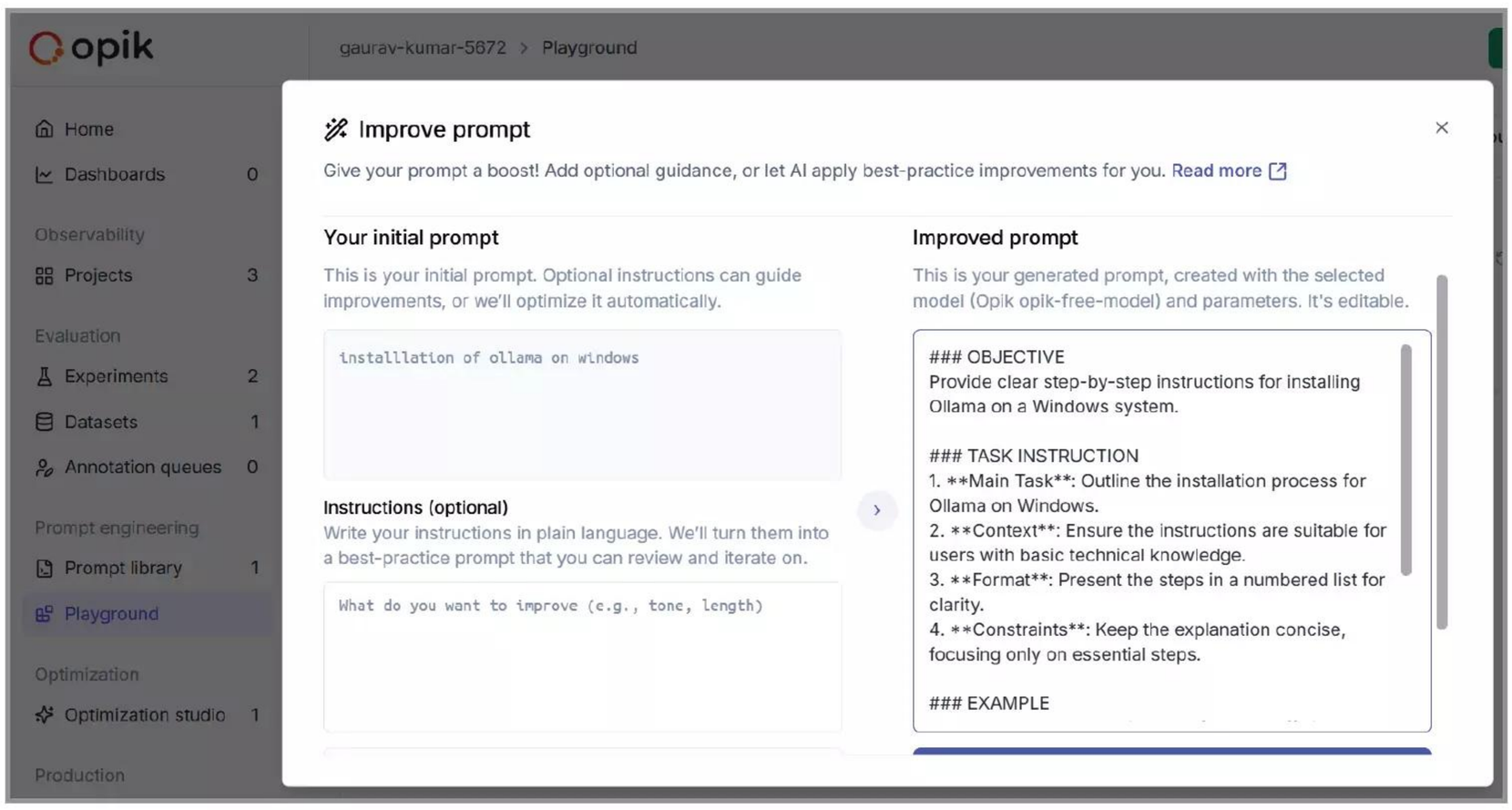


Figure 3: LLM prompt improvement in Comet Opik

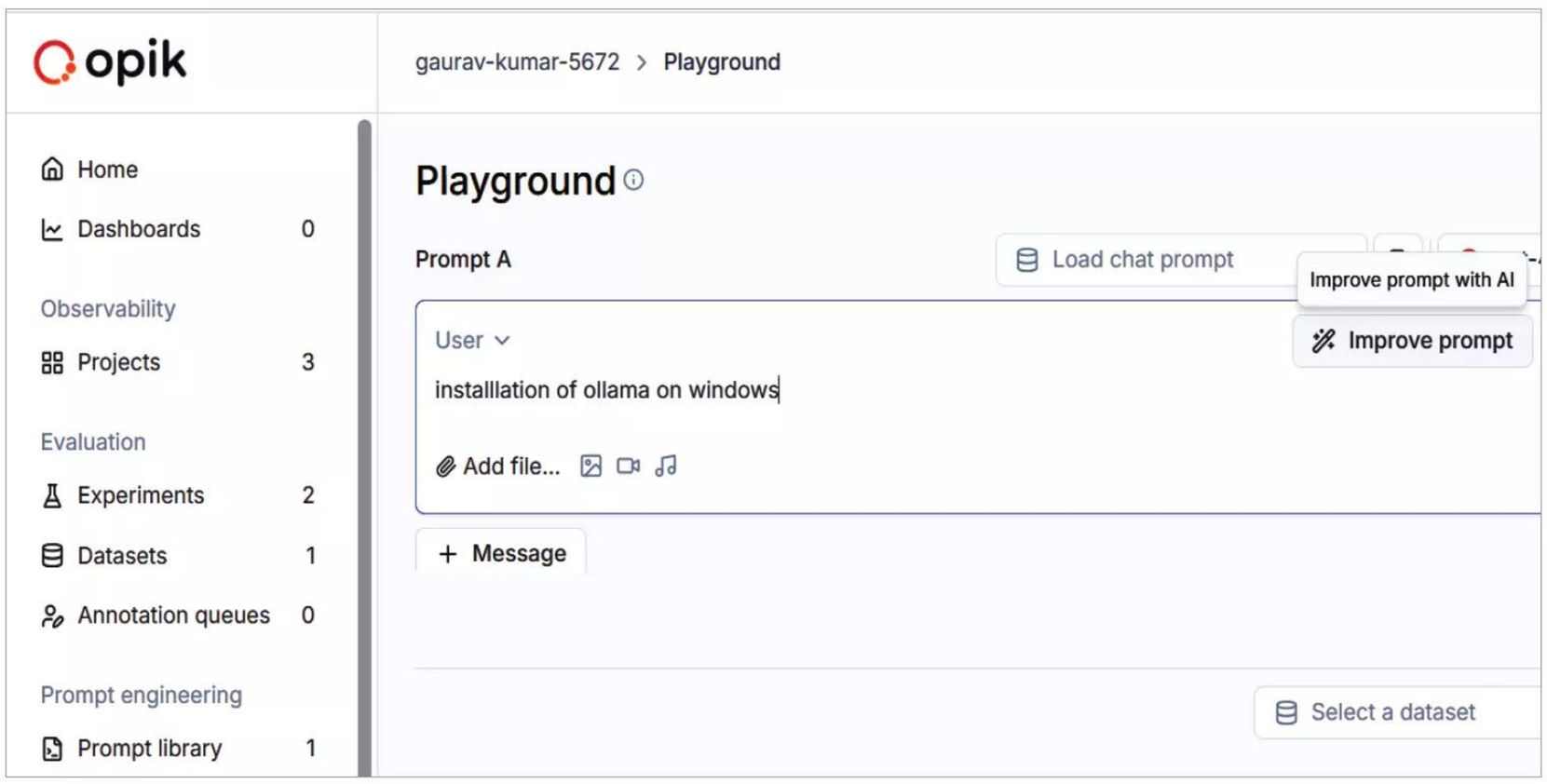


Figure 4: Playground for prompt processing in Comet Opik

rigorous model evaluation through comparison, drift detection, and scoring, while ensuring reliability and compliance via reproducible experiments, audit-ready evidence, and governance readiness, all supported by vendor-neutral debugging pipelines and secure offline deployments.

Several tools are available for observability and analytics of AI LLMs for the cloud as well as for dedicated deployment on site (Table 2). Using these tools, the performance of LLMs can be evaluated and optimised.

Working with Comet Opik

(comet.com/opik, comet.com/site/products/opik/)

Comet Opik is a powerful and multi-featured platform for observability, analytics and evaluation of LLMs, including prompts.

Opik provides a range of features for debugging, evaluating and monitoring LLM applications, RAG implementations and agentic AI based workflows so that tracing and metrics analytics can be done effectively for enhancing the overall performance of the deployment. It integrates optimisation and benchmarking features with ease of implementation.

Comet Opik is available for cloud as well as dedicated deployments. Here's how to install it on Windows:

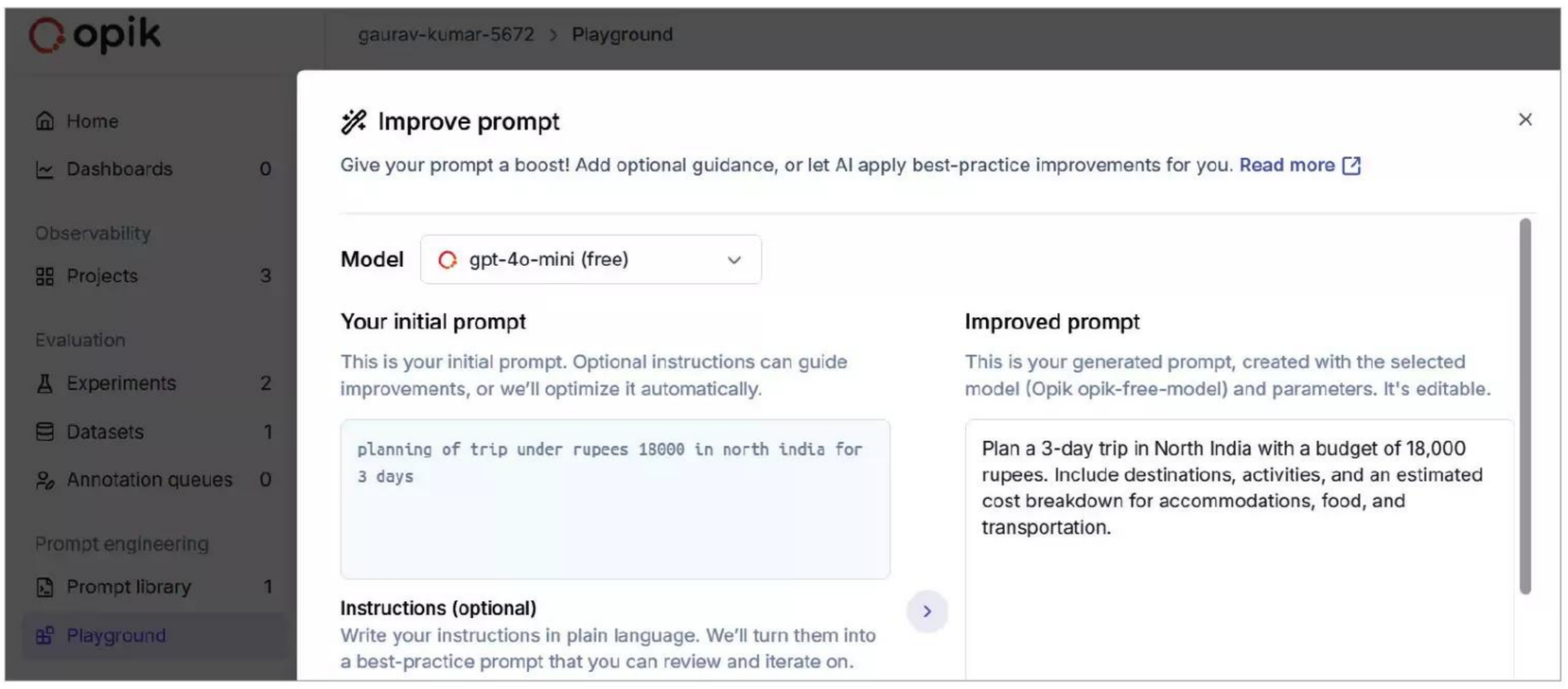


Figure 5: Initial and improved prompt in Comet Opik

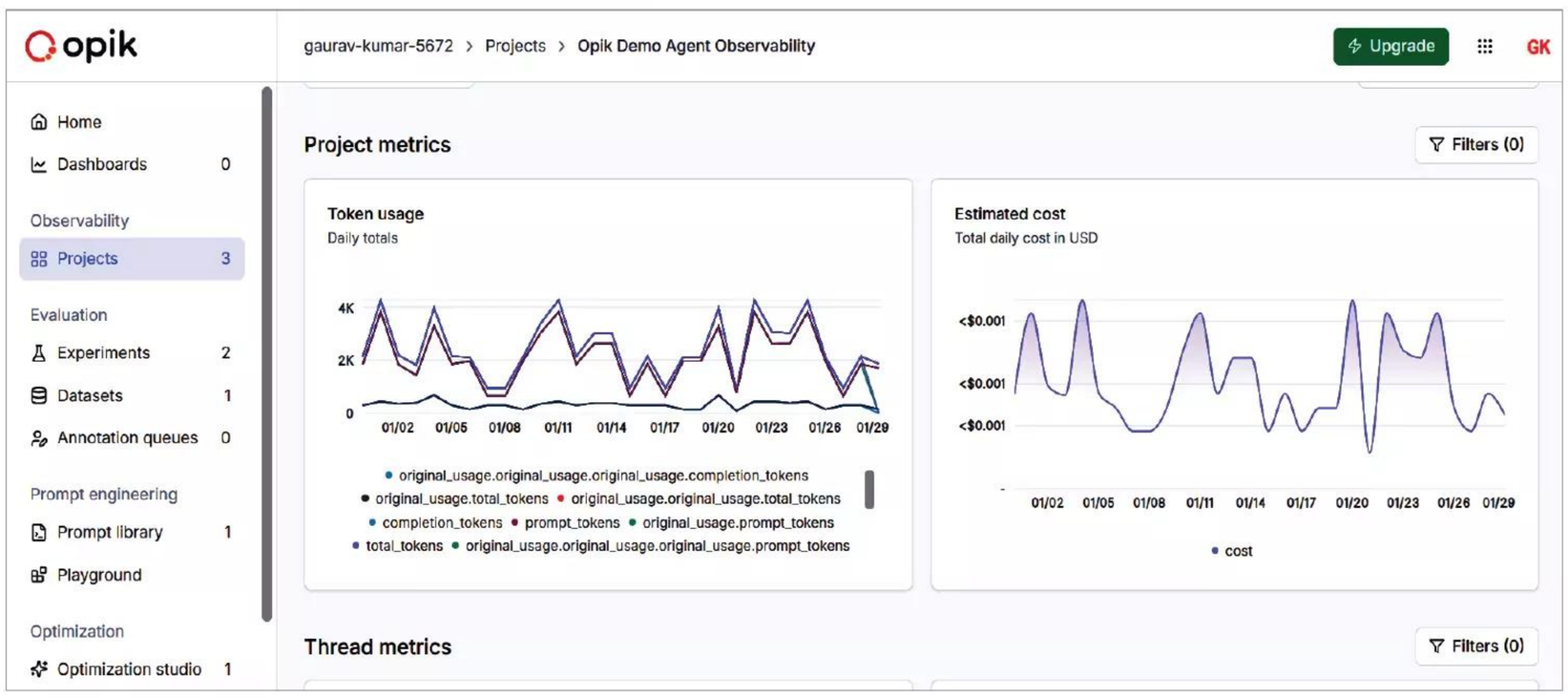


Figure 6: Project metrics and evaluations in Comet Opik

```
git clone https://github.com/comet-ml/
opik.git
cd opik
powershell -ExecutionPolicy ByPass -c
".\opik.ps1"
```

To install on Linux/Mac, use the following code:

```
git clone https://github.com/comet-ml/
opik.git
cd opik
./opik.sh

Opik URL for Access: http://
localhost:5173
```

The cloud deployment of Opik is available at *comet.com/opik*.

Key features

The Comet Opik platform offers a comprehensive suite of features for the analytics and optimisation of LLMs, including using an LLM as a judge, detailed trace visualisation, and OpenTelemetry support for robust monitoring. It enables complete experiment management through prompt logging, reproducible runs, and dataset versioning, while providing deep analytical tools for optimisation factors analysis, LLM comparisons, and metric customisation. The platform ensures rigorous evaluation via customisable workflows, response tracking, and judge scoring, and supports governance and research alignment with capabilities for bias inspection, error analysis, drift analysis, and audit trails, all with offline compatibility for secure and flexible operation.

LLM prompts can be improved using Comet Opik so that the desired results can be fetched from AI LLMs.

Often, users of AI applications give a prompt in their own natural or casual language. Such prompts can be analysed and further improved with optimisation evaluations in Opik.

Project executions on Opik can be analysed in terms of different types of graphs and plots so that the analytics can be used for further improvement or optimisation.

LLM-as-a-judge platforms can be used by researchers and practitioners who are working on resource optimisation so that AI LLMs can be developed and deployed with limited or minimal resources. END

References

- India now largest market in world in AI model adoption: BofA, *https://economictimes.indiatimes.com/tech/artificial-intelligence/india-now-largest-market-in-world-in-ai-model-adoption-bofa/articleshow/126033187.cms*
- Comet Opik, *https://www.comet.com/opik*
- Opik Open-source AI Observability, Evaluation, and Optimization, *https://github.com/comet-ml/opik*

By: Dr Gaurav Kumar

The author is associated with various academic and research institutes for delivering expert lectures and conducting technical workshops on the latest technologies and tools.

The Transformation of Prompts into Source Code

Prompts are no longer simple text. As generative AI evolves, they are turning into engineering assets, and more specifically, code that needs to be designed, tested, versioned, and maintained.



We know of prompts as intelligent instructions that can be written quickly and modified with no need for written proof. However, as generative AI advances from experimental environments to legitimate operational settings its prompts have begun to function more like software components than regular sentences. A single change in the choice of words can lead to major output differences, impacting customer experience, business choices and regulatory compliance. Software teams have discovered that prompts contain hidden dependencies, implicit logic and failure modes that engineers commonly associate with code.

This transformation is a game changer. Text no longer serves as the material for prompts because they have turned into structured components for specific purposes with operational limits and potential dangers. The prompt that directs a model to summarise legal documents and create financial summaries requires the same control that all vital system elements need.

Today, prompts function as behaviour instructions because they determine how a model thinks, what it considers important, and what it will overlook. People use prompts to express their needs while machines apply their cognitive abilities to meet these. Prompts are no longer ordinary text; they have started turning into code!

The question now is not how to write better prompts, but how to design, test, version, and maintain them—just like any other piece of software that operates at scale.

Viewing prompt engineering through a software lens

Once prompts were recognised as engineering assets, software teams started building them by applying software development methods that require designs to be structured, readable and reusable. The process of creating prompts changed from using intuition to building systems through planned design work.

This software-oriented approach requires a prompt to demonstrate its intended function through its operational guidelines. Casual prompts allowed certain ambiguous elements but now all unclear components have become unacceptable. The major advantage of these prompts is that they produce written material which teams can utilise to review their work and make improvements through collaborative processes.

Today, engineers create prompt designs that establish specific word choices and build multiple boundaries to direct how the model will think. Software teams maintain control over their creative work through systematic procedures, which enable them to produce work that matches expectations.

A simple example illustrates this shift:

```
BASE_INSTRUCTIONS = """
You are an AI analyst.
Follow instructions precisely and use neutral language.
"""

TASK_PROMPT = """
Analyze the input document and produce:
1. A concise summary (max 5 bullet points)
2. Key risks, categorized as legal, financial, or operational
3. Any missing or unclear information
"""

final_prompt = BASE_INSTRUCTIONS + TASK_PROMPT
```

Table 1: Version control for thinking machines

Aspect	Without prompt versioning	With prompt versioning (code-like approach)
Change tracking	Changes are informal and undocumented	Every modification is recorded with history
Debugging	Hard to identify why outputs changed	Easy to trace issues to a specific prompt version
Reliability	Model behaviour shifts unpredictably	Outputs remain consistent and reproducible
Collaboration	Prompts live in chats or personal notes	Teams collaborate using shared repositories
Experimentation	Risky trial-and-error in production	Safe branching and controlled testing
Rollback	Previous working prompts are lost	Instant rollback to stable versions
Audit and compliance	No traceability for decisions	Full accountability with prompt lineage

Testing prompts before they fail at scale

The prompt quality assessment methods that were used initially relied on people's intuition to determine if the output matched acceptable standards. This approach fails when prompts are used in production environments. The operational needs of prompts require them to function accurately through testing. This involves all types of user inputs, testing their limits, and tracking user interaction changes. Systematic testing is now a necessity.

Testing prompts serves two purposes -- verifying a prompt's response and observing how users interact with it. The testing process reveals all the hidden problems that originate from small modifications to the instructions or the addition of an example.

Today's teams use established testing methods from software engineering to test their prompts. Unit tests assess how well a prompt processes input types. Regression tests ensure that enhancements made to one aspect of the system do not result in a fall in performance in other parts of the system. Prompt quality assessment uses golden datasets that contain pre-selected input samples and their corresponding expected results to establish evaluation standards.

In prompt testing, assessment criteria are as important as evaluation criteria. The testing process examines prompts for their ability to produce correct results while evaluating their structural integrity, completeness, potential biases, and safety features. Human assessment works with automated systems to provide better evaluation results in situations that carry high risks. Multiple testing sessions lead to a fact-based method for developing prompts.

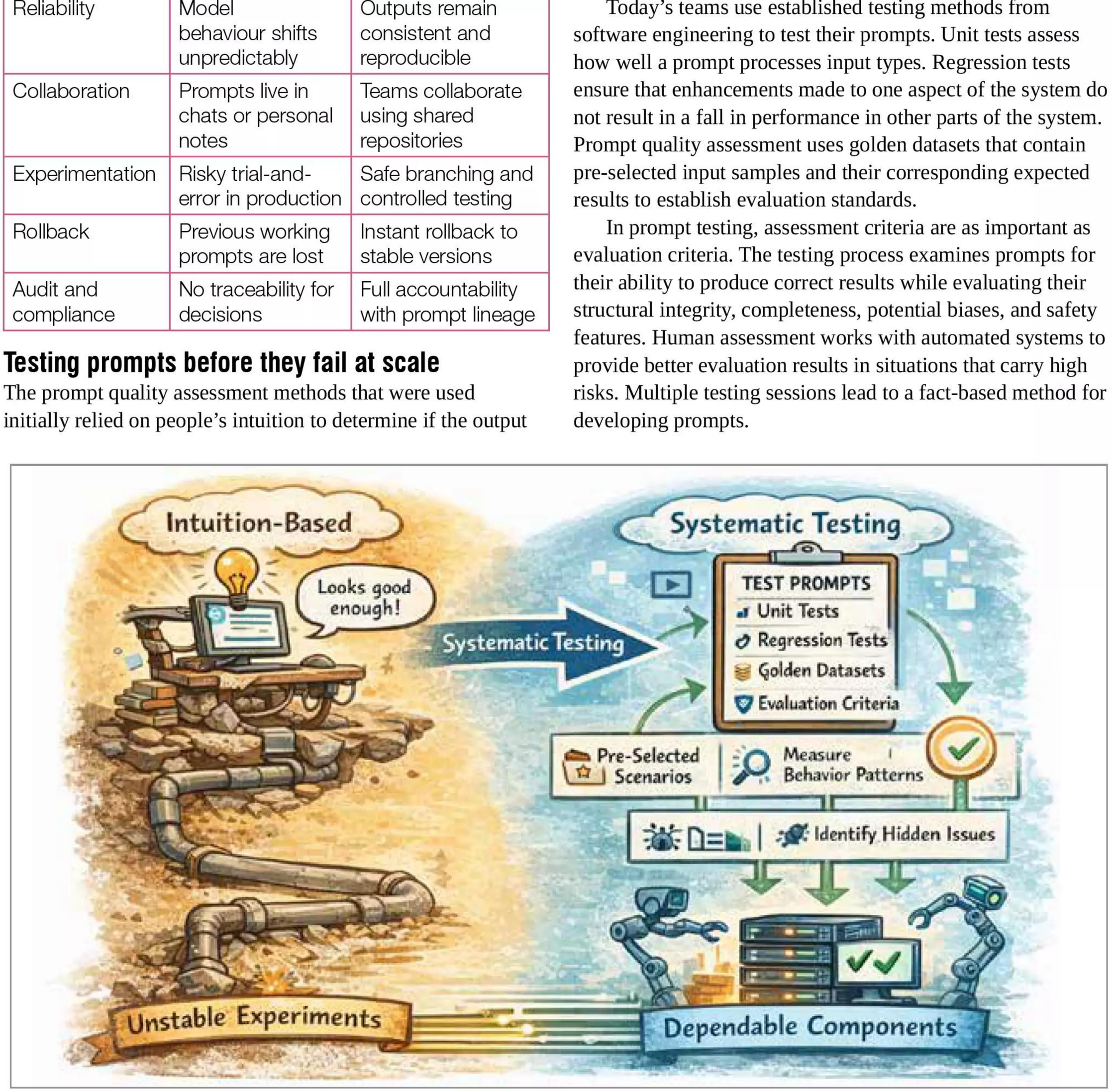


Figure 1: The benefits of testing prompts

Table 2: The prompt engineering toolchain

Stage in prompt lifecycle	Purpose	Typical tools / Practices	Engineering benefit
Prompt authoring	Design clear, structured prompts	Templates, YAML/JSON formats, prompt libraries	Consistency and readability
Version control	Track prompt changes over time	Git, DVC, prompt repositories	Traceability and rollback
Experimentation	Test alternative prompt designs	Prompt branching, A/B testing setups	Safe innovation without risk
Evaluation and testing	Measure prompt behaviour	Golden datasets, automated evaluators, human review	Reliability and quality assurance
Optimisation	Improve cost, latency, and accuracy	Token analysis, prompt compression techniques	Efficient and scalable AI usage
Deployment	Integrate prompts into applications	CI/CD pipelines, configuration files	Controlled and repeatable releases
Monitoring	Detect performance drift	Logging, feedback loops, analytics dashboards	Early detection of failures

Organisations that test their prompts before launching them into operation improve public confidence and trustworthiness towards their AI systems. The testing process changes prompts from unstable experimental approaches into dependable operational elements.

Optimisation as a discipline

Once prompt testing is completed, optimisation work can begin. In the initial phase of prompt tuning, intuition is used for determining which instructions, examples and restrictions must be added until the output reaches an acceptable standard. The temporary solution functions properly, but it fails to handle larger operations.

Prompt optimisation extends beyond achieving correct results. Production systems require prompts to determine how much time they take to respond, which tokens they consume, which expenses they incur, and which results they produce. The use of a lengthy prompt will result in better output quality. The use of a brief prompt will decrease expenses. Optimisation seeks to find the optimal solution that applies to the context of the model.

The assessment process requires evaluation through metrics. Teams assess output quality scores, failure rates, token consumption, and response variability for different prompt versions. The performance of the system improves through small structural changes like reordering instructions, tightening constraints, and removing unnecessary examples.

Optimisation leads to better results but does not establish one optimal prompt. A prompt designed for internal analysis needs different design elements than a prompt that serves customer-facing interactions. The scheduling of optimisation as a permanent practice enables prompt development to progress with model enhancements, workload changes, and business objectives.

Organisations develop a consistent prompt engineering process by implementing measurement and iterative testing as their foundation for prompt optimisation work. The process leads to AI systems that function efficiently and produce dependable outcomes.

PromptOps: The evolution of prompt engineering

PromptOps enables AI systems to function with operationalised prompts that act as ongoing elements just like DevOps enables software development teams to build and maintain their products. The system establishes a link between prompt testing and operational deployment while it maintains prompt performance throughout its lifecycle.

PromptOps unites all the elements of prompt engineering, which include versioning, testing, optimisation, deployment, and monitoring, into one complete workflow. Teams handle prompts as dynamic assets that they can modify to fit their changing requirements. The organisation builds systematic processes for testing and implementing changes, which help to maintain system performance during production.

This new system establishes a new way for teams to work together. Data scientists, engineers, product teams, and domain experts all participate in the process of designing and assessing prompts, making decisions based on shared metrics and tools.

PromptOps considers intelligence in modern AI systems extends beyond their underlying models. Organisations that adopt it early and excel at it will gain a market advantage. END

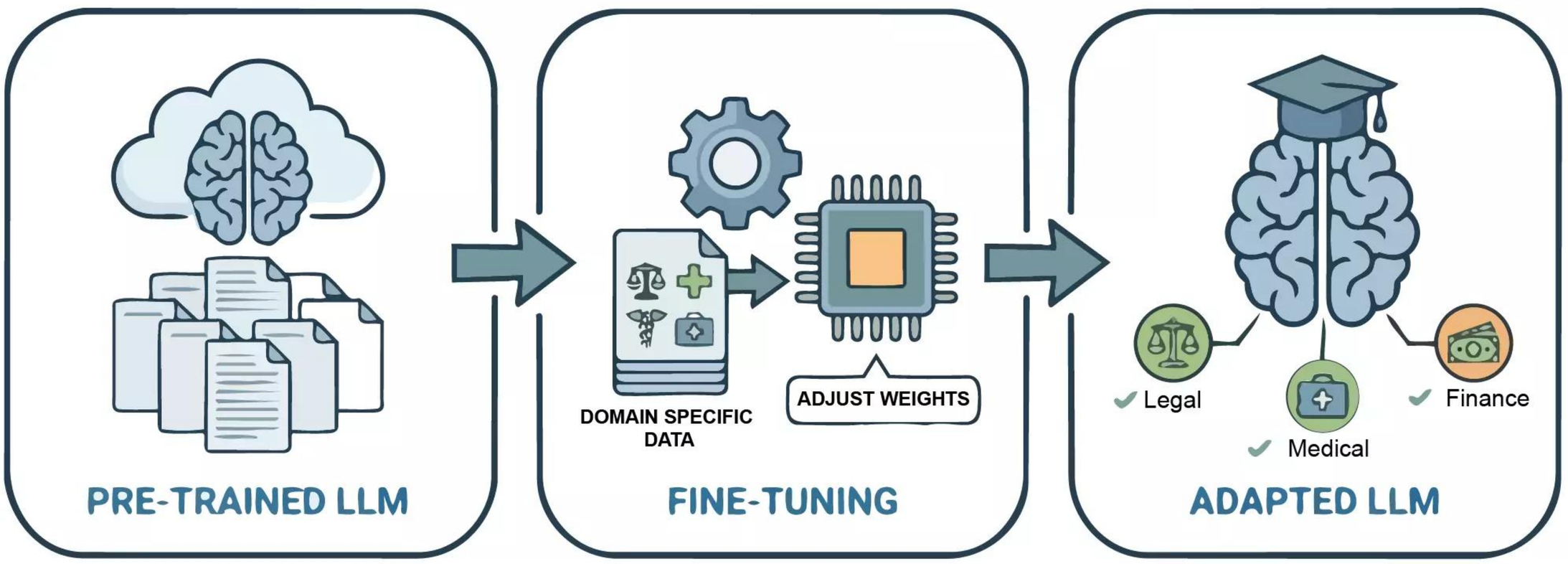
By: D. Banerjee and Sourav K.

D. Banerjee is an IT professional with 15+ years of experience in Tier-1 IT companies, specialising in delivering scalable enterprise solutions and leading complex technology initiatives.

Sourav K. is an IT professional with around 5 years of experience in Tier-1 IT companies, skilled in modern technologies and focused on building efficient, data-driven solutions.

How to Fine-Tune LLMs for Domain-Specific Adaptation

In this first part of a two-part series, discover why we need to fine-tune large language models, and get introduced to the core concepts and techniques used for their domain-specific adaptation.



Large language models (LLMs) are deep neural networks, typically based on the Transformer architecture, trained on massive text corpora to predict the next token in a sequence. These models can generate text, write code, summarise documents, and answer complex questions. However, a general-purpose LLM cannot automatically perform well in specialised domains such as finance, healthcare, legal compliance, or enterprise-specific tasks. This limitation creates the need for LLM adaptation, where a pre-trained model is modified to behave correctly for domain-specific use cases.

Approaches to LLM adaptation

LLMs often need adaptation for two main reasons. The first is the limitations of base models -- pre-trained models possess generic knowledge and may produce inaccurate or contextually irrelevant responses (i.e., prone to hallucinations).
The second reason is the need for customisation, as enterprises require outputs that follow internal standards — specific tone, formats, templates, and regulatory language.

There are three main approaches to adapting an LLM: prompt engineering, RAG, and fine-tuning.

Prompt engineering: Prompt engineering adjusts the model's behaviour through carefully crafted instructions without changing the model itself. It is best suited for quick behavioural adjustments in situations where the base model already has sufficient domain knowledge.

RAG (retrieval-augmented generation): RAG enriches the model's responses by retrieving relevant external documents and injecting them into the prompt at inference time. This enables the model to generate knowledge-based answers using fresh or domain-specific information without requiring retraining.

Fine-tuning*:* Fine-tuning updates the model's parameters using domain-specific datasets, so it learns terminology, style, and task patterns internally. It is used for deep domain expertise or specialised tasks where consistency and precision are essential.

What is fine-tuning?

Fine-tuning involves taking a pre-trained LLM and further training it on a focused, domain-specific dataset. During fine-tuning, some or all the

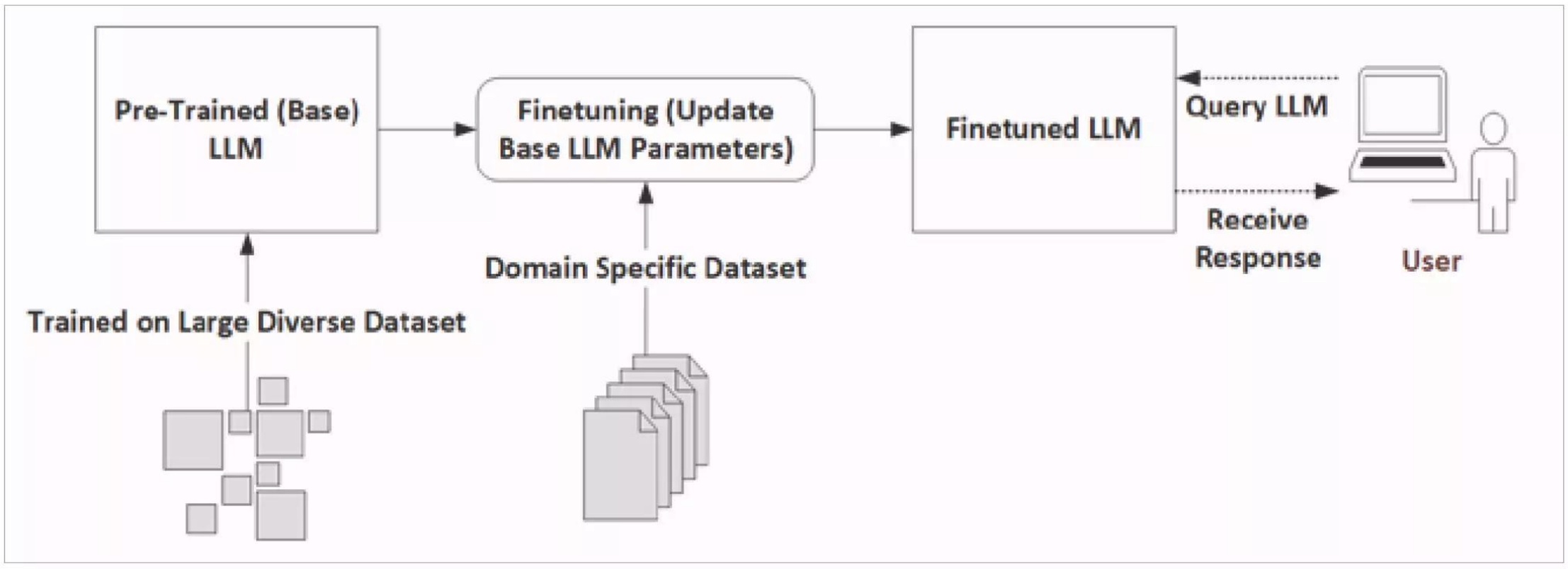


Figure 1: LLM fine-tuning

model's parameters are adjusted so it can learn the domain's terminology, style, and reasoning patterns. This focused training helps the model go beyond generic knowledge and develop a more expert understanding of how information is expressed and interpreted in that field.

Fine-tuning is required over prompt engineering and RAG as:

- Prompt engineering can guide a model but cannot force it to learn domain terminology or internal workflows it has never seen.
- RAG augments the model with retrieved documents, but it doesn't change model behaviour.
- Prompts and retrieved documents cannot guarantee deterministic and compliant behaviour. Fine-tuning allows embedding organisational compliance directly into the model parameters.

Types of fine-tuning

Based on the training objective and kind of data used, fine tuning can be categorised as unsupervised fine-tuning, supervised fine-tuning (SFT) and instruction fine-tuning (IFT).

Unsupervised fine-tuning: Unsupervised fine-tuning involves training the model further on raw, unlabelled text from a specific domain. It does not teach the model specific tasks; instead, it strengthens domain understanding, reduces hallucinations, and improves fluency.

Example use cases are:

- Adapting a general LLM to understand medical literature.
- Training on legal contracts to learn clause structure, legal terminology, and formal writing style.

Supervised fine-tuning (SFT): The goal of SFT is to teach the model how to perform a specific task using labelled input-output pairs. Each training example consists of an input prompt and a desired response. The focus is on learning patterns, formats, and decision logic for a particular workflow.

Unlike unsupervised fine-tuning, SFT explicitly guides the model towards correct behaviour and output generation for well-defined workflows.

Example use cases are:

- Training a code-generation model using problem–solution pairs.
- Adapting a model for news classification.

Instruction fine-tuning (IFT): Instruction fine-tuning (IFT) is a specialised form of supervised fine-tuning in which the input is an explicit natural language instruction, followed by the expected response. The goal of IFT is to train the model to understand and follow human instructions reliably across a wide range of tasks.

IFT improves conversational ability, task compliance, and reasoning behaviour, making it a key technique behind modern chat-based LLMs. Popular open source models such as LLaMA-Instruct and Mistral-Instruct rely heavily on IFT. IFT is typically the final fine-tuning stage before deploying an LLM for interactive applications.

Example use cases are:

- Training a chatbot to respond in a specific tone or writing style.
- Training the model to respond safely and refuse invalid or harmful requests.

Techniques of fine-tuning

Fine-tuning methods can be divided into two broad technical approaches depending on how much of the model is updated and the computational cost involved. These approaches are full fine-tuning and parameter-efficient fine-tuning (PEFT).

Full fine-tuning: Full fine-tuning involves updating all the parameters of the pre-trained LLM during training. The entire LLM (billions of parameters) is unfrozen and learns the new domain end-to-end. It provides the highest level of adaptation and can significantly change model behaviour. It requires large compute resources and substantial training data. However, it can carry risks of overfitting and

catastrophic forgetting.

Full fine-tuning can be used:

- When the domain is very different from the general knowledge of the model (e.g., biomedical, aerospace domains).
- When maximum accuracy and deep adaptation are needed.
- When compute resources and large training datasets are available.

Note:

- *Overfitting:* The model may memorise training examples instead of learning general patterns if the dataset is too narrow/small. The model performs perfectly on the training examples but loses its ability to handle new and unseen data.
- *Catastrophic forgetting:* The model may lose general capabilities because new training overwrites pre-training knowledge.

Parameter-efficient fine-tuning (PEFT): Parameter-efficient fine-tuning (PEFT) updates only a small subset of the model's parameters while keeping the base pre-trained model frozen. This approach significantly reduces training costs, GPU memory usage, and the overall time required for fine-tuning.

Instead of modifying billions of parameters, PEFT techniques add lightweight trainable components, such as low-rank matrices, adapter layers, or learned prompts to the existing model. These components learn task-specific behaviour while preserving the general knowledge already present in the base model. As a result, PEFT makes fine-tuning practical even on limited hardware and is widely adopted in the open source community.

PEFT can be used:

- When compute resources are limited (single GPU or consumer-grade hardware).
- When multiple task-specific models need to be derived from a single base LLM.
- When fast experimentation and iteration are required.
- When preserving the base model's general knowledge is important.

Some commonly used PEFT techniques include LoRA (Low-Rank Adaptation) and QLoRA (Quantized LoRA).

LoRA (Low-Rank Adaptation): LoRA introduces small trainable low-rank matrices into selected Transformer layers. During training, only these added matrices are updated, while the original model weights remain frozen. This greatly reduces the number of trainable parameters without significantly affecting model performance. Use LoRA when you need general enterprise fine-tuning with moderate domain data and want high accuracy at low training cost.

This is how LoRA works internally:

- During fine-tuning, the base model weights remain frozen (W_0), and LoRA learns only a small task-specific update (ΔW). The updated weight used by the model is: $W = W_0 + \Delta W$
- Instead of learning a large d × d update matrix, LoRA represents the update in a compact low-rank form: $\Delta W = BA$

Here:

d = the model's hidden size (its internal feature dimension)

A = a small matrix with r rows and d columns

B = a small matrix with d rows and r columns

r = the low rank (a small number compared to d; typically, 4, 8, or 16)

- This reduces trainable parameters from d^2 to 2dr, making training faster, cheaper, and more memory-efficient while maintaining strong accuracy.

QLoRA: QLoRA extends LoRA by loading the base model in 4-bit quantized precision, which further reduces GPU memory usage. The LoRA components remain in higher precision and are trained normally. This combination enables fine-tuning of very large models (30B-70B parameters) on a single GPU. Use QLoRA when training large models (13B-70B) on limited hardware; 4-bit quantization reduces memory while preserving model quality.

Note:

- *Low-rank matrix:* A compact mathematical representation that approximates large weight updates using much smaller matrices, reducing memory and computation.
- *Quantization (4-bit):* A technique that stores model weights using fewer bits (4 instead of 16 or 32), lowering memory usage with minimal loss in accuracy.

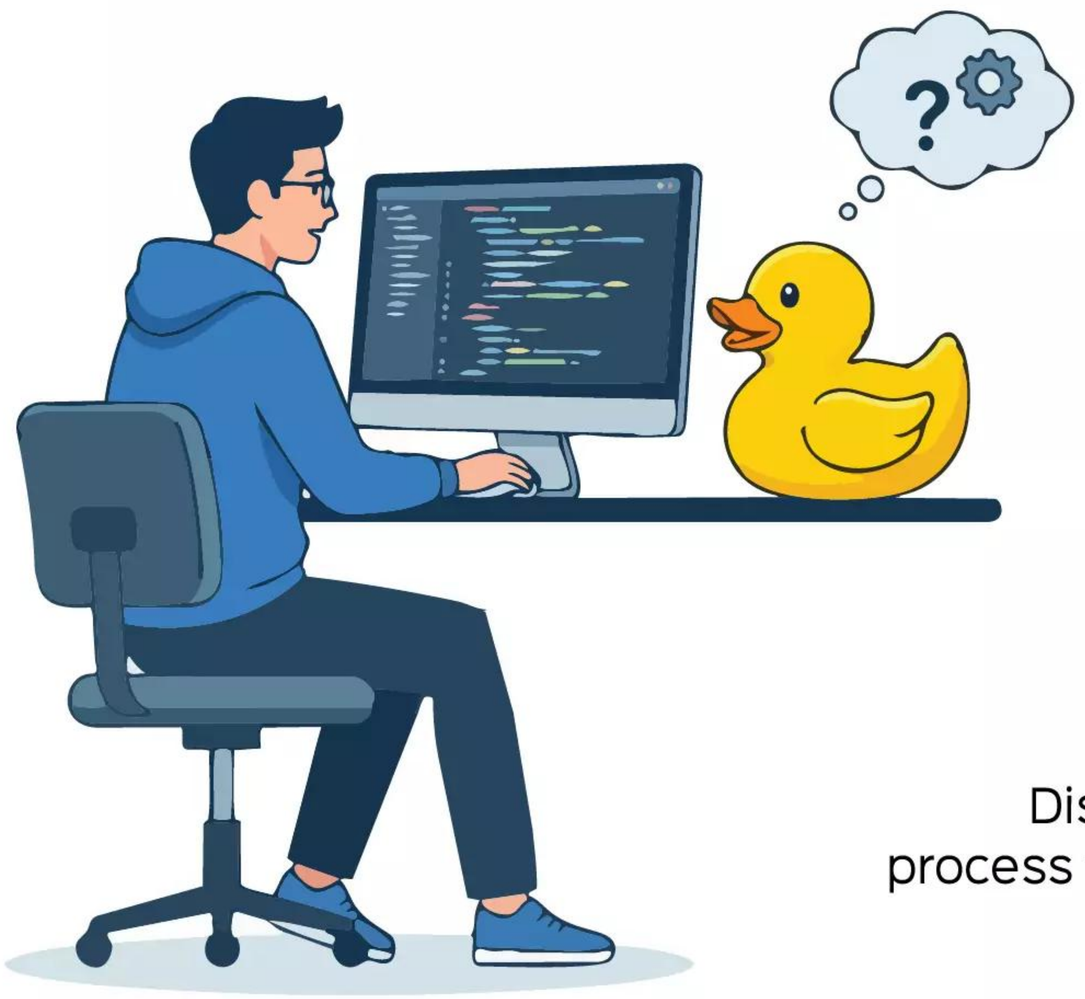
This first article in the two-part series has established the foundation for understanding how and why large language models must be adapted for real-world, domain-specific use cases. With this conceptual grounding, the next part will demonstrate a complete hands-on workflow for fine-tuning an open source LLM (Mistral-7B-Instruct) using PEFT QLoRA in Google Colab environment. END

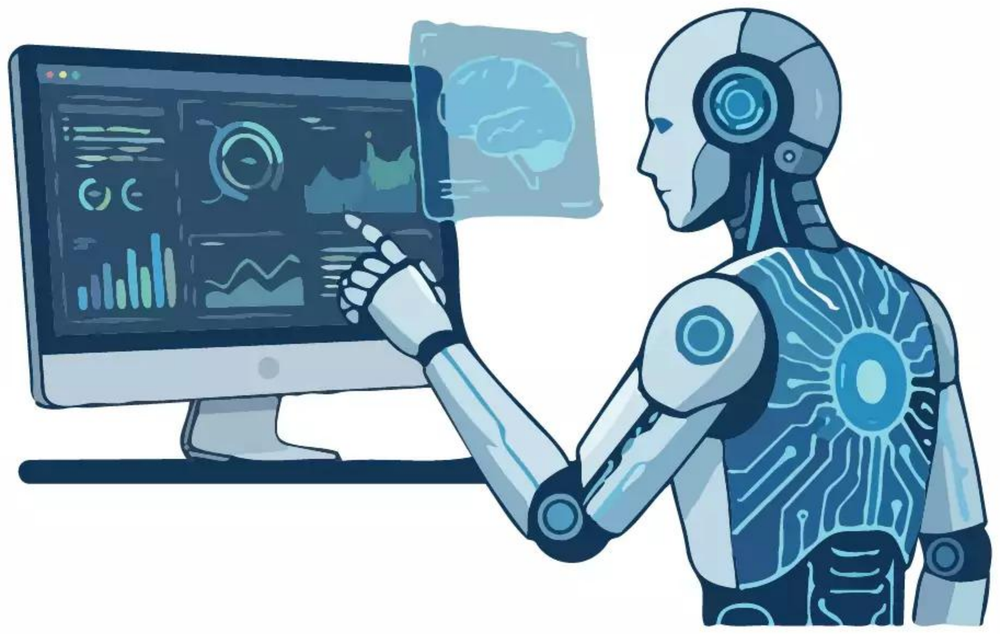
By: Subranshu Patel and Shwetabh Srivastava

Subranshu Patel is a group manager in the system software division of HCL Technologies Limited. He has extensive experience in Linux-based OS and ecosystem design and development, and in open source technologies.

Shwetabh Srivastava is a solutions architect at HCL Technologies Limited. He is experienced in designing and developing Linux-based system software, with expertise in operating system principles and the inner workings of the Linux kernel.

The Relevance of Rubber Duck Debugging in the Age of AI





Discover why rubber duck debugging is a powerful process today. There's also a step-by-step guide on how to use it in the age of artificial intelligence.

In an era of sophisticated AI coding assistants, the humble practice of rubber duck debugging is gaining new relevance. While an AI can instantly generate code or identify syntax errors, it often lacks the nuanced context of the programmer's original intent. Thus, the 'duck' evolves from a simple passive listener into an active, querying partner; the core magic lies not in receiving an answer, but in the act of formulating the question with such clarity that the solution often reveals itself in the process.

Every engineer has experienced this moment.

You are stuck on a problem for hours. The logs look fine. The logic seems correct. You run the program again and again. Still nothing changes. Finally, you turn to a colleague and begin explaining:

"See, first this function validates the input. Then it calls the API. After that—"

And suddenly you pause.

"Wait… that API call can fail before validation completes."

The bug was not hiding in the code. It was hiding in your thinking. This phenomenon has a name: rubber duck debugging.

In today's world of AI-powered tools, this simple technique has become even more powerful — but only when used correctly.

Where did rubber duck debugging come from?

The term became popular after the 1999 book 'The Pragmatic Programmer' by Andrew Hunt and David Thomas. In the book, the authors describe a programmer who keeps a rubber duck on his desk. Whenever he gets stuck, he explains his code line by line to the duck. Quite often, the solution appears during the explanation itself.

But this behaviour existed long before the book. Developers would figure out the problem or the bug while:

- Writing a detailed email about a production issue — and solving it while writing;
- Explaining logic on a whiteboard — and spotting the flaw mid-sentence;
- Discussing a bug during a stand-up meeting — and realising the mistake while speaking.

The duck is only symbolic. The real idea is simple: When you explain clearly, you think clearly.

Why does this technique work?

Software development is not just typing code. It is structured thinking. Rubber duck debugging works because it improves that thinking.

It forces sequential thinking: Inside our mind, logic feels complete. But when we explain it, we must go step by step:

- What is the input?
- What happens next?
- Why is this condition needed?
- What happens if it fails?

Many developers mentally skip transitions. While explaining, those hidden gaps become visible.

It exposes assumptions: Many real-world bugs are based on assumptions:

- This value will never be null.
- The API always returns valid JSON.

- Users will never enter negative values.

When you say these statements out loud, they suddenly sound risky. Explanation makes hidden assumptions visible.

It improves problem understanding: Sometimes the issue is not in the code — it is in misunderstanding the requirement.

By clearly explaining expected behaviour, you may discover:

- The requirement was incomplete.
- Edge cases were not considered.
- Failure scenarios were ignored.

Clarity in understanding significantly reduces debugging time.

From rubber duck to AI

A physical rubber duck listens silently. AI, however, can:

- Ask clarifying questions.
- Reflect your logic back to you.
- Point out missing scenarios.
- Simulate edge cases.

But there is an important difference. If you use AI only to generate solutions, you reduce your own growth. If you use AI to test and refine your thinking, you become stronger. The goal is not faster answers. The goal is clearer reasoning.

How to use AI as a rubber duck

Here is a structured approach suitable for students, working engineers, and system architects.

Step 1: Define the role clearly: Before sharing code, guide the AI.

Prompt example: Act as a rubber duck debugger. Do not immediately give me a solution. Listen to my explanation and point out unclear logic, missing steps, or wrong assumptions.

This keeps the focus on reasoning rather than quick fixes.

Step 2: Explain the intended behaviour (not the code): Start with what the system should do.

Prompt example: I am writing a function to calculate discounts.

> Premium users get 20%.
> Regular users get 10% if purchase is above ₹5000.
> Others get no discount.

Immediately, important questions arise:

- What if the purchase is exactly ₹5000?
- What if the amount is negative?
- Can a user be both premium and regular?
- What if the user type is missing?

Clarity begins before the first line of code is written.

Step 3: Walk through the logic sequentially: Now explain your implementation step by step.

Prompt example: Here is my step-by-step logic:

1. Check if the user is premium.
2. If yes, apply 20%.
3. Else check if purchase > 5000.
4. Apply 10%.
5. Return final amount.

Please restate this and tell me if any case is not handled.

AI may highlight:

- Missing boundary condition (exactly 5000).
- Lack of validation for negative input.
- Overlapping conditions.

This improves completeness and robustness.

Step 4: Identify assumptions: This is the most critical stage.

Prompt example: What assumptions am I making here? Under what real-world situations can this logic fail?

Step 5: Explore edge cases: Most production failures occur in rare situations.

Prompt example: List possible edge cases for this logic.

This step prepares your system for real-world unpredictability.

Step 6: Request improvement: This is the final step once there is clarity and reasoning is clear.

Prompt example: Based on our discussion, suggest a clean and safe implementation.

In this discussion AI supports your thinking instead of just replacing it.

Technology keeps evolving. AI tools are becoming more powerful every year. But one thing remains constant — software quality depends on clarity of thought.

Whether you use a physical duck, a colleague, a whiteboard, or an AI tool — the principle remains the same. Clear explanation leads to clear thinking. Clear thinking leads to better engineering. END

By: Anil Dhaneshwar

The author is a senior principal software engineer in InfoScale at Pune, Maharashtra. He has expertise in the Linux kernel, C, storage and system-level development.

THE COMPLETE MAGAZINE ON OPEN SOURCE
OpenSource ForYou
The latest from the Open Source world is here.
OpenSourceForU.com
Join the community at facebook.com/opensourceforu
Follow us on Twitter @OpenSourceForU

Why Open Source Large Language Models are Popular

Open source large language models mark a pivotal moment in the evolution of generative AI. By lowering barriers to entry and fostering collaborative innovation, these models are enabling a broader spectrum of organisations to benefit from AI.



Generative artificial intelligence has ushered in a new era of innovation, transforming the way machines understand and generate human language. At the heart of this revolution are large language models (LLMs), which have become indispensable for tasks ranging from text generation to complex reasoning. While commercial LLMs have garnered significant attention, the emergence of open source alternatives has democratised access to advanced AI capabilities. These models, developed collaboratively by global communities, empower organisations and individuals alike to harness, adapt, and refine generative AI for a multitude of use cases.

Popular open source large language models

Several open source LLMs have risen to prominence, each offering unique features, architectures, and advantages. The Llama series, developed by Meta, exemplifies efficient and scalable language modelling, attracting widespread adoption in research and industry. EleutherAI's GPT-Neo and GPT-J initiatives have successfully replicated and extended the capabilities of earlier transformer models, providing robust alternatives with permissive licences. Falcon, another noteworthy entrant, focuses on optimising inference speed and resource efficiency, making it suitable for deployment in cost-sensitive environments. Models such as BLOOM and MPT have further expanded the ecosystem, emphasising multilingual support and modular architectures. The collaborative development and transparent release of these models have fuelled rapid innovation and adaptation across the AI landscape.

The benefits: The adoption of open source LLMs confers several significant benefits. Foremost is the flexibility to customise and fine-tune the models according to specific organisational needs, which is often constrained in commercial alternatives. Cost efficiency is another major advantage, as the absence of licensing fees lowers barriers for startups, research institutions, and enterprises. Transparency is inherent to open source projects, allowing practitioners to scrutinise model architectures, training data, and underlying assumptions. This openness cultivates trust and facilitates the identification and mitigation of biases. Additionally, the vibrant global community supporting open source LLMs accelerates knowledge sharing, troubleshooting, and collective improvement, ensuring that the technology evolves rapidly and responsibly.

Common use cases of genAI

Open source generative AI models find application in a diverse array of industries, each benefiting from tailored solutions. In healthcare, LLMs assist in medical documentation, summarisation of patient records, and even support clinical decision-making with evidence-based suggestions. The finance sector leverages these models for automating customer service, generating financial

reports, and detecting fraudulent transactions by recognising linguistic anomalies. Education technology platforms employ LLMs for personalised tutoring, content generation, and automated grading, thereby enhancing both teaching effectiveness and student engagement. Beyond these, manufacturing, legal services, and creative industries all exploit the adaptability of open source LLMs to streamline processes, generate insights, and unlock new business opportunities.

Open source vs commercial LLMs: A comparison

A fundamental distinction between open source and commercial LLMs lies in accessibility. Open source models are freely available for download, modification, and deployment, whereas commercial offerings often impose usage restrictions and recurring costs. This openness translates to greater customisation potential, enabling organisations to tailor models to domain-specific vocabularies or compliance requirements. In contrast, commercial LLMs typically provide limited transparency regarding their internal workings and training data, raising concerns about explainability and potential biases. However, commercial models may offer dedicated support, regular updates, and optimised performance out of the box. Ultimately, the choice between open source and commercial LLMs hinges on an organisation's priorities—whether they value autonomy and adaptability or prioritise convenience and vendor support (see Table 1).

Fine-tuning opensource LLMs: Steps and resources

Fine-tuning is the process of adapting a pre-trained open source LLM to excel in a particular task or domain. The journey begins with data collection, where practitioners curate high-quality, representative datasets relevant to their objectives. The next step involves preprocessing and formatting this data to align with the model's input requirements. Fine-tuning is conducted using frameworks such as Hugging Face Transformers or PyTorch, leveraging robust computational resources—often GPUs or TPUs—to update the model's weights. It is crucial to monitor training metrics, adjust hyperparameters, and apply early stopping to prevent overfitting. Upon completion, the fine-tuned model should undergo rigorous evaluation against validation datasets to ensure it meets performance standards. Documentation and version control throughout the process are essential for reproducibility and future enhancements.

Validating reliability and ensuring data protection

Reliability and data protection are paramount when deploying open source LLMs in sensitive environments. To validate reliability, practitioners should employ comprehensive evaluation protocols, including cross-validation, adversarial testing, and benchmarking against established datasets. Regular audits help detect and rectify unintended

Table 1: A comparison of open source and commercial LLMs

Aspect	Open source LLM	Commercial LLM
Accessibility	Freely available for download and local deployment, with the flexibility to modify and adapt as required.	Usually accessible through paid subscriptions or APIs, often with restrictions on usage and modification.
Customisation	Can be extensively customised to suit specific needs, including domain-specific vocabulary and compliance requirements.	Customisation options are generally limited, with most models offering standardised performance out of the box.
Transparency	Source code, model architecture, and training data are typically open for review, promoting explainability and trust.	Internal workings and training datasets are often proprietary, making it difficult to assess or address potential biases.
Support	Relies on community-driven forums and documentation for troubleshooting and updates.	Usually backed by dedicated customer support, regular updates, and performance optimisations by the vendor.
Cost	Minimal to no licensing fees, lowering the barrier for adoption by startups, academics, and enterprises.	Involves recurring costs, such as subscription fees or pay-per-use charges, which can increase with scale.
Innovation pace	Benefits from rapid advancements and knowledge sharing within a global open source community.	Improvements are governed by the provider's release cycles and commercial priorities.
Security and compliance	Allows for greater control over data handling and compliance with specific regulatory requirements.	May offer built-in compliance features, but users have limited visibility or control over internal processes.

behaviours or biases. Data protection mandates that all training and inference processes adhere to applicable privacy laws and organisational policies. Employing techniques such as differential privacy, data anonymisation, and secure multi-party computation can mitigate risks associated with handling confidential information. Best practices also include minimising the retention of sensitive data, encrypting data at rest and in transit, and maintaining detailed access logs. By embedding these safeguards into the development lifecycle, organisations can foster trust and ensure ethical deployment of generative AI.

Handling hallucinations in open source LLMs

Hallucinations—outputs that are apparently fluent but false or not connected to reality—remain one of the most stubborn obstacles for open source LLMs to be used in production in IT, finance, healthcare, and legal processes. In the open source context, the challenge is becoming more formidable due to heterogeneity in datasets, fine-tuning methodologies, and inference engine. Thus, mitigating hallucination necessitates a multi-layer solution from data, model alignment, retrieval grounding, decoding control and post-generation validation.

A good start is grounding. Retrieval-augmented generation (RAG) mitigates fabrication by making the model condition its answers on authoritative sources (policies, manuals, knowledge bases). Modern RAG implementations should not be limited to naive 'top-k' retrieval: use hybrid search (dense + BM25), rerankers (cross-encoders), document chunking fine-tuned to domain structure, and citation-aware prompting such that the response can be tied to retrieved passages. For business precision, add freshness controls (time-based filters) and access control aware retrieval so the model doesn't cite content the user shouldn't see.

Next, 'align the model's behaviour'. Instruction tuning and preference optimisation (e.g. DPO/IPO replacing classical RLHF pipelines) can diminish overconfident guessing by incentivising behaviours like "I don't know," and "there is not enough evidence." Where domain accuracy is important, employ domain-adaptive fine-tuning along with counterfactual/hard-negative training (penalises answers that are similar but wrong). It's also useful to train with structured output constraints (JSON schema/function-calling style) so the model has to distinguish claims, evidence and uncertainty.

Hallucinations are also known to be correlated with sampling aggressiveness during inference time. Perform conservative decoding (decreasing temperature, using nucleus sampling bounds) and introduce logit biasing/refusal policies for unsafe or unverifiable questions. For tool-using agents, prefer tool-first patterns: the model 'retrieves', computes, or queries systems of record (CMDB, ticketing, inventory) rather than 'memorising' facts. Introduce self-consistency checks only when evidence-constrained (unbounded 'chain-of-thought' style deliberation can increase confident nonsense if retrieval is weak).

Table 2: A comparison of leading open source LLMs

Model	Key features	Advantages	Common use cases	Industry adoption
Llama (Meta)	Efficient transformer architecture, scalable sizes (7B to 65B parameters), multilingual capabilities	High performance on limited resources, flexible licensing, active community	Chatbots, research, content generation, code assistance	Technology, research, academia, startups
GPT-Neo/GPT-J (EleutherAI)	Open weights, transformer-based, compatible with GPT-3 benchmarks	Transparency, no usage restrictions, strong text generation	Automated writing, summarisation, language modelling	Media, education, research
Falcon	Optimised for inference speed and efficiency, open weights	Reduced operational costs, ease of deployment	Real-time applications, virtual assistants, customer service automation	Customer service, retail, SMEs
BLOOM	Multilingual, trained on diverse datasets, collaborative development	Wide language support, community-driven updates	Cross-lingual tasks, translation, social media analysis	Global enterprises, NGOs, research
MPT	Modular transformer, flexible context window, open licensing	Customisable, easy integration, suitable for long documents	Document processing, legal tech, knowledge management	Legal, compliance, enterprise

Now add verification loops. Combine an LLM 'judge' with deterministic checks: schema validation, unit conversions, rule engines, and fact verification against sources. Monitor hallucination at the operational level with groundedness metrics, citation precision/recall, and acceptance tests per task. Red-team with adversarial prompts and prompt-injection tests (especially in RAG) to stop 'retrieval hijacking'. In real systems, hallucination management is not a 'set it and forget it' affair – it is an observability issue that uses prompts, retrieved contexts, model versions and outputs so that failures can be reproduced, measured, and systematically reduced.

Key considerations for choosing the right LLM

Choosing an open source LLM is an engineering decision across its ability, cost, deployment constraints, and governance. Begin with the workload archetype: conversational support, document understanding, code generation, extraction into structured fields, summarisation, or agentic tool use. Model selection should be based on quantifiable requirements such as minimum accuracy, latency, context length, multilingual support, and acceptable risk, not just on the number of parameters.

Model capability and fit: Test models on domain-specific benchmarks and internal datasets. Overall benchmarks are nice, but the real success to production are your own distributions: ticket text, clinical notes, finance narratives, or policy documents. Evaluate instruction following, long-context performance (including deterioration patterns), and tool-use reliability. For enterprise workflows, choose models that have high structured output fidelity and stable refusal/uncertainty behaviours.

Licensing and compliance: Open source is not always 'free and unrestricted'. Check the licence for rights to use commercially, redistribute, and derivatives finetunes. Verify data handling considerations -- for PII/PHI, favour on-prem or VPC deployment, encryption, access controls and audit logging. Ensure you can address retention, residency and model governance (versioning, reproducibility, rollback).

Compute, latency, and total cost of ownership: Tune inference — GPU memory, throughput (tokens/sec), concurrency, and peak loads. Quantization is important: for example, there are additional gains (8bit/4bit) from newer serving stacks (vLLM/TGI-like runtimes) that can significantly reduce the cost, but may reduce quality for reasoning or multilingual workloads. Account for context window cost: long-context models raise memory and latency; RAG may be cheaper than 'pressing the prompt'. Determine if you need full fine-tuning, parameter-efficient tuning (LoRA/QLoRA), or prompt/RAG-only adaptation. In rapid evolving domains, RAG+lightweight adapters tend to be more maintainable than heavy finetunes. Design a data strategy: curated instruction data, preference data for alignment, and evaluation sets with well-defined criteria.

Safety, robustness, and security: Evaluate for hallucination rate, toxicity, data leakage, and prompt injection—especially if the model ingests user-provided documents or tickets. Enjoy models with well-established safety tuning and a strong community. Establish guardrails: allow listed tools, policy engines, and human approval for high-stakes actions.

Ecosystem maturity: Evaluate the level of community engagement, the quality of the documentation, whether there are checkpoints, and if it is compatible with your stack (PyTorch/HF, deployment runtimes, monitoring). The 'right' LLM is the one you can run reliably — with measurable quality, controlled risk, and predictable cost — throughout its life, not just on a demo.

With ongoing improvements in fine-tuning methodologies, reliability validation, and data protection protocols, open source LLMs are poised to drive responsible and impactful adoption across industries. As the community continues to address challenges related to scalability, bias, and security, the future of generative AI is set to become even more inclusive, transparent, and transformative. END

References

- *https://www.ibm.com/think/topics/open-source-llms*
- *https://github.com/eugeneyan/open-llms*
- *https://www.civo.com/blog/open-source-vs-proprietary-llms*
- *https://www.datacamp.com/tutorial/fine-tuning-large-language-models*
- *https://medium.com/data-scientists-from-future/fine-tuning-open-source-language-models-a-step-by-step-guide-a38bed8df923*
- *https://www.ibm.com/think/topics/ai-hallucinations*
- *https://www.appsmith.com/blog/de-hallucinate-ai-agents*
- *https://www.evidentlyai.com/blog/llm-hallucination-examples*
- *https://www.freecodecamp.org/news/how-to-evaluate-and-select-the-right-llm-for-your-genai-application/*
- *https://www.dataiku.com/stories/blog/key-criteria-when-selecting-an-llm*

By: Dr Magesh Kasthuri and Dr Anand Nayyar

Dr Magesh Kasthuri is a PhD in artificial intelligence and the genetic algorithm. He currently works as a distinguished member of the technical staff (master) and chief architect at Wipro Ltd. This article expresses his views and not of the organisation he works in.

Dr Anand Nayyar is a PhD in wireless sensor networks and swarm intelligence. He works at Duy Tan University, Vietnam, and loves to explore open source technologies, IoT, cloud computing, deep learning, and cyber security.

How a Job Portal Benefited from Microservices Architecture

Microservices architecture has emerged as a preferred pattern for building scalable and maintainable software applications. Here's how a monolithic job portal application was re-engineered into a microservices-based system. The migration process, key design decisions, the technology stack used, and measurable improvements in performance and flexibility are all laid out for you.



As an application scales, it must migrate from a monolithic architecture to a microservices-based one. Here, we document the migration of a job portal from a monolithic architecture to a microservices-based design. Initially, all core features—job listings, company profiles, and user reviews—were embedded in a single application. While this structure offered simplicity, it became a bottleneck as the platform grew. To improve scalability and maintainability, the system was redesigned using independent services for job, company, and review management. Each service now operates with its own database and deployment pipeline, enabling fault isolation and independent scaling.

The schema of the project is designed to represent the core entities: Job, Company, and Review. In the monolithic version, these entities share a centralised database. However, in the microservices architecture, each service manages its own database, ensuring data autonomy and service independence.

The transition

The transition from a monolithic to a microservices architecture involved carefully decoupling the tightly integrated components of the original system. Initially, all business logic for job postings, company information, and user reviews resided within a single codebase and shared a common database, as illustrated in Figure 1. To enable independent development and deployment, each core feature was extracted into its own service as depicted in Figure 2, with separate repositories and databases. APIs were introduced for inter-service communication, replacing direct

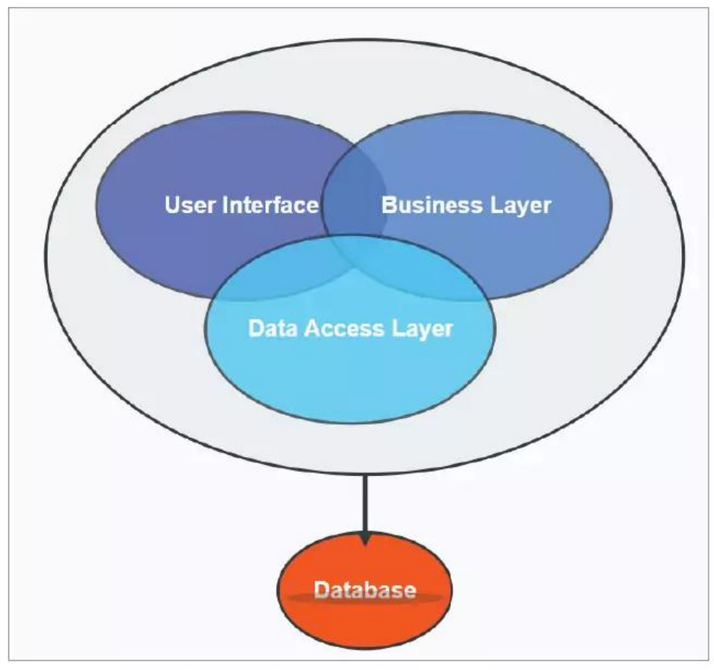


Figure 1: Monolithic architecture

function calls and database joins. This shift required restructuring data flow, implementing service discovery, and addressing challenges such as data consistency, API versioning, and centralised logging. Despite the complexity, the transition significantly improved scalability, maintainability, and the ability to deploy updates without affecting unrelated parts of the system.

In the process of converting the job portal from a monolithic to a microservices architecture shown in Figure 3, several mathematical and computational models were utilised to ensure optimal system performance and efficiency. Queuing theory, analyse request handling and inter-service communication were applied to the model, helping to minimise latency and avoid bottlenecks. Load balancing algorithms, such as round-robin and least connections, were evaluated to distribute traffic evenly across services. Dependency graphs were used to identify tightly coupled components in the monolith, guiding the service decomposition strategy. Additionally,

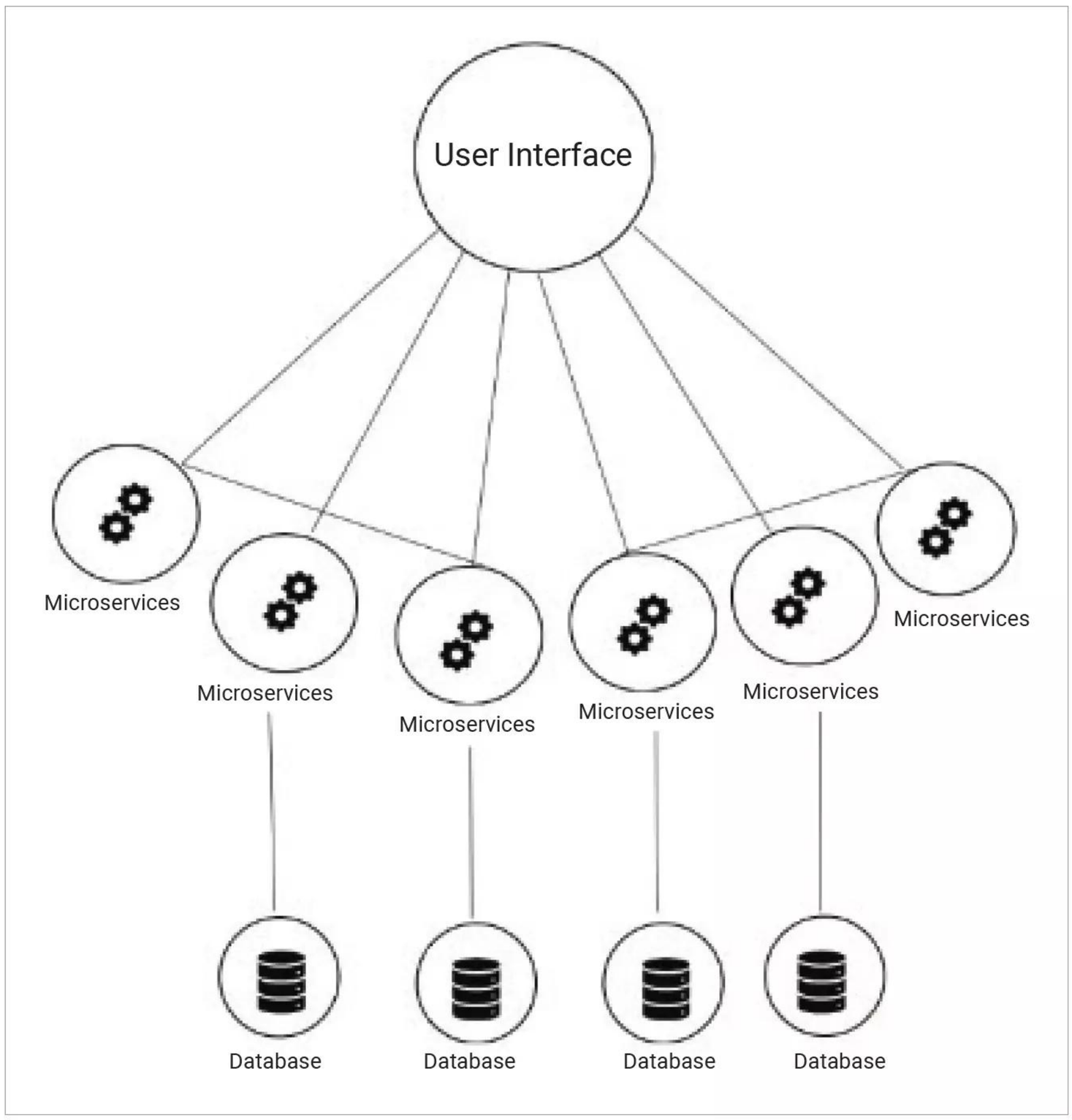


Figure 2: Microservices architecture

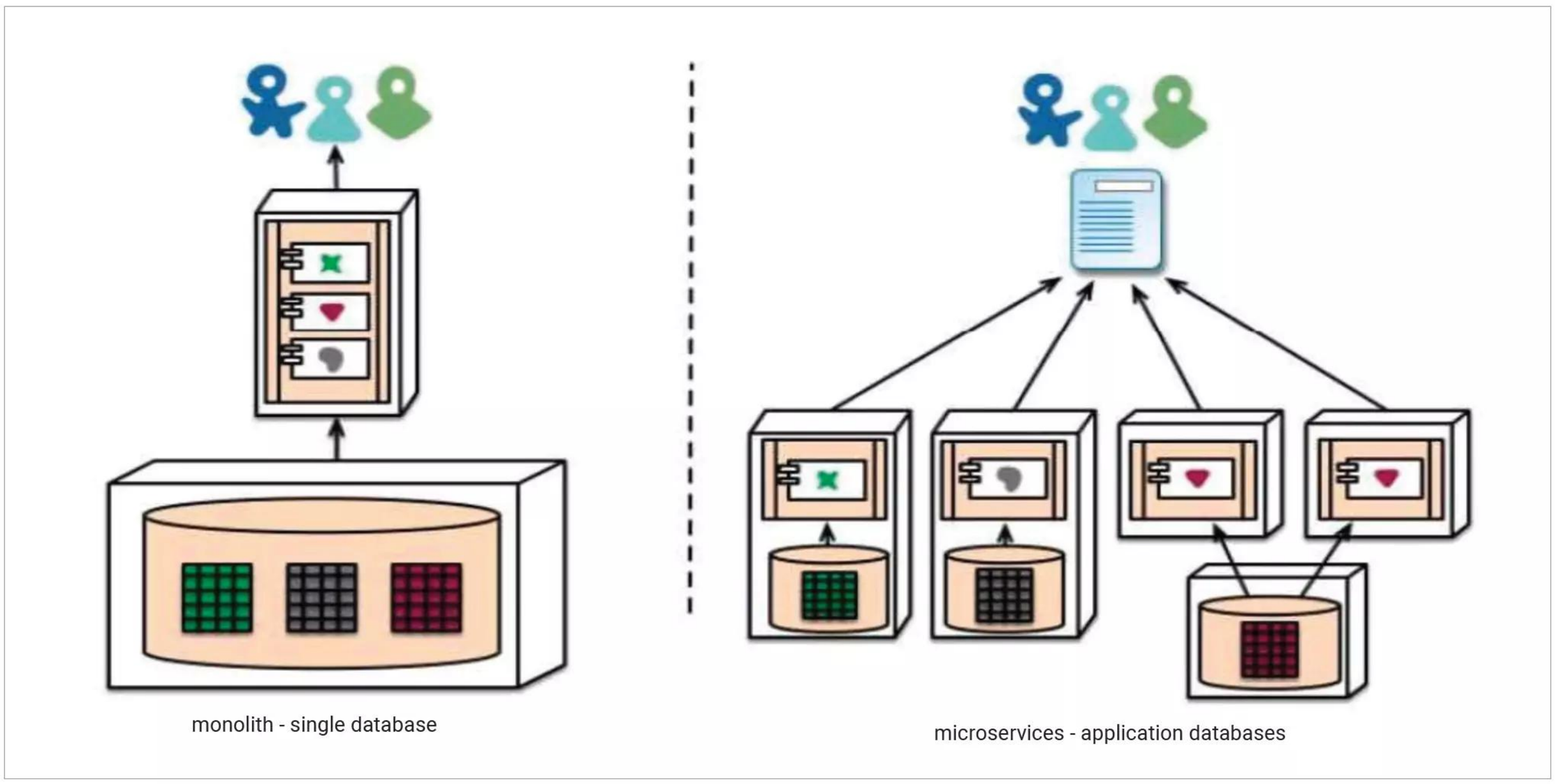


Figure 3: Transition process overview

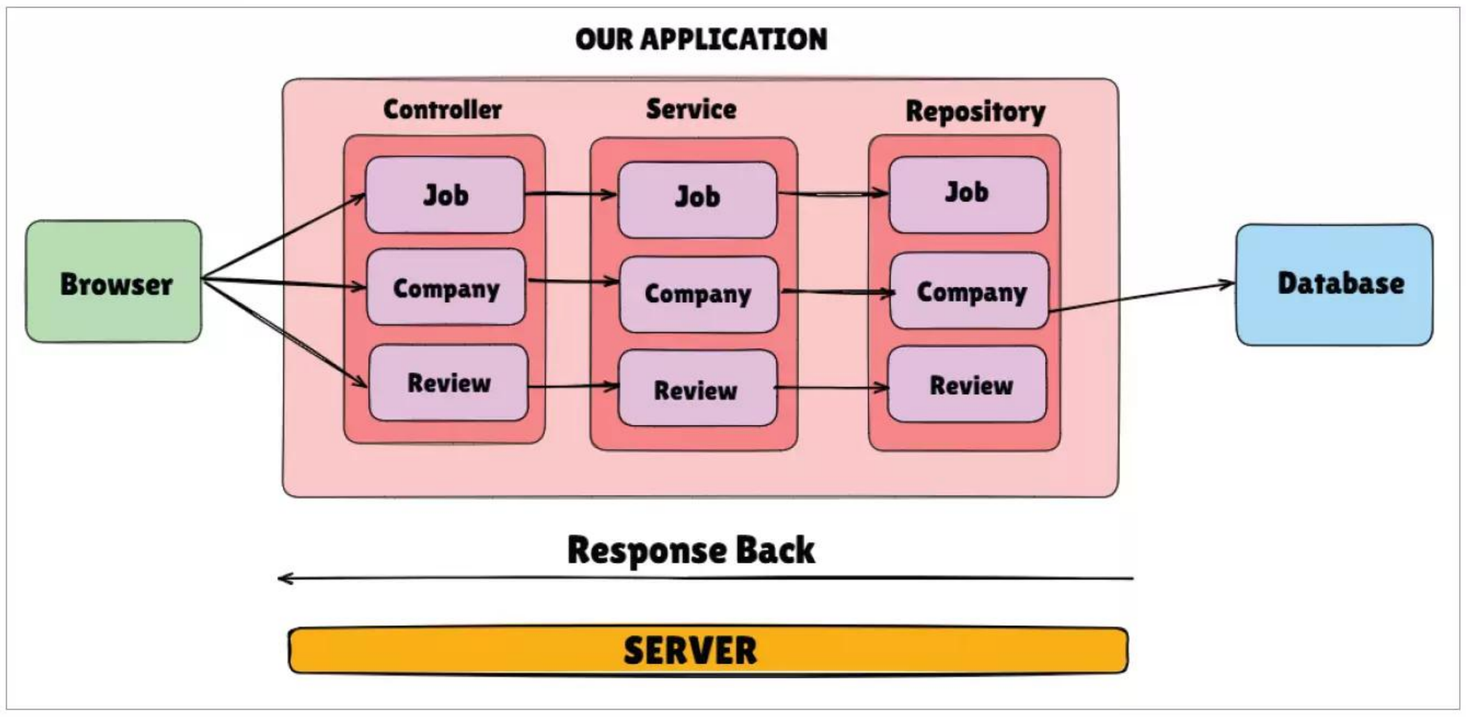


Figure 4: Job portal - monolithic application

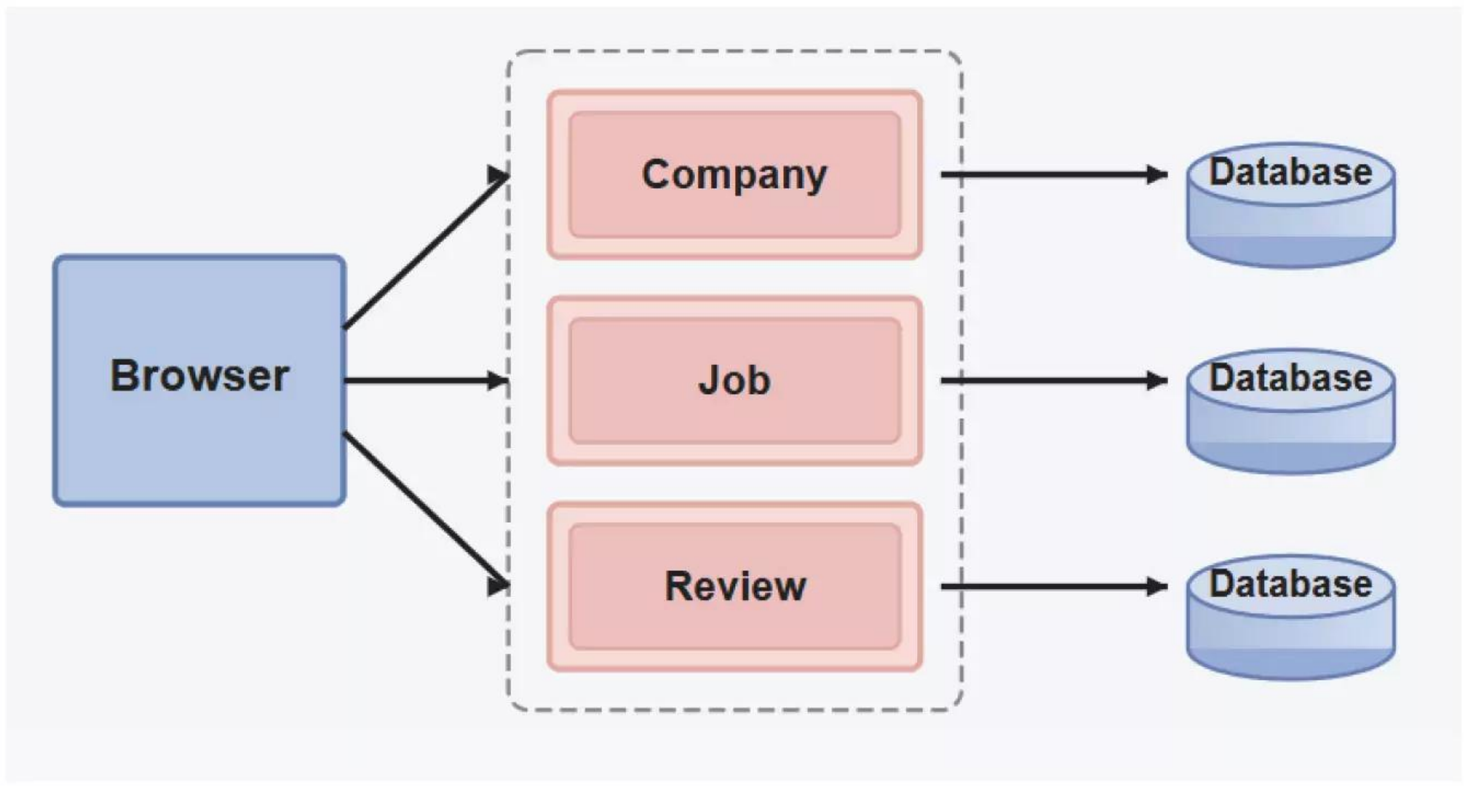


Figure 5: Job portal – microservices application

consistency models such as eventual consistency and CAP theorem principles were considered to manage data integrity across distributed databases. These models provided a structured approach to making architectural decisions, ensuring that the transition maintained high system reliability, scalability, and responsiveness.

Technology stack

Spring Boot was chosen for implementing the microservices architecture due to its robust support for building scalable, production-ready applications with minimal configuration. Prior to this transition, the system followed a monolithic architecture, where all functionalities were tightly coupled within a single application as shown in Figure 4. It provides a lightweight framework that simplifies the development of individual services, allowing each microservice—such as Job, Company, and Review—to be developed, tested, and deployed independently as presented in Figure 5. Spring Boot's seamless integration with Spring Cloud offers powerful tools for service discovery (via Eureka), centralised configuration, load balancing (using Traefik), and fault tolerance (with Hystrix or Resilience4j). Moreover, its built-in support for RESTful APIs, embedded servers like Tomcat, and easy integration with databases makes it an ideal choice for microservices. These features significantly accelerated development while ensuring consistency, flexibility, and maintainability across the entire system.

Technology stack for the microservices-based open source job portal

Component	Technologies
Frontend	React, Next.js
Backend	Java, Spring Boot
Database	PostgreSQL
Containerisation	Docker
Monitoring/Tracing	Zipkin
Message queue	RabbitMQ
Service discovery	Eureka
Load balancer	Traefik

Folder and file structure

Figure 6 describes the project structure of the microservices application, which is modularized as follows:

- **dto/**: Contains Data Transfer Objects for efficient data communication between services.
- **external/**: Handles inter-service communication using REST clients.
- **impl/**: Contains business logic implementation.
- **Job.java**: Defines the Job entity mapped to the database.
- **JobController.java**: Manages HTTP endpoints for Job-related operations.
- **JobRepository.java**: Provides CRUD operations for the Job entity.
- **JobService.java**: Contains core job-related business logic.

DTO (Data Transfer Object)

DTO stands for Data Transfer Object. It is an object that is used to transfer data between software application subsystems or layers, typically over a network. DTOs are designed to be simple containers for data, and they do not contain any business logic. They are primarily used to reduce the number of method calls by encapsulating multiple data elements in a single object. For example, when transferring data from a database to a client application, a DTO can aggregate various pieces of information (such as a user's name, email, and address) into a single object. This helps optimise communication, especially in distributed

```
.
├── companyms
│   ├── docker-compose.yml
│   ├── HELP.md
│   ├── mvnw
│   ├── mvnw.cmd
│   ├── pom.xml
│   ├── src
│   └── target
├── configserver
│   ├── HELP.md
│   ├── mvnw
│   ├── mvnw.cmd
│   ├── pom.xml
│   ├── src
│   └── target
├── gateway
│   ├── HELP.md
│   ├── mvnw
│   ├── mvnw.cmd
│   ├── pom.xml
│   ├── src
│   └── target
├── jobms
│   ├── HELP.md
│   ├── mvnw
│   ├── mvnw.cmd
│   ├── pom.xml
│   ├── src
│   └── target
├── project-structure.txt
├── reviewms
│   ├── HELP.md
│   ├── mvnw
│   ├── mvnw.cmd
│   ├── pom.xml
│   ├── src
│   └── target
└── service-reg
    ├── HELP.md
    ├── mvnw
    ├── mvnw.cmd
    ├── pom.xml
    ├── src
    └── target
```

Figure 6: Folder structure of a microservices-based job portal

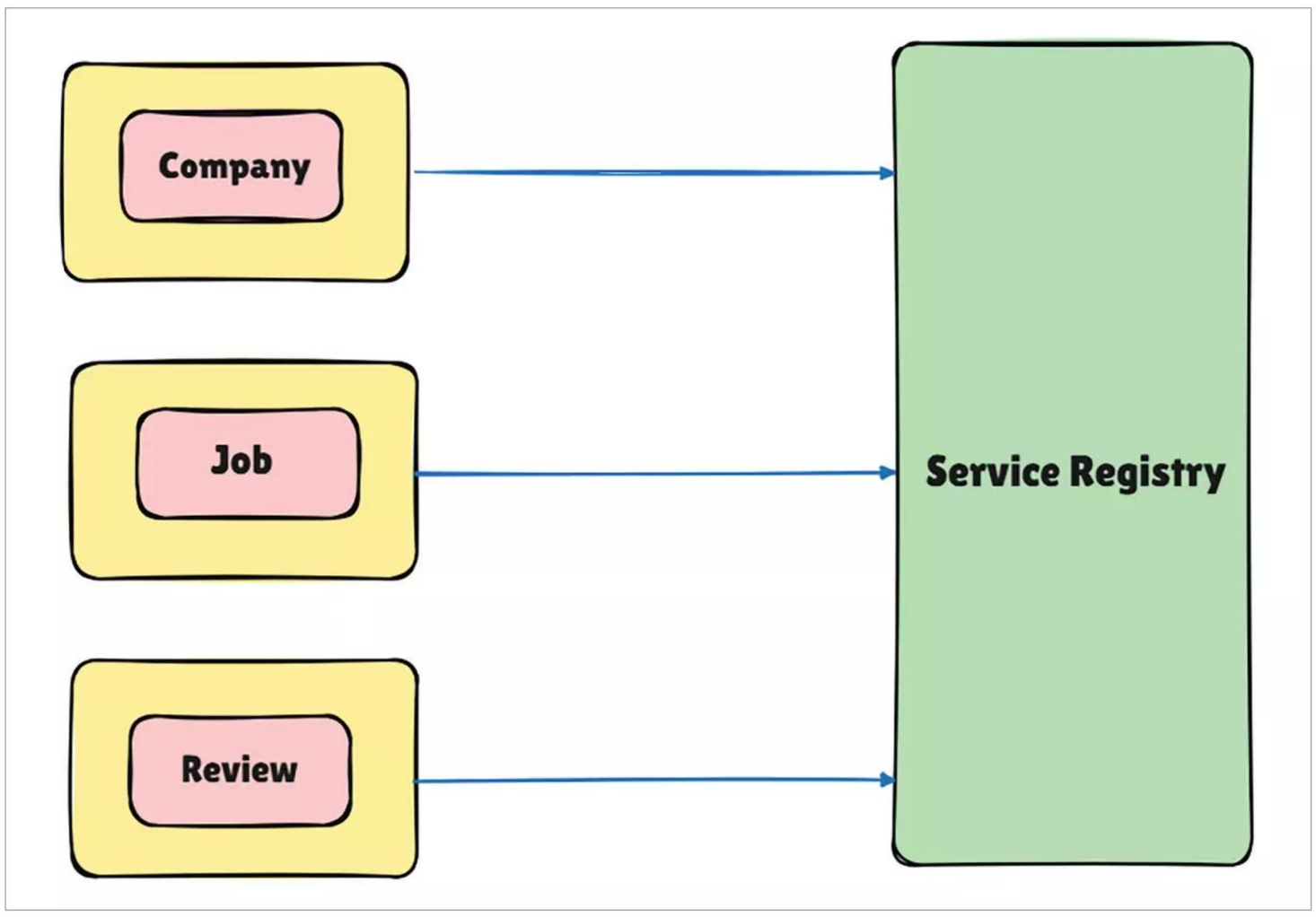


Figure 7: Service registry

systems where network calls may be expensive or slow. DTOs are often used in combination with other patterns, such as the DAO (Data Access Object) pattern, and are particularly useful in web services and APIs where efficient data transfer is crucial.

Service registry

A service registry is a centralised database or repository that maintains information about available services in a distributed system or microservices architecture. It acts as a directory where services register themselves upon startup, providing metadata such as the service name, network location, and status. The registry allows other services to discover and connect to the registered services dynamically without hardcoding their locations. This is especially useful in environments where services may scale up or down, and their network addresses change frequently. Service registries are often used in conjunction with service discovery mechanisms, which help clients find and access services by querying the registry. Tools like Eureka are commonly used to implement service registries as demonstrated in Figure 7, ensuring high availability, fault tolerance, and scalability in microservices-based architectures.

Fault tolerance

Fault tolerance refers to the ability of a system to continue functioning correctly even in the presence of failures. A fault-tolerant system is specifically designed to handle unexpected disruptions, such as hardware failures, network issues, or software bugs, without compromising the overall user experience or service reliability. To achieve fault tolerance, systems often employ redundancy, graceful degradation, and various error-handling mechanisms, such as retries, timeouts, and fallback procedures.
For instance, critical components may be replicated across multiple servers or regions to ensure that if one server fails, another can take over seamlessly, minimising service interruptions.

One key strategy for maintaining fault tolerance, particularly in microservices architectures, is the implementation of circuit breakers. A circuit breaker, such as the companyBreaker instance used in this system, monitors service interactions, specifically tracking failure rates and slow call rates. The circuit breaker ensures that if the failure or slow call

rate exceeds predefined thresholds (for example, a failure rate threshold of 50), appropriate protective actions are triggered.

Under normal operation, when no failures or slow calls are detected, the circuit breaker remains in the CLOSED state, allowing uninterrupted service. By carefully monitoring metrics such as failure rates, buffered calls, and slow calls, the circuit breaker helps maintain system resilience while minimising performance degradation. This approach contributes significantly to fault tolerance by preventing systemwide failures and ensuring high availability and reliability, even during adverse conditions.

The goal of fault tolerance, supported by mechanisms like circuit breaking, is to minimise the impact of failures, ensuring continuous, reliable service for mission-critical applications.

Circuit breaker pattern

The circuit breaker pattern is a design pattern, represented in Figure 8, used to detect failures in a system and prevent them from cascading. In a microservices architecture or distributed system, services often depend on other services to function properly. When one service becomes unavailable or experiences issues, it can cause a chain reaction of failures across the system. The circuit breaker pattern helps mitigate this by monitoring service calls and breaking the circuit (that is, preventing further calls) when a threshold of failures is reached. Listing 1 illustrates the implementation of a circuit breaker using RestTemplate.

After a certain time or recovery period, the circuit breaker allows a limited number of test requests to check if the service has recovered. If the service is healthy again, the circuit breaker closes, and normal operation resumes. This approach helps improve system resilience, reduce downtime, and ensure that failures in one service do not impact the entire system.

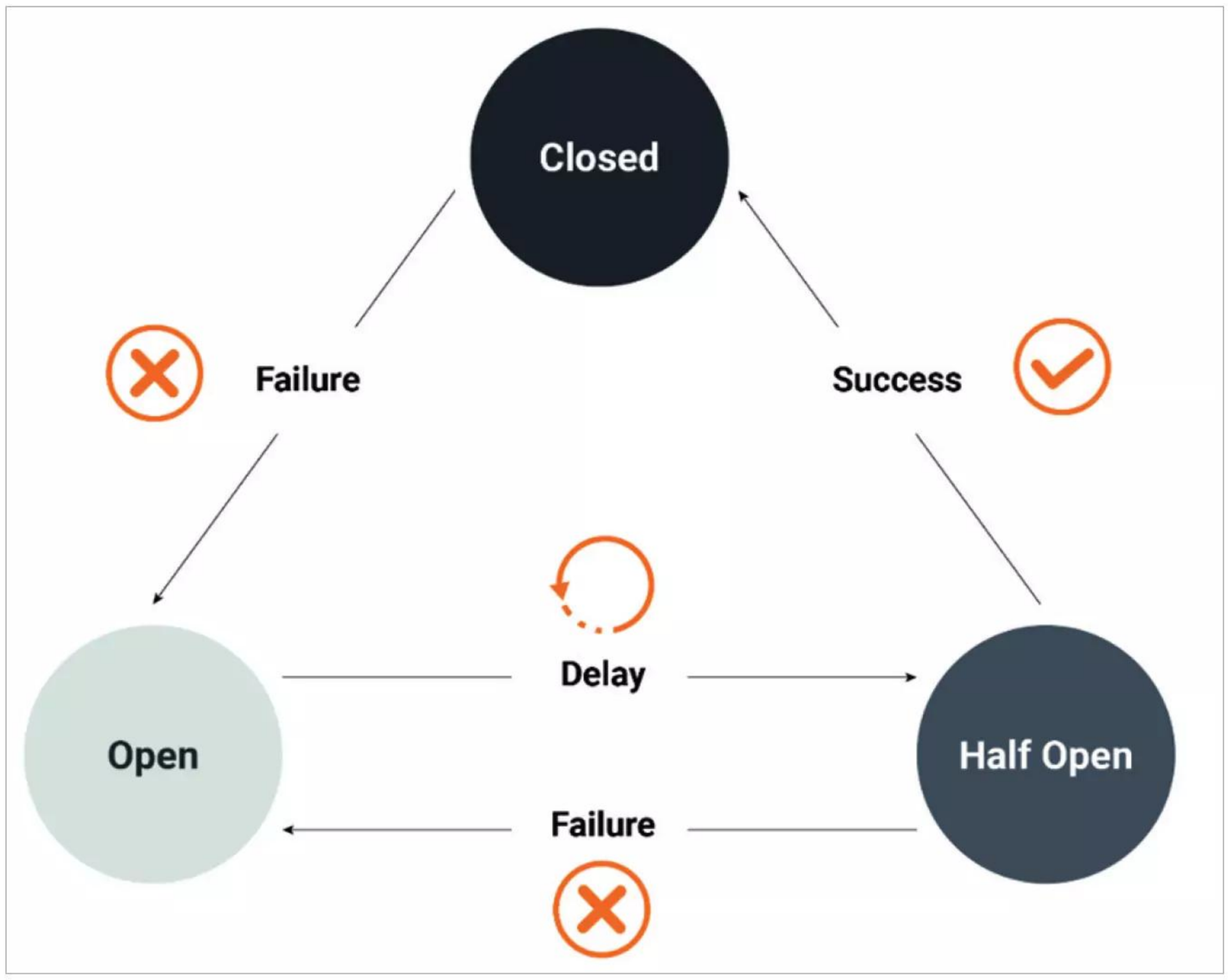


Figure 8: Methodology of circuit breaker pattern

Listing 1: Circuit breaker with RestTemplate

```
@Service
public class CompanyServiceClient {
@Autowired
private RestTemplate restTemplate;
private static final String BASE_URL =
“http:// COMPANY-SERVICE/companies”;
@CircuitBreaker(name =
“companyService”, fallbackMethod =
“fallbackGetCompanyById”)
public CompanyDTO getCompanyById(Long
companyId)
{
return restTemplate.getForObject(BASE_
URL + “/” + companyId, CompanyDTO.
class);
}
// Fallback method called when the
circuit is open
public CompanyDTO
fallbackGetCompanyById(Long companyId,
Throwable throwable) {
CompanyDTO fallbackCompany = new
CompanyDTO ();
fallbackCompany.setId(companyId);
fallbackCompany.setName(“Company
information
not available at the moment”);
return fallbackCompany;
}
```

RabbitMQ integration

RabbitMQ is an open source message broker that facilitates communication between distributed applications or services by enabling them to exchange messages asynchronously. It implements the Advanced Message Queuing Protocol (AMQP) and is designed to ensure reliable, scalable, and fault-tolerant messaging. RabbitMQ, as outlined in Figure 9, allows producers to send messages to a queue, from which consumers can retrieve and process them. This decouples the components of an application, enabling them to communicate without direct dependencies.

RabbitMQ supports features such as message persistence (ensuring messages are not lost), message routing, load balancing, and delivery guarantees, which make it ideal for systems that require high throughput and fault tolerance. It is widely used in microservices architectures, event-driven systems, and for integrating

disparate systems with different communication patterns.

Use of OpenFeign

In this project, OpenFeign has been integrated as a declarative REST client to simplify inter-service communication between the microservices within the Micro Job App architecture. Instead of manually using RestTemplate or WebClient, Feign clients have been defined via interfaces annotated with @FeignClient. This allows different microservices (such as the Job Service, User Service, or Notification Service) to easily call each other over HTTP without writing boilerplate code for request handling.

For example, when the Job Service needs to fetch user profile data from the User Service, it simply calls a method on the corresponding Feign interface, and OpenFeign automatically handles the request, response mapping, and error handling under the hood. This greatly enhances readability, maintainability, and modularity of the code.

Listing 2: Feign client example

```
@FeignClient(name = “user-service”)
public interface UserClient { @
GetMapping(“/users/{id}”)
UserDTO getUserById(@PathVariable(“id”)
Long id);
}
```

Use of Zipkin

Zipkin has been incorporated into this project for distributed tracing across the microservices, helping developers track and visualise the flow of requests through the system. Each microservice is configured to send tracing data to the Zipkin server, which collects and aggregates these traces. When a client initiates a request (for example, posting a new job), the system records detailed timing information as the request travels through multiple services (such as Job Service → Notification Service → Email Service). This helps in identifying performance bottlenecks, understanding request latency, and diagnosing failures across service boundaries. The integration with Spring Cloud Sleuth further enriches the tracing data by automatically tagging logs with trace IDs and span IDs, making it easier to correlate logs and traces.

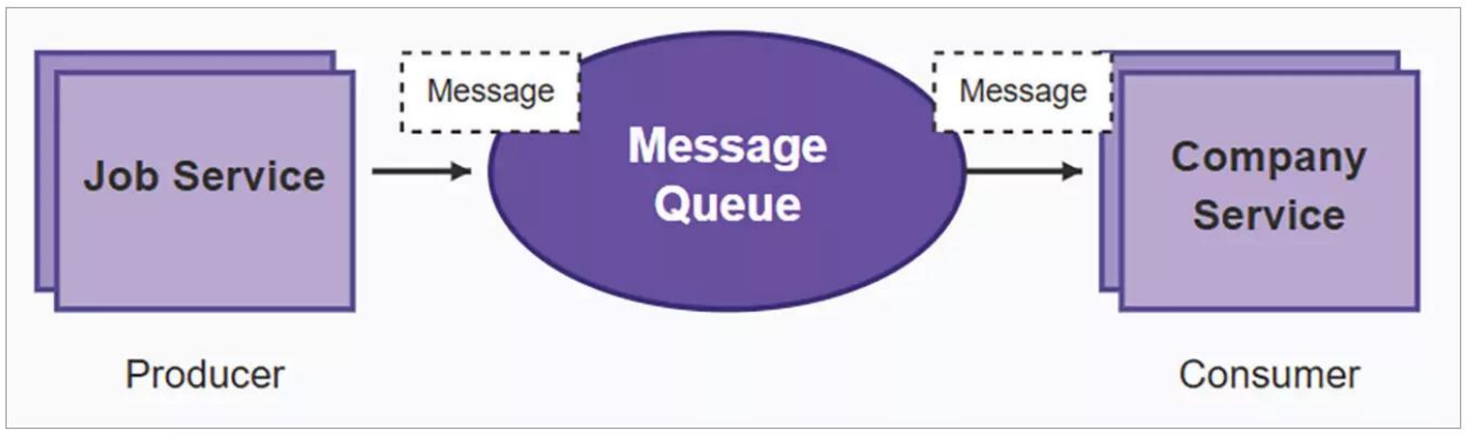


Figure 9: Flow of messages

Listing 3: Zipkin configuration

```
spring.zipkin.base-url=http://
localhost:9411/ spring.sleuth.sampler.
probability=1.0
```

The results

The transition from a monolithic architecture to a microservices-based architecture for the job portal resulted in significant improvements across multiple dimensions of system performance, development agility, and operational efficiency. Here are the key benefits we observed.

Before migration, the monolithic application could reliably handle up to 500 concurrent users before response times degraded significantly (average latency increased to over 1200ms). After migration, the microservices-based system handled 2000 concurrent users with acceptable performance (average latency stayed below 400ms). This represents a 4X increase in user capacity with a 66% reduction in average response time. The experimental setup and results are demonstrated in Figures 10 and 11.

Deployment time: The full deployment of the monolithic application took approximately 30 minutes with downtime. In the microservices setup, deployment of individual services (Job, Company, and Review) now takes about 5 minutes per service with zero downtime using rolling updates. This resulted in an 83% reduction in deployment time.

Fault isolation and recovery: Previously, a failure in one module could cause partial or total system downtime. With microservices, failures are isolated; for instance, issues in the Review service do not affect Job or Company services. Average recovery time was reduced from around 20 minutes to about 3 minutes due to independent service restarts.

Resource utilisation: Monolithic deployment required 4 vCPUs and 8GB RAM constantly, irrespective of module loads. In contrast, microservices scaled independently:

- **Job service:** 2 vCPUs, 4GB RAM
- **Company service:** 1 vCPU, 2GB RAM
- **Review service:** 1 vCPU, 2GB RAM

This led to a 30–40% reduction in total resource consumption under normal load conditions.

Improvements and challenges

The migration to a microservices architecture significantly enhanced the flexibility, performance, and reliability of the job portal system. By decoupling services, implementing asynchronous communication (RabbitMQ), enabling service discovery (Eureka), utilising declarative REST clients (OpenFeign), and adopting distributed tracing (Zipkin), the system now supports dynamic scaling, fault isolation, and faster feature rollouts.

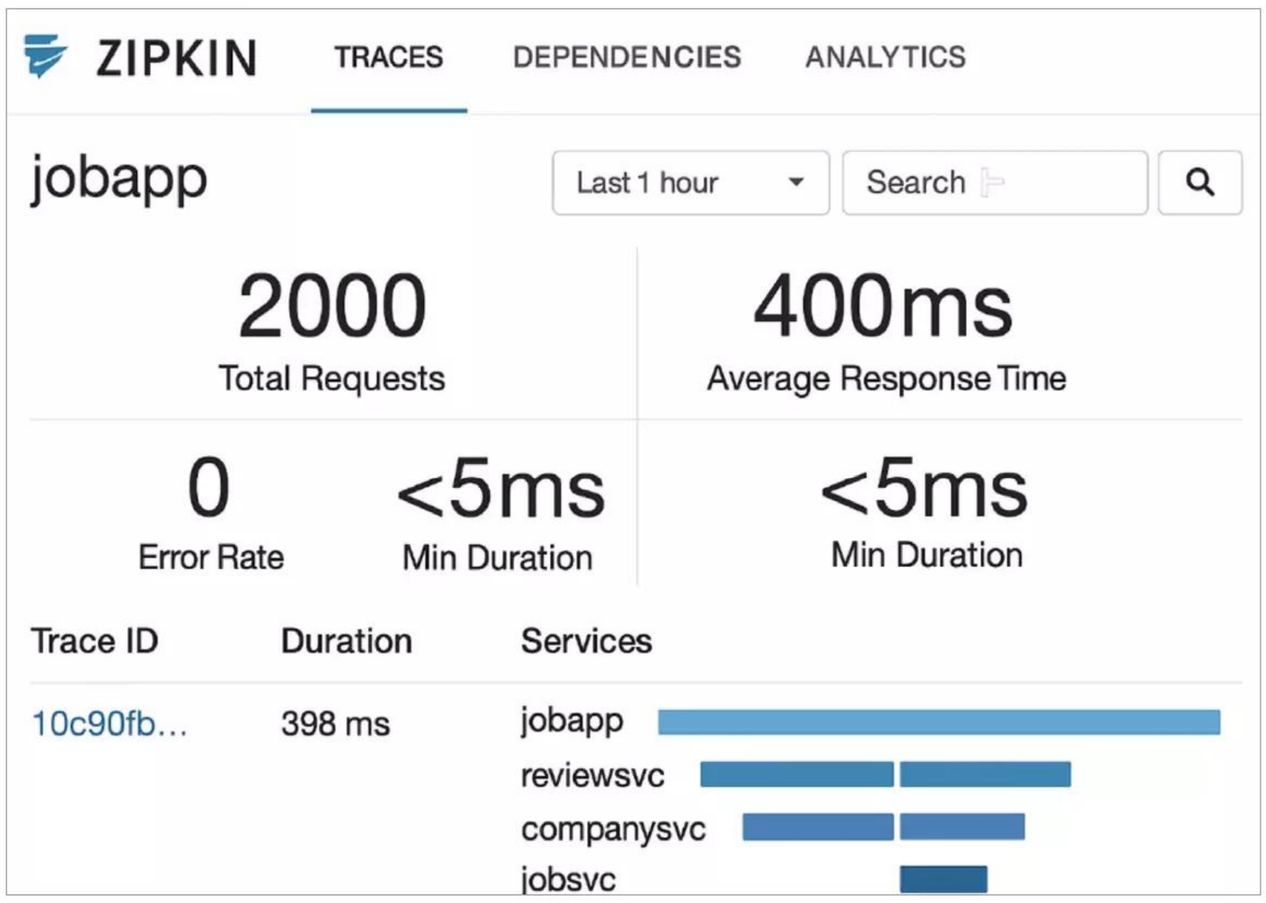


Figure 10: Metrics using Zipkin

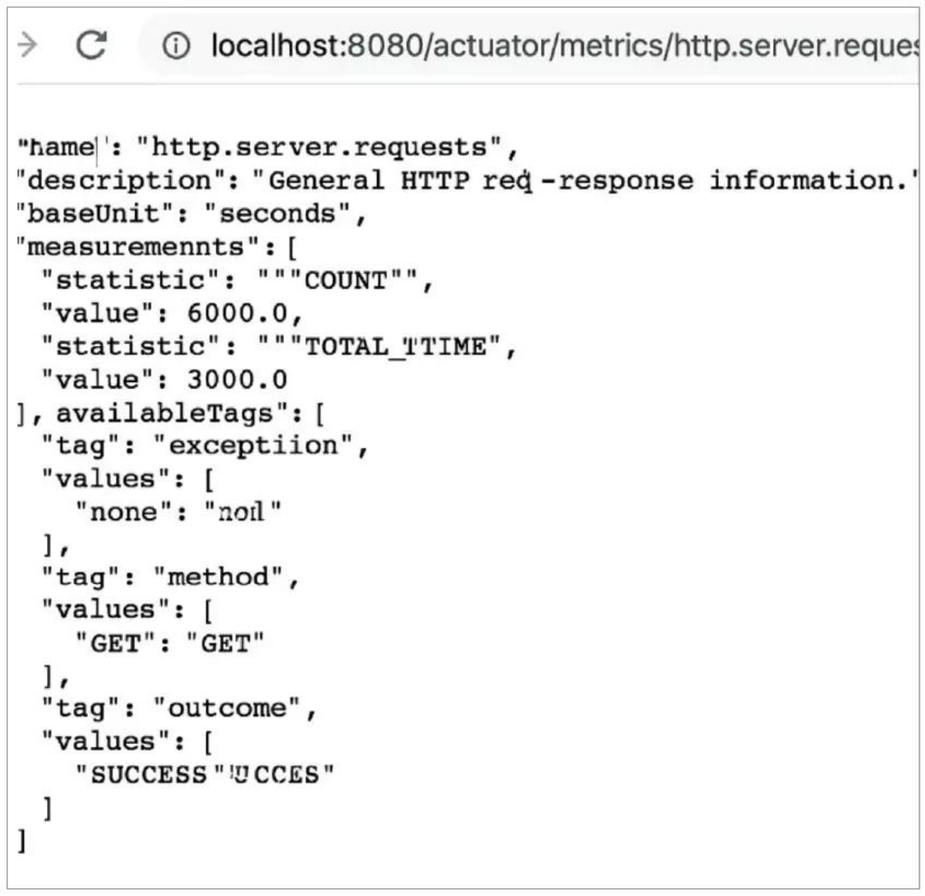


Figure 11: Metrics using Spring Boot Actuator

Quantitatively, the platform achieved a 4X improvement in scalability, 83% faster deployment, and 30–40% savings in resource utilisation, validating the architectural shift and setting a robust foundation for future expansion.

Though this migration brought significant improvements, it also introduced new complexities that warranted careful management. While scalability, resilience, and development agility were greatly enhanced, challenges such as service orchestration, distributed data management, and inter-service communication overhead emerged. To address these, techniques like asynchronous messaging (RabbitMQ), service discovery (Eureka), client-side load balancing (Ribbon), and declarative REST clients (OpenFeign) were employed.

Furthermore, the implementation of circuit breakers using Resilience4j ensured that transient failures in one service did not propagate through the system, preserving overall stability. Monitoring tools such as Zipkin for distributed tracing and Spring Boot Actuator for health checks provided critical visibility into system behaviour, enabling proactive maintenance. Overall, the microservices approach introduced manageable trade-offs, and the net gain in system flexibility, fault isolation, and scalability validated the architectural transition as a sustainable foundation for future expansion. END

References

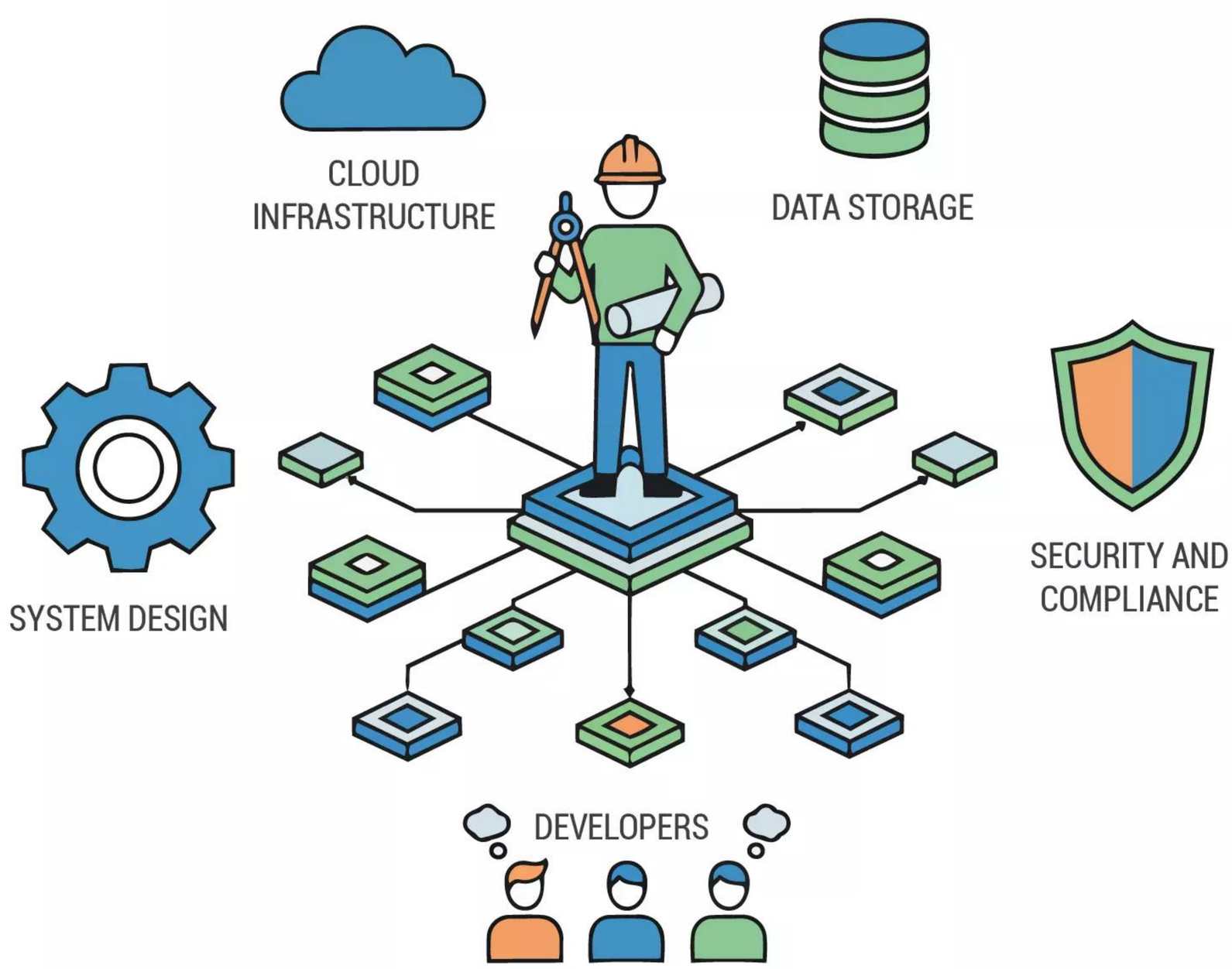
- R. Mishra, N. Jaiswal, R. Prakash, and P.N. Barwal, 'Transition from monolithic to microservices architecture: Need and proposed pipeline', in 2022 International Conference on Futuristic Technologies (INCOFT), 2022, pp. 1–6
- A.A. Kiani, A. Sarwar, H. Arshad, A. Bibi, F.B. Ahmad, and W. Ahmed, 'Catalysing monolithic to microservices migration: A strategic approach using refactoring and pilot projects', in 2024 International Conference on Engineering Computing Technologies (ICECT), 2024, pp. 1–7
- S. Bataieneh, A. Ziadeh, and L.F. Al-Qora'n, 'Microservices architecture for improved maintainability and traceability in MVC-based e-learning platforms: Roadmap for future developments', in 2024 15th International Conference on Information and Communication Systems (ICICS), 2024, pp. 1–6
- Y. Jayawardana, R. Fernando, G. Jayawardena, D. Weerasooriya, and I. Perera, 'A full stack microservices framework with business modelling', in 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer), 2018, pp. 78–85
- N.S.S.A and S. Sebastian, 'Circuit breaker: A resilience mechanism for cloud native architecture', in 2023 Global Conference on Information Technologies and Communications (GCITC), 2023, pp. 1–8
- G.S. Kumar and J.T. Jose, 'Developing an electronic commerce platform', in 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), 2017, pp. 1309–1313
- A.E. Malki and U. Zdun, 'Combining API patterns in microservice architectures: Performance and reliability analysis', in 2023 IEEE International Conference on Web Services (ICWS), 2023, pp. 246–257
- J. Williams, (2012). *RabbitMQ in Action: Distributed Messaging for Everyone.* Manning Publications.
- K. Hmue, M.P. Phyu, and A.M.M. Paing, 'Microservices vs monolith: A comparative analysis and problem-solving approach in web development area', in 2024 5th International Conference on Advanced Information Technologies (ICAIT), 2024, pp. 1–5
- S. Athreya, S. Kurian, A. Dange, and S. Bhatsangave, 'Implementation of serverless e-commerce mobile application', in 2022 2nd International Conference on Intelligent Technologies (CONIT), 2022, pp. 1–5
- The complete source code for this work is available at: *https://github.com/siddeshwar-kagatikar/Job-Portal*

By: Bhavya Kapadia, Siddeshwar Kagatikar and B. Thangaraju

The authors are associated with the Open Source Technology Lab at the International Institute of Technology, Bengaluru.

The Role that Software Architects Play

Software architects design software projects and ensure these meet their goals. They must balance tech skills with leadership and mentoring abilities.



Software architects are the principal designers of software development projects. They are in charge of the overall software project structure, making sure that software modules are properly integrated with the stated business goals, while software developers concentrate on producing code for a particular objective or feature. An architect makes a project secure, scalable, and feasible given the resources at hand.

Software architects help technical teams to achieve business goals in a seamless way. Their responsibilities typically include:

High-level design: Designing the outline of the project by defining components, and their work and data flow.

Technology selection: Selecting the best technology from a range of possibilities, such as Java or Python, SQL or NoSQL, and cloud providers like AWS or Azure.

Setting standards: Advising the development team to adhere to quality standards, security procedures, and coding norms.

Risk mitigation: Identifying possible bottlenecks in the areas of maintenance, storage, performance, and scalability.

The roles of a software architect and software engineer are collaborative but differ significantly in scope.

Feature	Software architect	Software engineer
Focus	Designing high-level system structure	Developing individual features and module
Perspective	Designing the conceived project	Implementing the project
Key output	Flow-diagram, UML diagrams, plan	Program code, tests of modules, and bug fixing
Stakeholders	Clients, project managers	Architects, testing teams, and lead developers

Essential skills

To be a successful software architect, a software professional should balance technical skills with leadership qualities.

System design: Proficiency in architectural patterns like microservices, monolithic, or event-driven architecture.

Communication: The ability to explain technical trade-offs to non-technical business owners.

Strategic thinking: Comprehend the problem domain to ensure the best software-based solution for the given project.

Leadership and mentoring: Oversee the entire team's work and guide the group through the challenging technical aspects of the project.

Architects typically begin their careers as senior software engineers and are promoted to the position of software architect after five to ten years of experience. Then, rather than how to write a function, they focus on how to solve a problem as a whole.

Software engineers must maintain a balance between technical proficiency and emotional intelligence to become great software architects. An architect is frequently described as a "developer who knows how to talk to management" and "still remembers how to code."

Here are 10 critical 'Dos' and 'Don'ts' for the role.

The 5 success drivers

Focus on trade-offs: There are benefits and drawbacks to every architectural decision. The trade-offs of the possibilities should be weighed and the architecture most suited to the project's execution and success should be selected.

Write ADRs (architecture decision records): To make it possible for the customer and other members of the project team to understand the rationale behind the chosen design, it is important to document its background.

Stay 'hands-on': To guarantee the implementability of the selected design, the coding process should proceed with prototype implementation, PoCs (proof of concepts), and codebase execution.

Monitor parameters: For the system's long-term health, an eye must be kept on non-functional metrics like security, scalability, and maintainability.

Practice an evolutionary approach: In order to adjust to changes in the future, an evolutionary approach should be taken.

The 5 common pitfalls

Don't be an 'Ivory Tower' architect: Discuss with the development team rather than working alone. For a design to be successful, the developers must approve it.

Don't over-engineer: Avoid complex design; if a straightforward monolithic design on a single server can solve the intended business issue, stay away from designs utilising Kubernetes or microservices.

Don't be a bottleneck: Don't design the project with complex and hard to learn technologies. To enable the team to work quickly without you, design the project using straightforward and uncomplicated technology.

Don't ignore future technical risk: The project should never descend into chaos; instead, it should always emphasise the risks with justification and potential remedies.

Don't fall in love with a technology: Avoid being biased towards a technology; even if you don't like it, the best tool is the one that matches the particular project.

Example of an architectural design

ADR 000: Database selection for user profile and activity service

Given below is the architecture decision record (ADR) of a project where the main design task is to select a RDBMS or NoSQL for a user profile and activity tracking system. Since data transfer is much more difficult than rewriting code, selecting a database is one of the most important choices.

- *Status:* Accepted
- *Date:* 2026-01-15
- *Deciders:* Manoj Bhattacharjee (lead architect), Anjan Dutta (data engineer)
- *Consulted*: Backend team, Analytics team

Problem statement

A user appointment system needs to store two types of data:

- ***Core profile data:*** Highly structured information: user ID, email, subscription plan, password.
- ***Activity logs:*** Unstructured, high-volume data: clicks, page views, device metadata, and custom event attributes that change frequently.

The task is to decide whether to use a strict relational schema (PostgreSQL) or a flexible document store (MongoDB).

Decision-making factors

- ***Data consistency:*** Core user data must follow ACID properties: no partial updates to subscription status.
- ***Schema flexibility:*** Weekly adjustments in tracking points cause activity logs to change. To accommodate all these modifications, the system requires a flexible configuration.
- ***Query complexity:*** The system requires a process to join user profile data with billing data regularly.
- ***Write throughput:*** During peak hours, activity logs typically produce a large volume of data (5,000+) per second and grow quite quickly.

The options and the decision

- ***Option A: Pure relational (PostgreSQL)****:* Use standard tables for profiles and JSONB columns for unstructured logs.

- ***Option B: Pure NoSQL (MongoDB)****:* Store everything as documents.
- ***Option C:*** Use PostgreSQL for core profiles/billing and MongoDB for activity logs.
- ***Chosen option:*** Option A: Pure relational (PostgreSQL) with JSONB

Justification for the choice

Although Option C is theoretically feasible, the two distinct backup strategies, two security strategies, and the enormous effort of synchronisation between them can lead to significant operational complexity.

On the other hand, option A's PostgreSQL offers ACID compliance by default, and its JSONB capability enables the system to store unstructured activity logs with MongoDB-like performance. In a single, sophisticated database engine, this offers the best of both worlds.
For storing and working with JSON (JavaScript Object Notation) data, PostgreSQL has a very effective binary data type called JSONB. JSONB's binary format makes it significantly faster for processing and querying compared to the plain-text JSON type.

The positives of this choice are:

- ***Simplified operations:*** One database technology for scaling, backup, and monitoring.
- ***Strong integrity:*** The system can guarantee the integrity of activity logs in relation to a valid user ID by using the user ID as a foreign key.
- ***SQL power:*** The software team can use standard BI tools (Tableau/Metabase).

The downsides are**:**

- ***Storage size:*** Large PostgreSQL indexes on JSONB may necessitate a stringent data retention strategy by restricting the amount of time logs that are stored.
- ***Vertical scaling limits:*** In contrast to MongoDB, Postgres scaling is primarily vertical (larger servers) as opposed to horizontal (sharding), which could result in additional costs.

The golden rule for a software architect is to consider the design that is easy to implement both in terms of human and operational resources (hardware and software). It is preferable to view a design as immutable. For any significant modification, it is advisable to make a new design and designate the previous one as 'superseded' rather than altering the original. This helps us learn from earlier mistakes and maintains the history of the system's evolution. END

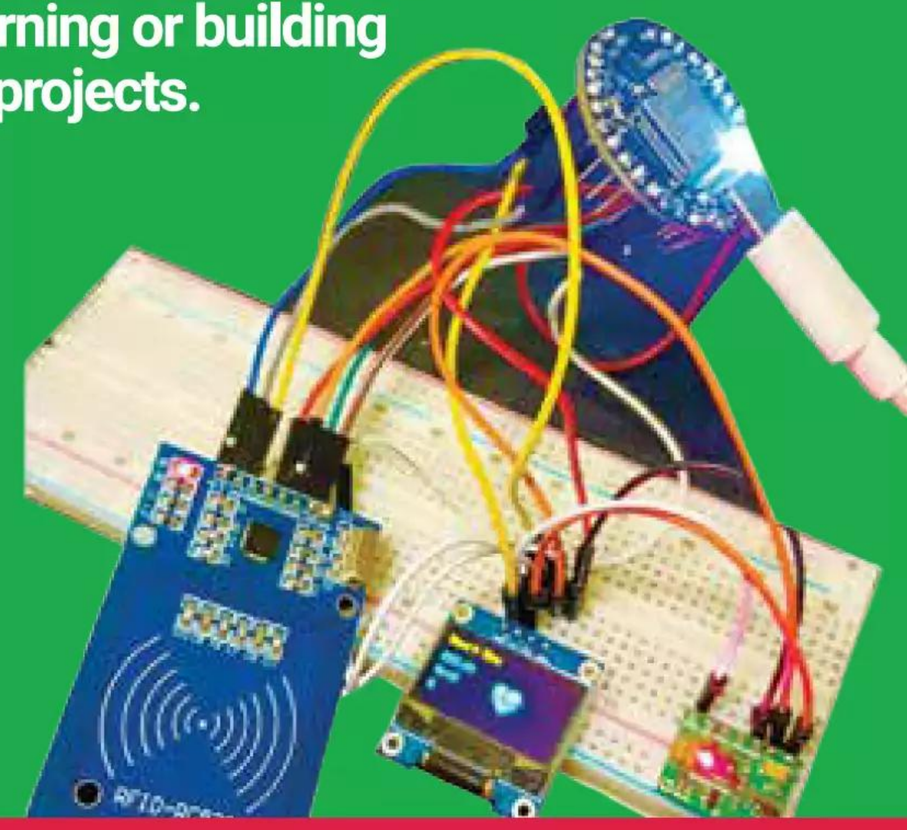
By: Dipankar Ray

The author has more than 30 years of experience in software development. He has published two books on MATLAB and medical image processing (published by PHI Learning and SPD).

DIY KIT:
NFC, RFID, IoT & Embedded Systems

Want to learn and master NFC and RFID technology along with an IoT-enabled embedded system?

This kit includes all the essentials to start learning or building RFID/NFC projects.



POWERED BY



IndusBoard COIN

Super small sized development board that enables you to create solutions for many use-cases where small size is super important.

SPECIAL INTRO PRICE

₹1,999 (Inclusive of GST)

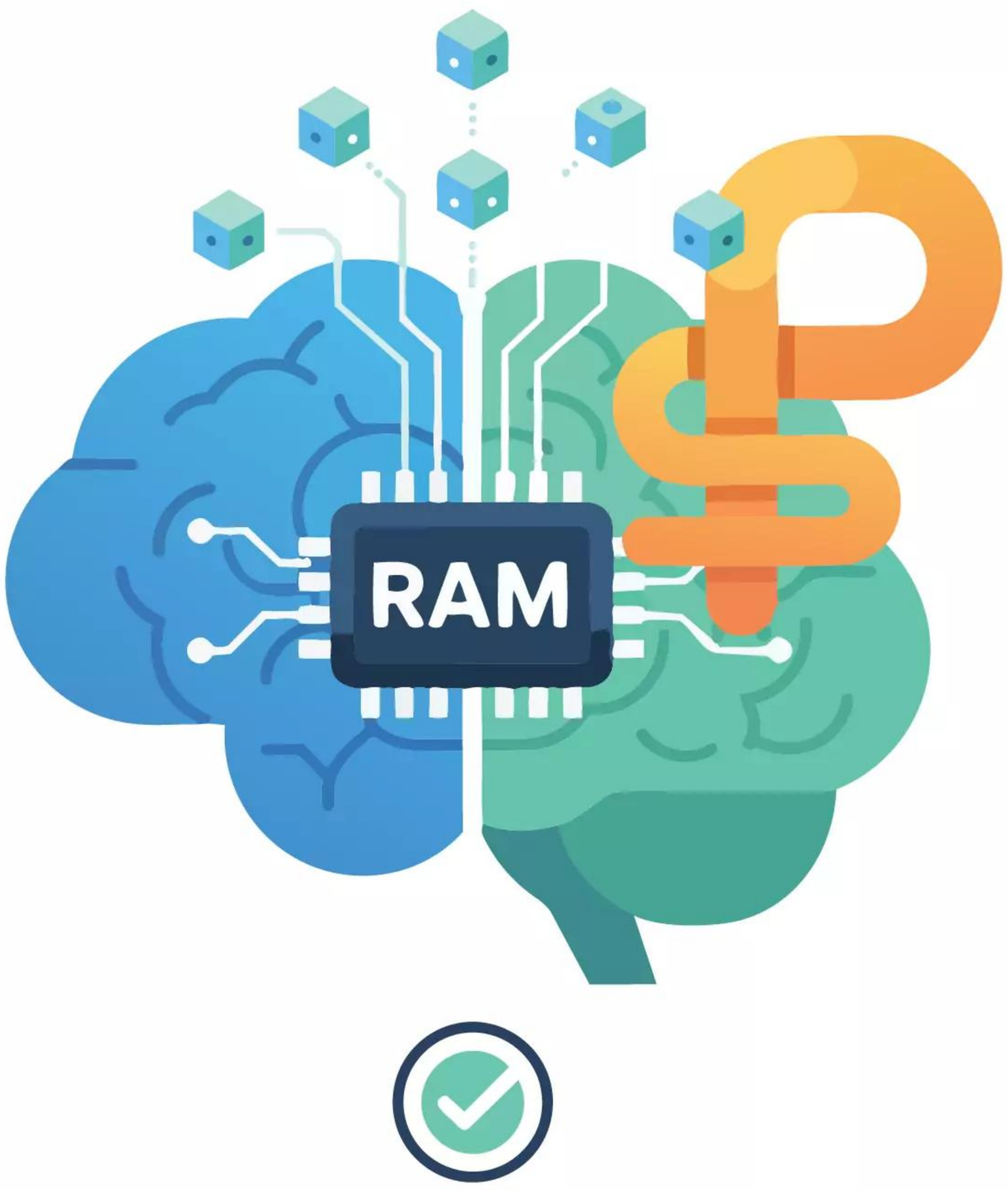
KNOW MORE / BUY NOW: efy.co.in/nfcrfid

Developed by the Electronics For You team, for you.

DEALERS: • Om Technology Centre, Bengaluru (Mob: +919845400864)
• Bharathi Electronics: Coimbatore (Mob: +919952228618)

Optimising RAM Usage with Python

Discover how we can make better use of RAM by applying various Python optimisation techniques.



In today's world, with the abundance of RAM available, we rarely think about optimising our code. But sooner or later, and before we can realise it, we hit the limits of the hardware.

Let's begin with how to measure the amount of RAM used by a Python program. There is always more than one way to do so and each has its merits.

Using *sys.getsizeof*

This is a built-in function from the sys module that returns the shallow size of a single Python object in bytes.

What it measures: It measures only the memory directly attributed to the object itself (e.g., the list structure), excluding the size of any referenced objects (e.g., the integers or strings inside a list).

The pros are:

- Simple, fast and lightweight.
- Useful for quick comparisons of basic object overhead (e.g., empty list vs small list).

Cons/limitations:

- Underestimates containers like lists, dicts, or custom objects with reference values in them.
- As it is not recursive, it doesn't help with complex structures or total memory footprint.

When to use: Use it for basic, one-off checks on simple objects or understanding Python's object overhead. It's not suitable for profiling real memory usage in applications.

```
import sys

def get_object_size_mb(obj):
    """
    Returns the size of a Python object in megabytes (MB).
    """
    size_bytes = sys.getsizeof(obj)
    size_mb = size_bytes / (1024 * 1024) # Convert bytes to
MB
    return size_mb

# create huge list
a = [i for i in range(5_000_000)]
size_of_list_mb = get_object_size_mb(a)
print(f"Size python object is: {size_of_list_mb:.2f} MB")

##OUTPUT
% /usr/bin/python3 getsizeof_sample.py Size python object is:
38.35 MB
```

Using *tracemalloc*

This built-in standard library module (since Python 3.4) traces Python memory allocations.

What it measures: It tracks memory blocks allocated by Python, providing detailed statistics (size, count, traceback) grouped by line, file, or traceback. You can take snapshots and

compare them to find leaks or differences.

The pros are:
- No external dependencies.
- Precise for Python-allocated memory (lower overhead than some tools).
- Excellent for debugging leaks; shows exactly where allocations happen.
- Supports peak/current traced memory and object-specific tracebacks.

Cons:
- Only tracks Python allocations (misses C extensions like NumPy unless they use Python's allocator).
- Requires starting tracing early (ideally at program start).
- More low-level; needs code to take/compute snapshots.

When to use: Use it when investigating memory leaks, finding allocation hotspots, or comparing memory before/after code sections. It's great for detailed, programmatic analysis in production-like debugging.

```
import tracemalloc

tracemalloc.start()

# create huge list
a = [i for i in range(5_000_000)]

current, peak = tracemalloc.get_traced_memory()
print(f"Current memory: {current / 10**6:.2f} MB; Peak: {peak
/ 10**6:.2f} MB")

tracemalloc.stop()

##OUTPUT
% /usr/bin/python3 tracemalloc_sample.py
Current memory: 180.21 MB; Peak: 180.21 MB
```

Using *memory_profiler*

This is a third-party package (pip install memory_profiler) for line-by-line memory profiling.

What it measures: Measures memory usage (typically Resident Set Size -- the portion of a process that is living in RAM right now) after each line in decorated functions, showing increments. Uses psutil as the default backend (optional switch to tracemalloc for more precise Python-only allocations).

The pros are:
- Easy line-by-line view (similar to line_profiler for time).
- Works cross-platform (Windows/Linux/Mac).
- Plots memory over time with mprof.
- Backend flexible (can switch between psutil and tracemalloc).

Cons:
- External dependency (and often needs psutil).
- Higher overhead due to frequent sampling of process memory.
- RSS is coarse and may overestimate memory (includes shared libs, OS allocations).
- Less precise than tracemalloc for pinpointing pure Python object allocations.

When to use: Good for quick, high-level overview of memory growth per line in functions/scripts. Ideal for exploratory profiling for scripts or functions where you only need to know which lines are causing memory jumps, not deep allocation traces.

```
from memory_profiler import memory_usage

def measure_ram(func):
     def wrapper(*args, **kwargs):
       mem_before = memory_usage()[0]
       result = func(*args, **kwargs)
       mem_after = memory_usage()[0]
       print(f"RAM used: {mem_after - mem_before:.2f} MB")
       return result
     return wrapper

@measure_ram
def sample_func():
     a = [i for i in range(5_000_000)]

sample_func()

##OUTPUT
% /usr/bin/python3 memory_profiler_sample.py
RAM used: 3.60 MB
```

Techniques for optimising memory usage

Effective data structure usage

Choosing the right data structure is one of the most important ways to reduce RAM usage and improve performance. Different structures store data differently and have different memory footprints.

Array vs list: Using a regular list to store many numbers is much less efficient in RAM than using an array object. More memory allocations occur in a list, each of which take time; calculations on larger objects are less cache-friendly and more RAM is used overall, making less of it available to other programs.

```
from memory_profiler_sample import measure_ram #Note: imported
function is from above sample
from array import array
# Create many integers
N = 5_000_000

@measure_ram
def sample_list():
     print(“List Creation”)
     data_list = list(range(N))

sample_list()

@measure_ram
def sample_arr():
     print(“Array Creation”)
     data_array = array(‘i’, range(N))

sample_arr()

##OUTPUT
% /usr/bin/python3 list_VS_arr_sample.py
List Creation
RAM used: 41.45 MB
Array Creation
RAM used: 19.16 MB
```

Tuple vs list: Tuples use less memory as they cannot be resized and no extra capacity is reserved. They are faster and more cache-friendly. The same is true of set vs dict.

Choose the smallest numeric type: When working with large datasets in Python, NumPy arrays are far more memory-efficient than pure Python lists or Pandas Series (which often default to higher-precision types). The key is to select the smallest dtype that can safely represent your data without overflow or loss of precision. Here are a few options: int64 uses 8 bytes, int32 uses 4 bytes, int16 uses 2 bytes.

Use sparse data structures: Use libraries like scipy.sparse or pandas.SparseArray when the data holds most values as zero -- storing all zeros does not make effective use of RAM. Hence sparse matrices store only non-zero entries.

Bytes vs Unicode

Python distinguishes between text (human-readable characters) and binary data (raw bytes — images, files, network packets, compressed data, encrypted data). Understanding the difference is essential for file handling, networking, APIs, and encoding issues. Unicode strings use more memory because character metadata needs to be stored. Bytes are compact as they use only 1 byte per element.

```
import sys

def get_object_size_mb(obj):
     """
     Returns the size of a Python object in megabytes (MB).
     """
     size_bytes = sys.getsizeof(obj)
     size_mb = size_bytes / (1024 * 1024) # Convert bytes to
MB
     return size_mb

# Create N string
N = 500_000_000

s = “hello “

# Unicode string (each character may take multiple bytes)
def sample_text():
     print(“Text Creation”)
     data_text = s*N
     print(f”Size is: {get_object_size_mb(data_text):.2f}
MB”)

sample_text()

# Bytes version (UTF-8 encoded)
def sample_bytes():
     print(“Bytes Creation”)
     data_text = s*N
     data_bytes = data_text.encode(“utf-8”)
     print(f”Size is: {get_object_size_mb(data_bytes):.2f}
MB”)

sample_bytes()

##OUTPUT
% /usr/bin/python3 text_VS_unicode_sample.py
Text Creation
Size is: 13351.44 MB
Bytes Creation
Size is: 4768.37 MB
```

Handling large text data stores

Handling massive text data (logs, documents, transcripts, crawled data, chat history, etc) requires storage formats and structures that are memory-efficient, compressed, and scalable.

Compression (gzip, bz2, zstd): Use of compression reduces the size of the text by 10-30X -- storing raw text wastes huge amounts of space.

mmap: Map a file (or part of it) directly into memory and treat gigabytes of data as a giant string or byte array. This is

super-fast for large files with low memory usage. This gives a bytes-like object that can be terabytes in size but uses almost no RAM until you touch the pages, resulting in no RAM explosion.

DAWG (Directed Acyclic Word Graph): A DAWG is a highly compressed data structure used to store large sets of strings efficiently. It is especially powerful for dictionaries, NLP vocabularies, autocomplete engines, and prefix-based searches.

```
import dawg

words = [“cat”, “car”, “cart”, “dog”, “doing”]

d = dawg.CompletionDAWG(words)

print(d.keys(“ca”)) # prefix search: [‘cat’, ‘car’, ‘cart’]
print(“dog” in d) # True
```

Tries: This refers to a tree-based data structure also known as a prefix tree or digital tree. It is used to efficiently store and retrieve a dynamic collection of strings by sharing common prefixes. Prefix-based string data structures play a crucial role in text indexing, NLP, search engines, auto-complete systems and large dictionary storage. Given below are a few commonly used structures.

Marisa-Trie (Matching Algorithm with Recursively Implemented StorAge): This static compressed trie uses LOUDS (Level-Order Unary Degree Sequence) representation.

It has an extremely small memory footprint (often 5-10% of original text), and supports predictive search (common prefix search) and exact-match very efficiently. Once built, the structure is read-only, hence not suitable for frequent insertions/deletions. The Python module is marisa-tri.

Double-Array Trie (DAT): The Double-Array Trie (DAT) is a compact Trie representation that uses two arrays:

a. BASE array determines offset for child transitions

```
base[i]: starting index for children of node i
```

b. CHECK array stores the parent index to validate transitions

```
check[i]: parent node of node i (for collision detection)
```

It has very fast transitions (just array indexing) and supports dynamic insertion/deletion (though slower than lookup). The Python module is datrie.

HAT-Trie (Hash-Array mapped Trie): A HAT-Trie combines a trie for prefix organisation and hash buckets for fast leaf-node lookups. It starts with a hash table at the top levels, then ‘bursts’ into small trie containers when buckets fill. It is extremely cache-efficient because burst containers are small and contiguous. It supports insertion, deletion, prefix search, and can store values/frequencies. The Python module is hat-trie

General tips

Avoid copies: Avoid making unnecessary copies of the objects, which will fill RAM with redundant data. Instead, use references or views from NumPy, or use iterators. Remember, for strings Python reuses identical immutable strings; this avoids storing the same string many times.

Profiling: Use the profiling methods to check memory usage while programming. These will help to detect early issues like excessive allocations, memory leaks, and inefficient structures.

Use generators instead of storing everything in memory: A generator yields one item at a time as we don’t need to store the entire dataset in RAM.

NumPy: While working with numeric data, use NumPy arrays as they offer many fast algorithms.

Bitarray: If your code requires lots of bit strings, both numpy and bitarray packages provide efficient representation of bits packed into bytes.

Micro Python: Good for working with embedded systems, its tiny memory footprint allows developers to write high-level, readable Python code while still running on hardware traditionally restricted to low-level C/C++. It brings much of the core Python 3 functionality to devices with extremely limited resources—often with as little as 16-256KB of RAM and flash storage.

Unit test suite: Even as you focus on optimisation do not forget the fundamentals — the purpose of the code. Make sure to have unit test suite in place before you make algorithmic changes.

Just to recap, profiling helps you see the problem; effective structures and techniques help you fix it. Together, they form a solid foundation for writing memory-efficient, high-performance Python programs. The general tips and practices help improve performance, reduce memory pressure, and make applications more scalable—especially when working with large datasets or in resource-constrained environments. END

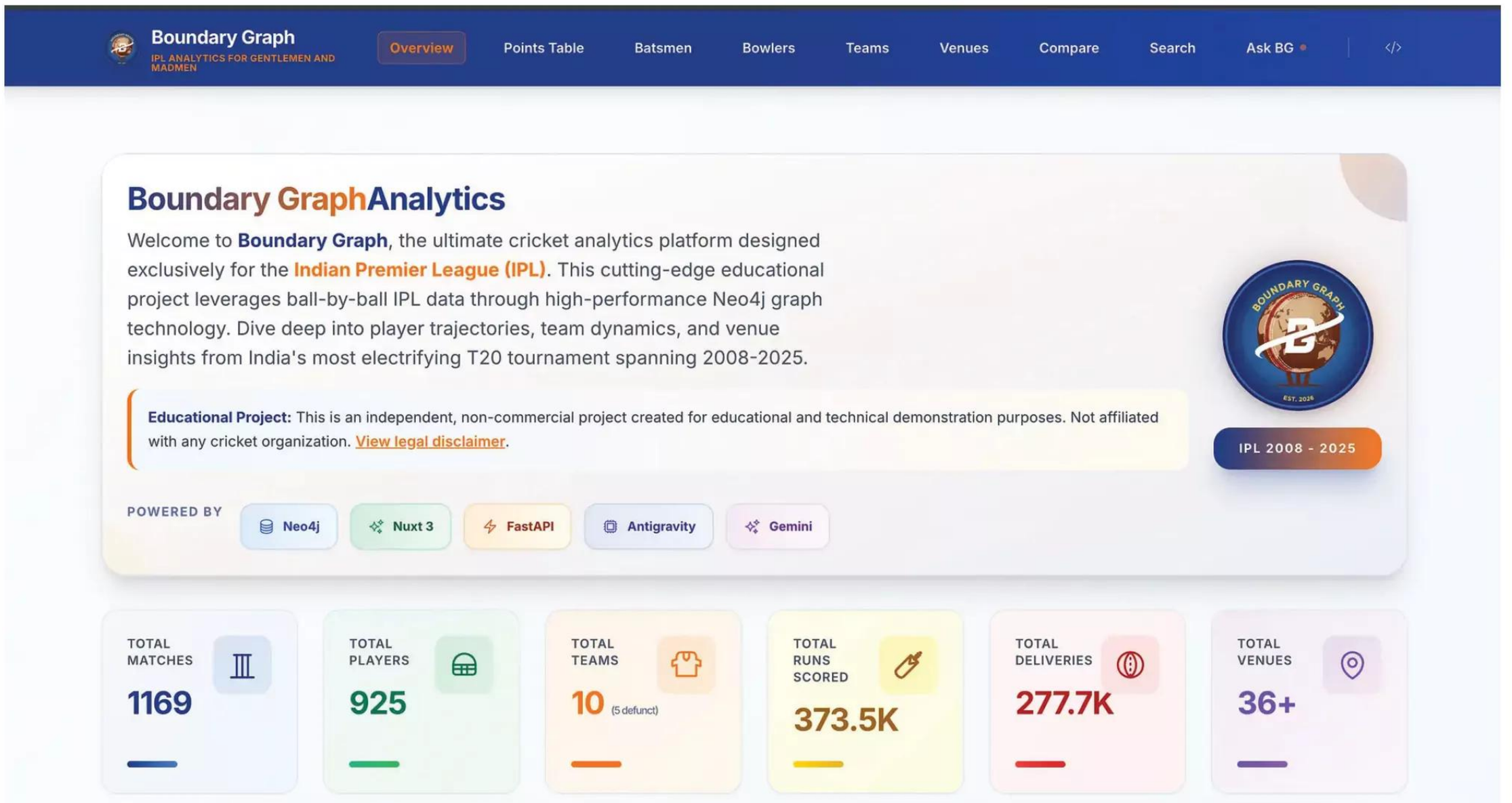
By: Thangaselvi Arichandrapandian

The author is a lifelong learner influenced by the quote:
“The more I learn, the more I realise how much I don’t know.”

— Albert Einstein.

Modelling the IPL as a Graph System

Performance analyses of cricketers and teams playing in the Indian Premier League (IPL) can benefit immensely from a graph model. Here's how...



Cricket analytics in the IPL (Indian Premier League) has traditionally evolved around summarisation. Batting averages, strike rates, economy numbers, and points tables provide convenient abstractions, but they compress a highly relational sport into static outcomes. Every performance in the IPL is shaped by a web of dependencies: the opposition, the batting order, the venue, team composition, and even the strategic intent of a particular season.

What often gets lost is continuity. Players move franchises, roles change, venues behave differently across years, and teams reinvent themselves every auction cycle. These are not isolated events. They are transitions, and transitions are fundamentally relational.

A graph model embraces this reality by making relationships explicit rather than inferred.

Why graph thinking fits the IPL

The IPL is not a dataset that sits still. Squads change annually, impact players alter match dynamics, and franchises experiment with long-term versus short-term strategies. Asking meaningful questions across seasons quickly becomes cumbersome in tabular systems because relationships must be reconstructed repeatedly.

Graph thinking allows analysis to begin from connections rather than records. It becomes possible to traverse how a player's career arcs across teams, how team cores evolve, or how venue conditions interact with playing styles.

This shift enables questions such as:

- How does a player's performance change when moving between franchises with different team cores?
- Which teams maintain structural stability across seasons despite roster churn?
- How strongly do venues influence specific player archetypes over time?

These are difficult to express cleanly without a relationship-native model.

Structuring the IPL as a graph

The graph is built around stable entities that rarely change identity:

- Players representing individual cricketers
- Teams representing IPL franchises
- Matches as discrete competitive events

- Seasons capturing tournament editions
- Venues representing stadiums

These nodes intentionally remain lightweight and descriptive rather than analytical.

Relationship first modelling

The analytical power of the model comes from where the data lives. Instead of embedding performance metrics into nodes, they are attached to relationships where context exists.

Examples include:

- Runs scored and strike rate in the player to match relationship
- Overs bowled and wickets in the bowler to match relationship
- Team membership in a player to team relationship in a season

This approach preserves temporal correctness and prevents historical distortion when players change teams or roles.

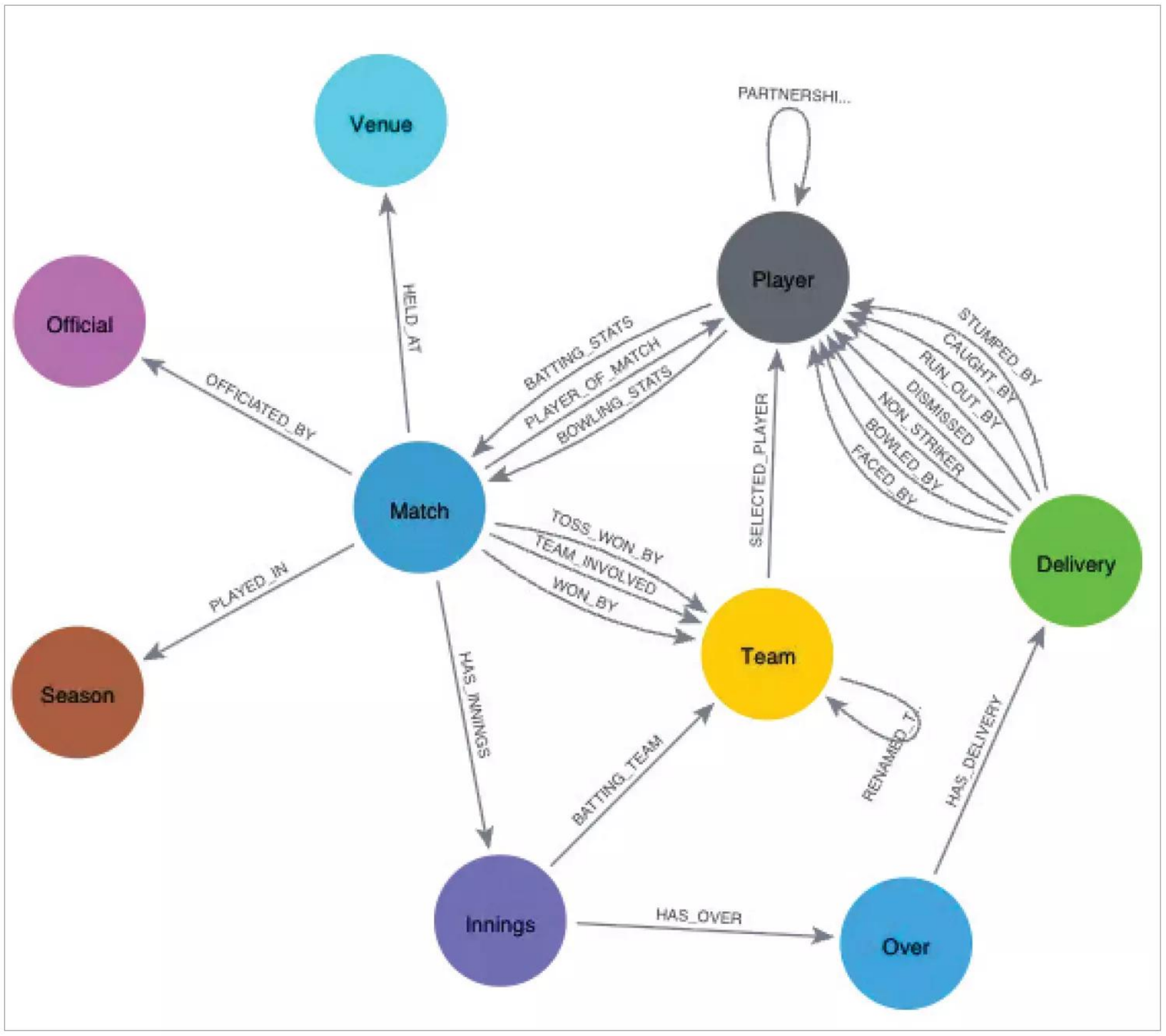


Figure 1: Relationship first modelling

Graph scale and infrastructure

The final Neo4j deployment contains more than 320,000 nodes and over 1.2 million relationships spanning multiple IPL seasons. At this scale, the graph reflects real tournament complexity while remaining efficient for deep traversal queries. Indexing and relationship-centric access patterns allow multi-hop queries without the join explosion typical of relational systems.

Once the IPL is represented as a graph, analysis shifts away from ranking tables and towards patterns of interaction.

Player adaptability across contexts

Rather than measuring success purely through aggregate runs or wickets, players can be evaluated on how they perform across varying conditions. Traversing player to team to venue paths reveals who adapts consistently versus who thrives only in specific environments.

Team cohesion and continuity

Team performance can be analysed through relational stability. Teams that retain a coherent player core across seasons often exhibit stronger structural signals than teams built around frequent overhauls. These patterns are visible in the density and persistence of player team relationships rather than in match results alone.

Identifying structural influencers

Graph algorithms introduce another dimension of insight. Centrality measures highlight players who act as connectors across teams and seasons. These players often influence tournament dynamics in ways not immediately visible through traditional statistics. Community detection surfaces clusters of players and venues that share similar performance characteristics, offering a data-driven view into tactical groupings.

Venues as contextual forces

Venues in the IPL are often reduced to simple labels, yet their influence is persistent and measurable. When modelled as first class entities, venues reveal how conditions shape performance over time rather than in isolated matches.

Sub-graphs anchored to specific venues expose:

- Players whose performance consistently improves or degrades at certain grounds
- Teams that rely heavily on home conditions for sustained success
- Venue-specific biases towards batting depth or bowling discipline

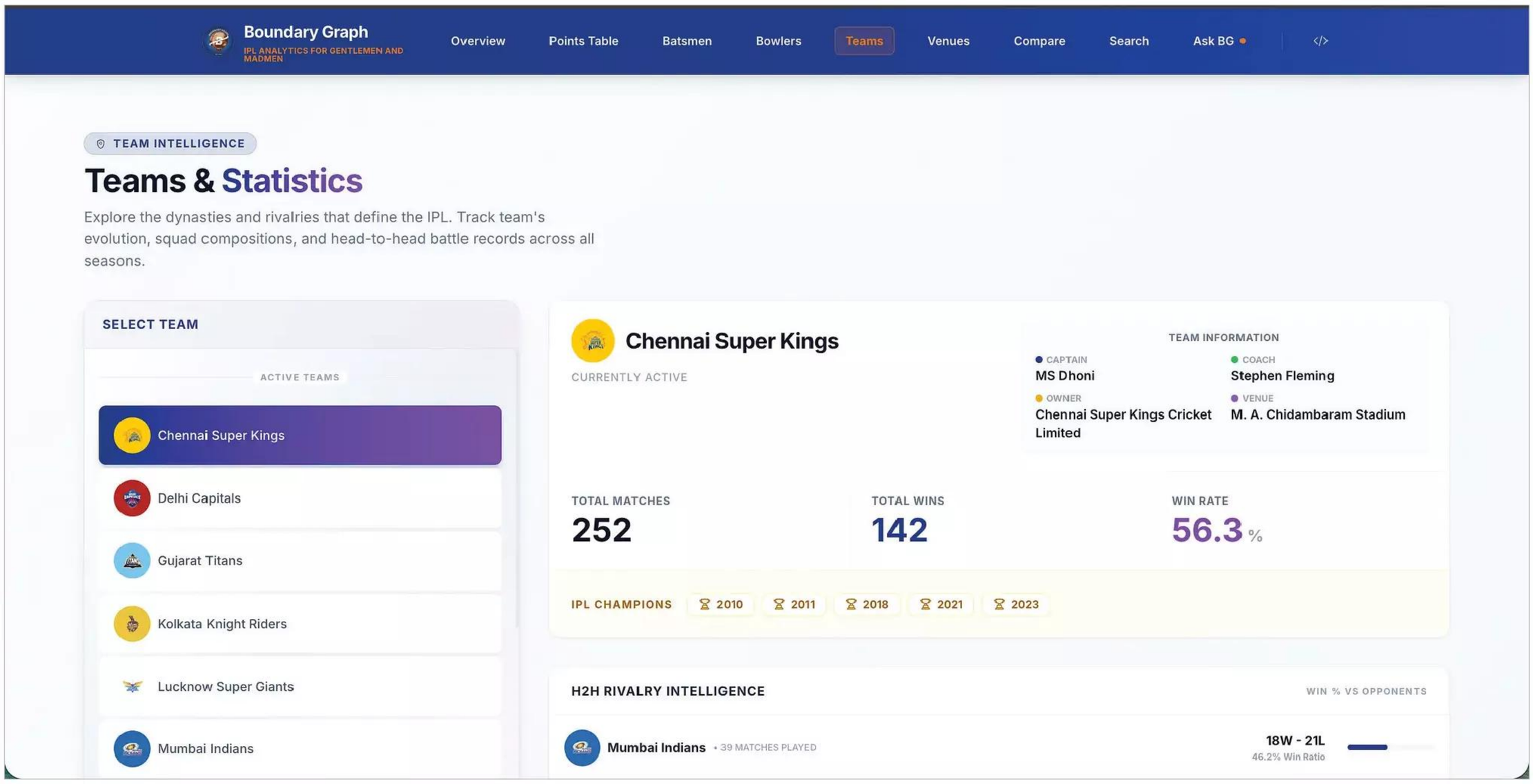


Figure 2: Teams and statistics

This shifts venue analysis from anecdotal commentary to structural evidence rooted in historical interaction patterns.

From scorecards to structures

Viewing the IPL through a graph lens reframes analysis from scorecards to structures. It enables reasoning that mirrors how the game unfolds as a network of evolving relationships shaped by context, continuity, and interaction. With hundreds of thousands of nodes and over a million relationships, the system is designed not just for historical analysis but for continuous evolution. The graph is updated as soon as a match concludes, allowing new performances, outcomes, and contextual shifts to be reflected immediately in the data model.

This foundation also enables more intuitive ways of interacting with the graph. An upcoming feature, 'Ask BG', introduces a natural language interface on top of the graph, allowing users to ask relationship-driven questions directly without writing queries. By combining real-time data updates with conversational access to the underlying graph, the platform moves closer to treating IPL analysis as a living system rather than a static archive, unlocking insights that traditional approaches struggle to surface.

Explore the boundary graph

The complete graph model and interactive exploration experience are available through the Boundary Graph application at *https://boundary-graph.netlify.app/*. The app allows readers to navigate players, teams, venues, and seasons through relationship-driven queries and visualisations, offering a deeper and more intuitive understanding of the IPL beyond tables and aggregates.

Beyond the boundary rope

Although the IPL provides a familiar and engaging dataset, the modelling principles demonstrated here extend far beyond sports analytics. Any domain where outcomes depend on interactions rather than isolated attributes benefits from graph-first thinking.

Common examples include:

- Recommendation engines built on user item interaction paths
- Fraud detection systems analysing transactional networks
- Knowledge graphs capturing evolving semantic relationships

The IPL serves as a compelling illustration of how graph models scale from theory to production.

Note: If you need more details and are unaware about the IPL cricket tournament, do visit the official website at *https://www.iplt20.com/*.

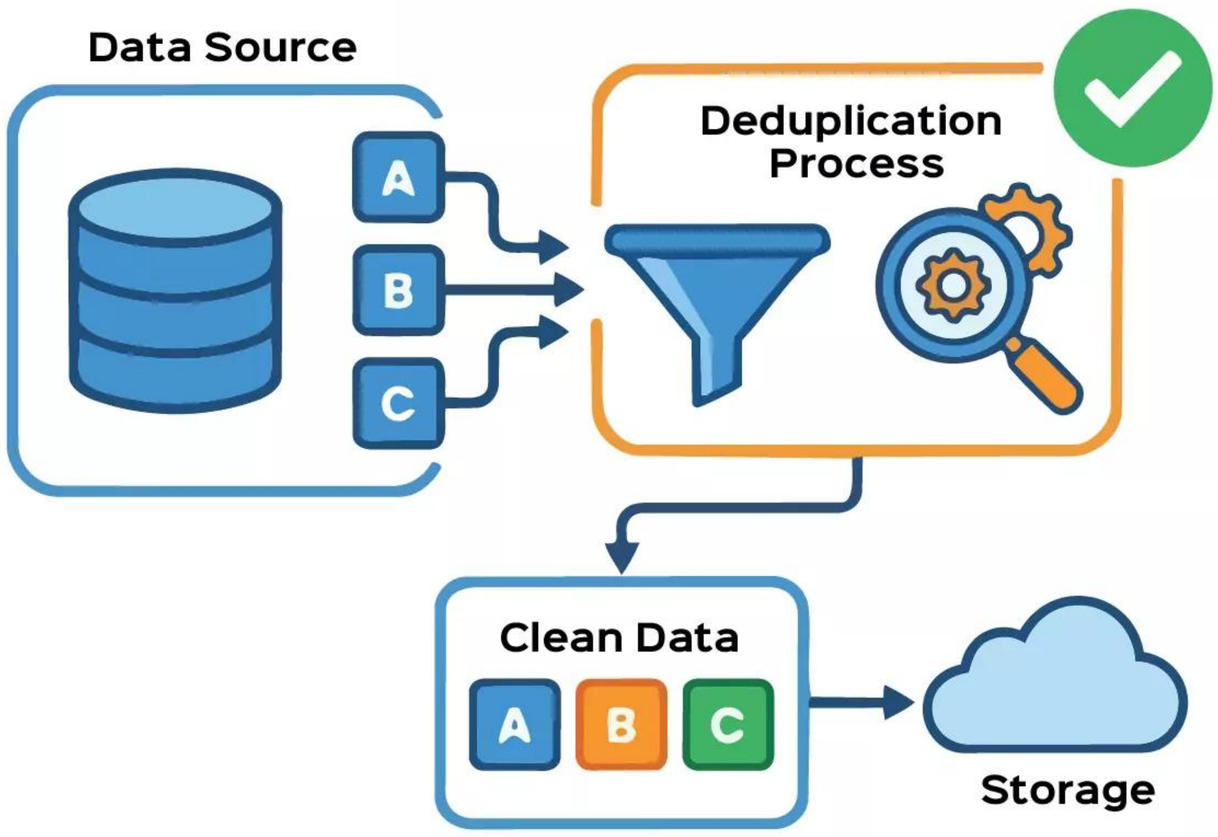
END

By: Siddhant Agarwal

The author is a seasoned DevRel professional and speaker. He has over a decade of experience in innovating and scaling developer ecosystems across the globe. He specialises in making even the most complex concepts relatable and engaging.

Data Deduplication Done the Right Way

Deduplication helps to save space on Linux-based storage systems. Choose the right platform and check whether it meets your goals.



Stop storing the same data a thousand times! If you snapshot the same virtual machine every night for a month, you're not really storing 30 versions but the same blocks repeatedly. That invisible redundancy quietly eats disks, backup windows, and your budget. Data deduplication is simply a way of saying: "Store each unique piece of data once and reuse it everywhere it appears."

On Linux, you don't need proprietary software to get there. Between modern filesystems like Btrfs and ZFS, block-layer solutions like dm-vdo, and user-space tools built around clever chunking algorithms, you can retrofit serious deduplication into an otherwise boring stack.

What exactly is deduplication doing?

Deduplication works in two big steps: first it decides how to slice your data, then it decides what is already there.

- *Chunking*: Data is split into fixed-size or content-defined chunks. Fixed-size chunks are simple, but content-defined chunking (CDC) uses patterns in the data stream to choose boundaries and is more robust when bytes are inserted or deleted in the middle of a file.
- *Fingerprinting*: Each chunk gets a fingerprint, typically a cryptographic hash, that acts like its identity card.
- *Indexing*: Fingerprints are stored in an index. When a new chunk arrives, the system checks if its fingerprint already exists.
- *Reuse*: If it's a duplicate, the system creates or updates references instead of writing the chunk again.

Inline deduplication runs this logic on the write path; data is inspected before hitting disk, while post-process schemes run a scanner later, when the system is quiet. Inline gives instant savings but increases CPU, memory, and I/O overhead. Post-process keeps writes fast and shifts the cost into scheduled jobs.

Linux has more dedup options than most people think

Linux doesn't have one dedup switch. Instead, you pick the layer that fits your problem.

- ***Btrfs: dedup-friendly copy-on-write***

Btrfs (B-tree filesystem) is a modern, copy-on-write (CoW) filesystem for Linux designed for high scalability, reliability, and advanced storage features. Developed to handle large data volumes, it features built-in snapshots, data compression, and integrated volume management (RAID 0, 1, 10). Btrfs is a copy-on-write filesystem, so it is already great at not duplicating blocks when you clone or snapshot data. Features like reflinks (cp --reflink) let you create instant logical copies that share blocks until they diverge.

But Btrfs does not do automatic inline deduplication. Instead, you run user-space tools that:

- Scan the filesystem, compute hashes of extents or blocks
- Look for duplicates in a database
- Ask Btrfs to 'merge' them by turning separate extents into shared ones

Two popular options are:

- *duperemove*: Batch scanner plus hash database; good for periodic dedup on backup volumes.
- *bees*: A long-running daemon that incrementally walks the filesystem tree and deduplicates in the background.

Admins report high savings on collections of VMs, container layers, and ISO trees, but also warn that heavy dedup plus fragmentation can slow down random reads if you are aggressive on a busy volume.

- ***ZFS on Linux: Inline dedup with a big brain***

OpenZFS is an open source, advanced storage platform that combines a filesystem and a volume manager, offering features like data integrity, snapshots, compression, deduplication, and built-in RAID (RAID-Z) for data protection and efficient storage, available across Linux, FreeBSD, macOS, and other systems. It focuses on reliable, self-healing data management. OpenZFS offers native, inline block-level deduplication.

Every write goes through this:

- Hash the block.
- Look it up in the dedup table (DDT).
- If present, bump a reference count instead of writing new data.

The good part is that you don't have to run external scanners. The risky part is that the dedup table lives largely in RAM, and its size scales with unique blocks, not logical size. People running ZFS for VM boot disks often see 3:1 to 5:1 dedup ratios, but they also plan memory carefully, and think gigabytes of RAM per few terabytes of unique data, depending on block size and workload.

Recent work in the ZFS community has made dedup smarter and less of a trap:

- Age-based pruning of old unique entries
- 'Opportunistic' dedup modes that try to use spare resources without committing to huge DDT footprints

ZFS dedup shines on carefully scoped pools — for example, a pool dedicated to VM images or template-based workloads, rather than as a blanket setting on every dataset.

- ***dm-vdo: dedup below the filesystem***

The dm-vdo (Virtual Data Optimizer) device-mapper target sits below your filesystem and offers inline block-level deduplication, compression, zero-block elimination, and thin provisioning.

Its design splits responsibilities:

- A dedup index (UDS) for quickly recognising duplicate blocks
- A reference-counted data store mapping logical to physical blocks

Because VDO lives at the block layer, it is filesystem-agnostic: you can run XFS, ext4, or Btrfs on top and still benefit. That's useful in environments where changing filesystems would be too disruptive but storage savings are urgent.

Research systems and page-cache aware designs

Academic and experimental work pushes Linux dedup further:

- *Dmdedup*: A kernel device-mapper target focused on primary storage dedup, exposing a flexible platform to test algorithms at block level.
- *Dual-dedup and related designs*: Schemes that share duplication knowledge with the page cache so the kernel avoids caching multiple copies of the same on-disk data, improving read performance.
- *Hybrid inline/out-of-line systems (e.g., RevDedup and successors):* Use coarse inline dedup for new backups, then offline 'reverse deduplication' to shift fragmentation onto old data and keep recent backups sequential and fast to read.

These ideas increasingly influence production stacks even when the exact prototypes are not deployed as-is.

Why dedup can be both a hero and a villain

On paper, deduplication looks like free capacity. In practice, it's a trade between space, speed, and complexity.

The good news

- *Backups*: Studies and vendor reports routinely show 60–80% logical space reduction on backup workloads because daily or weekly backups share large regions with previous ones.
- *VDI and VMs:* Clusters of similar virtual machines share kernels, libraries, and base images, often yielding 3:1 or better dedup ratios.
- *Network and cloud:* Client-side dedup in backup tools avoids sending duplicate data at all, shrinking WAN usage and enabling longer retention on the same hardware.

The less glamorous side

- *Fragmentation*: When many logical blocks map to a patchwork of shared physical blocks, sequential reads can degrade to random I/O. That hurts HDD-based arrays and cold storage particularly hard.
- *Metadata and RAM*: Dedup indices grow with the number of unique chunks. If they outgrow RAM, lookups spill to disk and performance plummets.
- *Encryption*: Application-level encryption (encrypt before writing to disk) effectively destroys dedup because identical plaintexts turn into different ciphertexts. Encrypted dedup requires special schemes where ciphertexts remain comparable.

A good rule of thumb: dedup is fantastic for backup and image-like data, risky for latency-sensitive primary databases, and mostly irrelevant for already-compressed or heavily encrypted payloads.

A practical playbook for Linux admins

Instead of treating deduplication as a magic tick-box, think of it as a workload-specific tool. Here is a pragmatic approach you can adapt.

Identify your 'low-hanging fruit':

Look for:

- Large backup repositories with many similar full backups
- VM image libraries, golden images, and lab snapshots
- ISO mirrors, container base layers, code repositories with many branches

These workloads naturally contain lots of duplicate blocks and are not usually as latency sensitive as transactional databases.

Choose the right layer: If already on Btrfs for backup disks, use *duperemove* weekly or *bees* continuously on that subvolume. Start with a test directory to measure space savings and runtime.

If you want filesystem-agnostic dedup on a SAN or local RAID, evaluate *dm-vdo* on a new test LUN, layer XFS or ext4 on top, then replay a typical backup workload and compare logical vs physical use.

If you are running ZFS for VM pools, enable dedup only on datasets that contain highly similar data. Monitor DDT size, RAM use, and read latency over a few weeks before scaling out.

Combine dedup with smart backup design: Hybrid designs from systems like RevDedup show that you don't have to choose between performance and savings.

A simple adaptation for Linux is:

- Take regular snapshots on a CoW filesystem (Btrfs/ZFS).
- Deduplicate older snapshots more aggressively, leaving the newest generation relatively unfragmented.
- When you prune snapshots, prefer to delete heavily fragmented, older generations.

This pushes fragmentation cost towards data that is rarely read and keeps recent backups fast.

Watch the numbers that actually matter: Track:

- Logical vs physical usage (your effective dedup + compression ratio)
- Average and tail latencies for backup and restore jobs
- Dedup index size and memory footprint (DDT size for ZFS, index size for VDO, database size for duperemove)

If your dedup ratio is below ~1.2:1 and you are paying a heavy performance or complexity tax, it may not be worth it for that workload.

Where Linux dedup is heading

Recent research and kernel work hints at where Linux storage is going next:

- Faster content-defined chunking (e.g., RapidCDC and newer CDC variants) that keep high dedup ratios while slashing CPU cost.
- Inline dedup optimised for non-volatile memory and SSDs, where random I/O penalties are smaller and metadata can live in persistent memory.
- Container-aware dedup engines that understand Docker or OCI layer semantics and avoid redundant copies across thousands of images without killing pull performance.

For practitioners, this means dedup will increasingly feel less like an expensive experiment and more like a predictable building block, especially in backup, VM, and container-image platforms. END

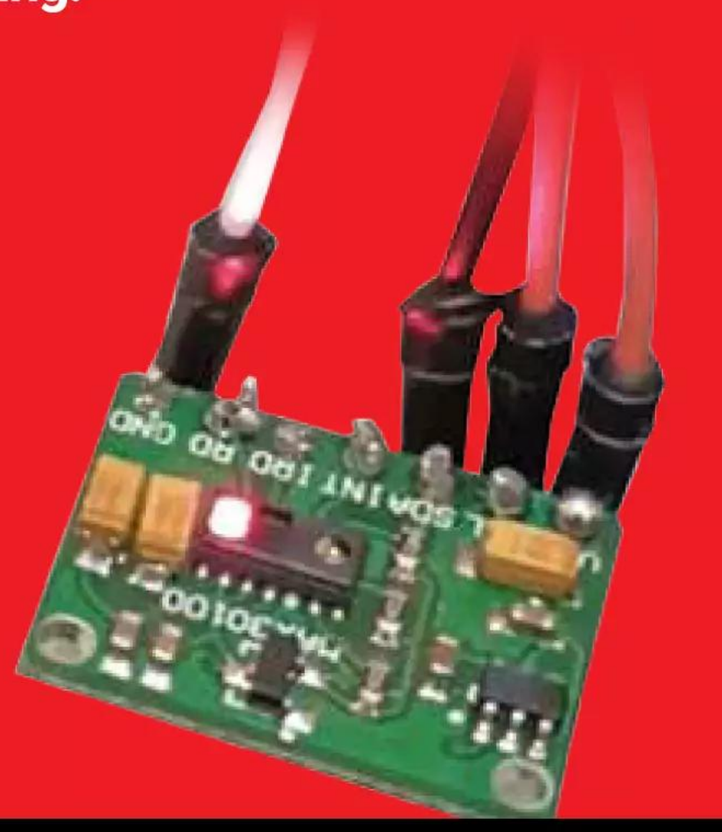
By: Mohamed Abubakkar S.

The author is an associate engineer at Presidio AI and a cloud enthusiast with deep interest in open source.

IOT DIY KIT: Health Tech & Medical Devices

Want to create exciting IoT projects related to medical electronics, medical devices, health or wellness tech?

This DIY kit lets you build smart IoT projects with WiFi connectivity and built-in sensors for monitoring, alerts, and tracking.



POWERED BY



IndusBoard COIN

Super small sized development board that enables you to create solutions for many use-cases where small size is super important.

SPECIAL INTRO PRICE

₹2,049 (Inclusive of GST)

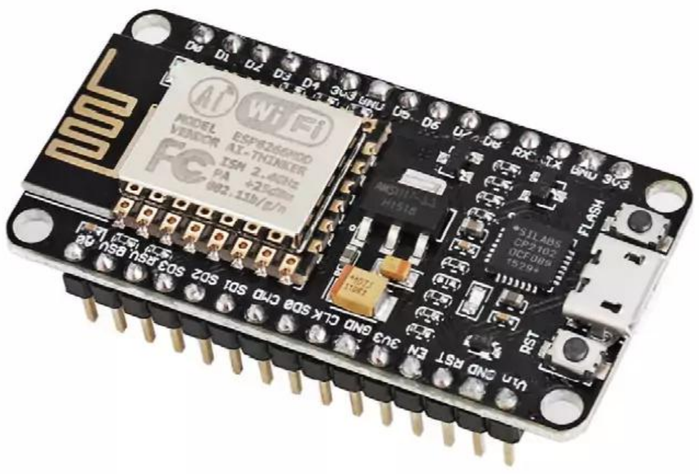
KNOW MORE / BUY NOW: efy.co.in/healthiotkit

Developed by the Electronics For You team, for you.

DEALERS: • Om Technology Centre, Bengaluru (Mob: +919845400864)
• Bharathi Electronics: Coimbatore (Mob: +919952228618)

Sending IoT Sensor Data to Public or Private Servers

This IoT system shows a simple and effective way to send sensor data using an ESP8266 microchip.



Modern IoT systems rely on sensors to collect real-world data and transmit it to public cloud servers or private on-premises servers for monitoring, analytics, and decision-making. Here's the core working logic of an ESP8266-based IoT node that reads temperature and humidity data and sends it securely to a remote server while also exposing the data locally.

Overview of the system architecture

The code implements a hybrid, dual-mode architecture for an IoT device (like an ESP8266 microcontroller) designed for maximum flexibility in different scenarios. The system doesn't just do one thing; it intelligently operates in two distinct modes, making it useful both when connected to the internet and when operating locally.

Remote data push (Client mode): Sensor data is periodically sent to a private/public server using HTTP POST (JSON format). This is ideal for cloud dashboards, databases, or analytics pipelines.

Local data access (Server mode): ESP8266 runs a lightweight web server. Nearby devices can fetch live sensor data directly using a browser or REST call.

This hybrid model is commonly used in smart homes, industrial monitoring, and edge computing systems.

We will now see how this IoT system works.

➢ Library and hardware abstraction is as follows:

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include "DHT.h"
```

These libraries provide:

- Wi-Fi connectivity (ESP8266WiFi)
- HTTP communication (ESP8266HTTPClient)
- Low-level TCP client support (WiFiClient)
- Sensor abstraction for the DHT11 temperature and humidity sensor

The abstraction ensures that networking and sensor handling remain clean and modular.

➢ Device identity and network configuration is:

```
const char* deviceName = "mohit2";
const char* ssid = "Jiowisdom";
const char* password = "Mo980616";
```

- *deviceName* uniquely identifies the IoT node on the network and server.
- *ssid* and *password* allow the ESP8266 to connect as a Wi-Fi station (STA mode).

In production systems, credentials are usually stored securely or provisioned dynamically.

➢ Remote server configuration (public/private) setup is:

```
const char* serverHost = "13.233…..";
const int serverPort = 5002;
const char* serverPath = "/data";
```

This allows:

- Sending data to a cloud VM, private LAN server, or edge gateway.
- A Flask or Node.js backend can easily parse incoming JSON.

This design is server-agnostic, meaning the ESP8266 does not care whether the server is public or private.

➢ Sensor initialisation and timing strategy is:

```
const int DHTPin = 14;
DHT dht(DHTPin, DHTTYPE);
            const unsigned long
uploadInterval = 60000;
```

- The DHT11 sensor is connected to GPIO14.
- Data is uploaded once every 60 seconds.
- Timing is controlled using *millis()* instead of *delay()* to keep the system responsive.

This approach follows non-blocking embedded design best practices.

➢ Wi-Fi connection and device boot sequence is as follows:

```
WiFi.hostname(deviceName);
WiFi.begin(ssid, password);
WiFi.mode(WIFI_STA);
During setup():
```

ESP8266 connects to Wi-Fi. It assigns a hostname for easy

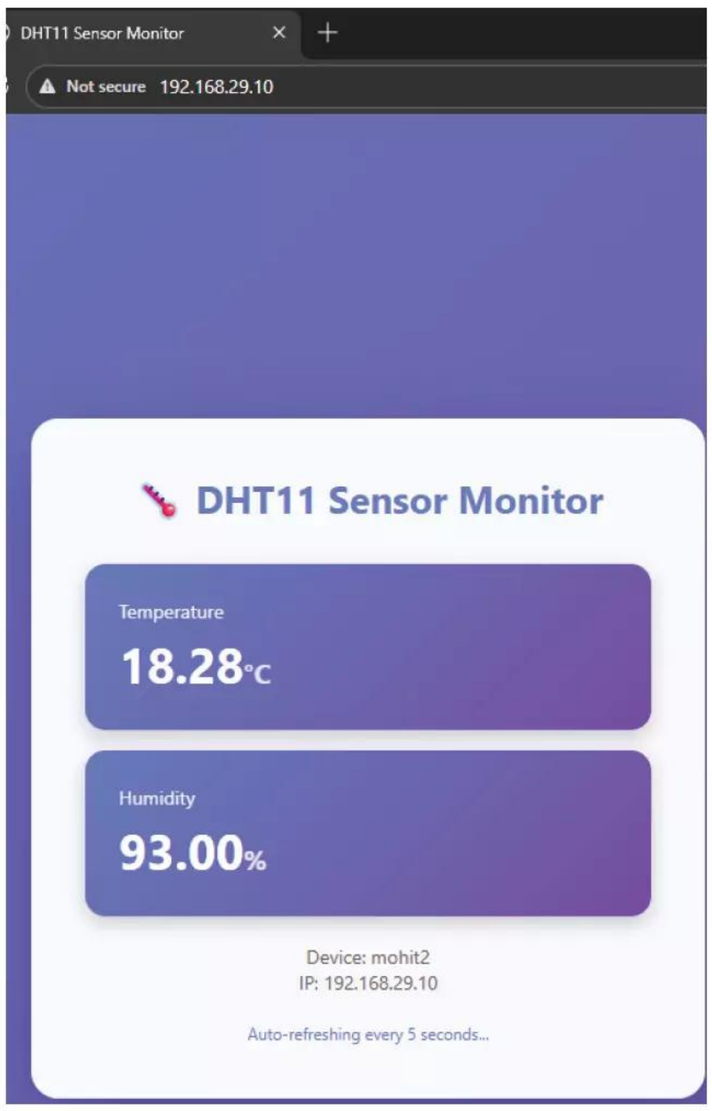


Figure 1: Local server temperature and humidity sensor

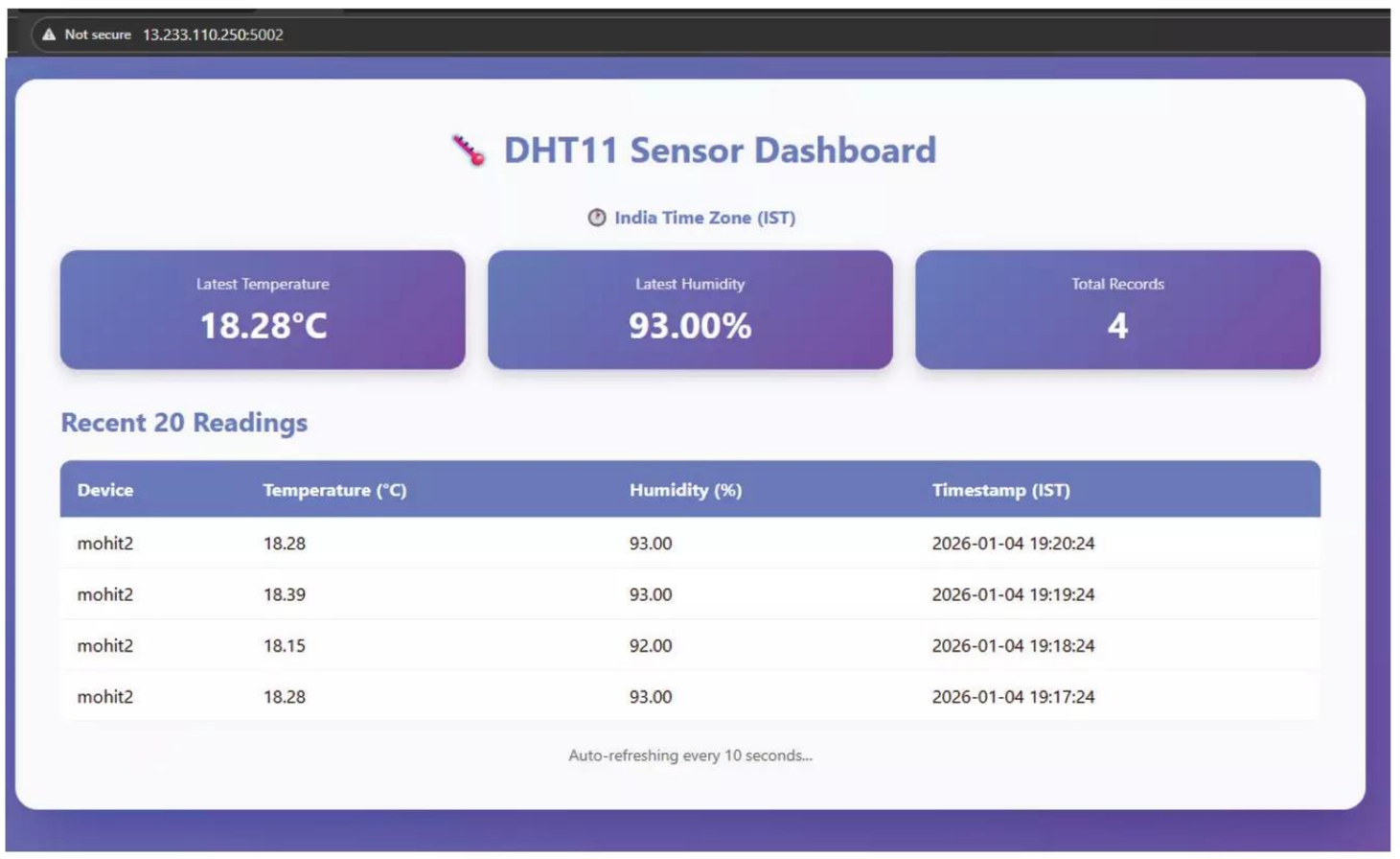


Figure 2: Public server, which can be accessed from any location

identification and starts a local web server on port 80.

Once connected, the device prints its local IP address, which can be accessed from any browser on the same network.

To send sensor data to the remote server (core logic), use the following code:

```
void sendDataToServer(float temp, float humidity)
```

Key operations inside this function are:

- *URL construction*

```
String url = "http://" + serverHost + ":" + serverPort + serverPath;
```

Creates a complete endpoint dynamically, allowing easy server changes.

- *JSON payload creation*

```
{
  "device": "mohit2",
  "temp": 27.45,
  "humidity": 61.20
}
```

JSON is lightweight, human-readable, and widely supported, perfect for REST APIs and cloud ingestion pipelines.

- *HTTP POST transmission*

```
http.POST(jsonPayload);
```

Sends sensor data securely over HTTP. The server can store data in a database, trigger alerts, or feed dashboards.

This pattern is widely used in IoT cloud platforms and private telemetry systems.

➢ Periodic sensor reading loop is:

```
if (currentTime - lastUploadTime >= uploadInterval)
```

Inside loop(): reads temperature and humidity from DHT11. It validates sensor data (!isnan) and sends data to the remote server at fixed intervals.

This ensures controlled network usage, predictable data sampling, and reliable long-term operation.

➢ Local web server for real-time monitoring is:

```
WiFiServer server(80);
```

The ESP8266 also acts as a mini HTTP server.

When a client connects:

- Sensor data is read instantly.
- Response is returned in JSON format:

```
{
  "temp": 28.10,
  "humidity": 60.50
}
```

This matters because it:

- Enables offline or LAN-only monitoring.
- Is useful for diagnostics, maintenance, or edge applications.
- Reduces dependency on cloud connectivity.

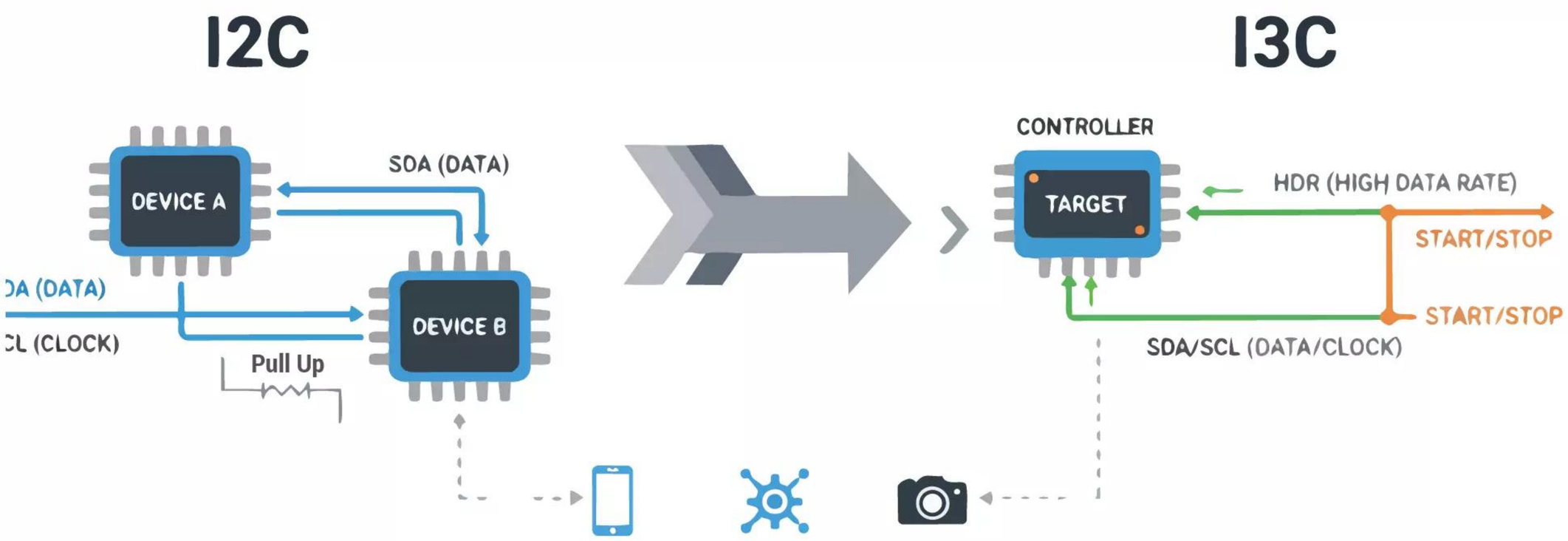
This IoT system can work in both online and offline conditions, making it reliable and flexible. The use of fixed time intervals and simple JSON data makes it easy to manage and understand. This approach is useful for smart homes, small industries, and basic monitoring systems. END

By: Dr Mohit

The author is a CEH and ECSA, and has worked at IBM, Teramatrix and Sapient. He has written the books titled 'Python Penetration Testing Essential', 'Python: Penetration Testing For Developers', and 'Learn Python in 7 Days'.

I2C and I3C: How Modern Devices Communicate

I3C and I2C are both two-wire communication protocols that help exchange data between multiple devices. While I3C preserves the simplicity of I2C, it introduces new features suited for today's sensor-rich devices.



Inside every electronic system, from smartphones and laptops to massive data centre servers, there are many small components or peripherals that must communicate with a central controller. These include sensors that monitor temperature and voltage, EEPROMS that store configuration data, and power management ICs.

All these peripherals work together to keep the system running smoothly. However, connecting each component with its own set of wires would make the board complex, messy and costly. To overcome this, engineers use shared communication buses, where devices use the same wires and take turns to communicate.

One of the simplest and most used protocols for this purpose is I2C (inter-integrated circuit) — a two-wire communication protocol to exchange data between multiple devices.

In server environments, I2C plays an essential role inside the baseboard management controller (BMC), which is a dedicated chip responsible for monitoring the system health, reading sensors, managing power rails, and more. I2C acts as the communication path behind the BMCs to monitor and control different parts of a running server.

How I2C works

I2C is a bus interface connection protocol incorporated into devices for serial communication. It uses only two bi-directional open-drain lines (as shown in Figure1):

- SDA (serial data line), which carries the actual data bits
- SCL (serial clock line), which carries the clock signal

Each device connected to these lines acts as either a master (which initiates communication) or a slave/target (which responds).

A device can only pull the bus line to LOW (0), and it never drives the line HIGH (1). When the device releases the line, an external pull-up resistor pulls the line up to HIGH (as shown in Figure 2).

So, each device has two states for the bus pin:

- *Drive LOW:* Turn ON the transistor → line connects to GND → 0
- *Release:* Turn OFF the transistor → line floats → pull-up resistor makes it 1

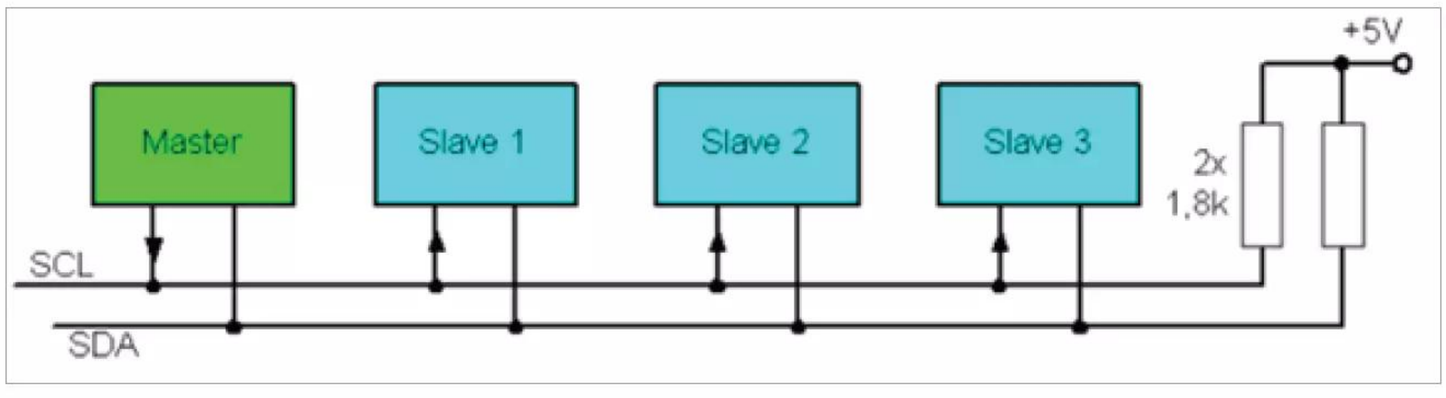


Figure 1: I2C connection

Open-Drain

This open-drain signalling ensures safe, collision-free communication:

- Any device can pull the line LOW.
- No device ever forces it HIGH.
- If two devices send data at once, the LOW signal always dominates, preventing conflict.

This design enables two powerful features:

Arbitration: Multiple masters can attempt communication without corrupting data.

Clock stretching: Slower devices can hold the clock LOW to pause communication until they're ready.

This is critical in a server motherboard, where various controllers (such as the CPU's power controller and BMC) may share the same I2C bus. Without open-drain logic, they could interfere with each other and destabilise system monitoring.

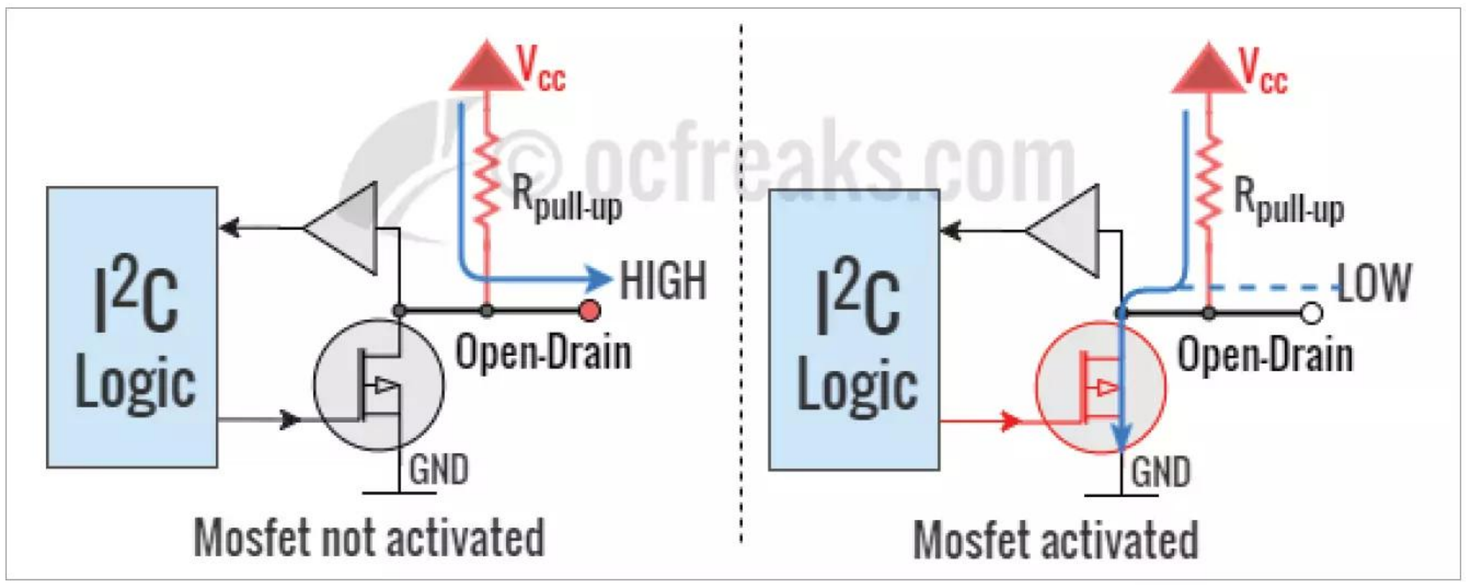


Figure 2: Open-drain in I2C (Image Source: *https://www.ocfreaks.com/i2c-tutorial/*)

SCL line

This is how the SCL line works in I2C communication:

- ***Master pulls SCL LOW*:** The master actively pulls the SCL line LOW to begin a clock cycle.
- ***While SCL is LOW, the transmitter sets the data (SDA):*** The device transmitting the data (master or slave) drives the SDA line to either logic 0 or logic 1.
- Data is allowed to change only when the SCL line is LOW.
- ***Master releases SCL (floating state):*** The master stops pulling SCL LOW and releases the line.
- ***Pull-up resistor pulls SCL HIGH:*** Since no device is pulling the line LOW, the pull-up resistor drives SCL HIGH.
- ***When SCL is HIGH, the receiver samples SDA:*** The receiving device reads the SDA line while SCL is HIGH. The data must remain stable during this period.

Steps of I2C data transmission

The I2C follows the steps given below to transmit data (as shown in Figure 3):

Start condition: The master device sends a start condition by pulling the SDA line low while the SCL line is high. This signals that a transmission is about to begin.

Addressing the slave: The master sends the 7-bit address of the slave device it wants to communicate with,

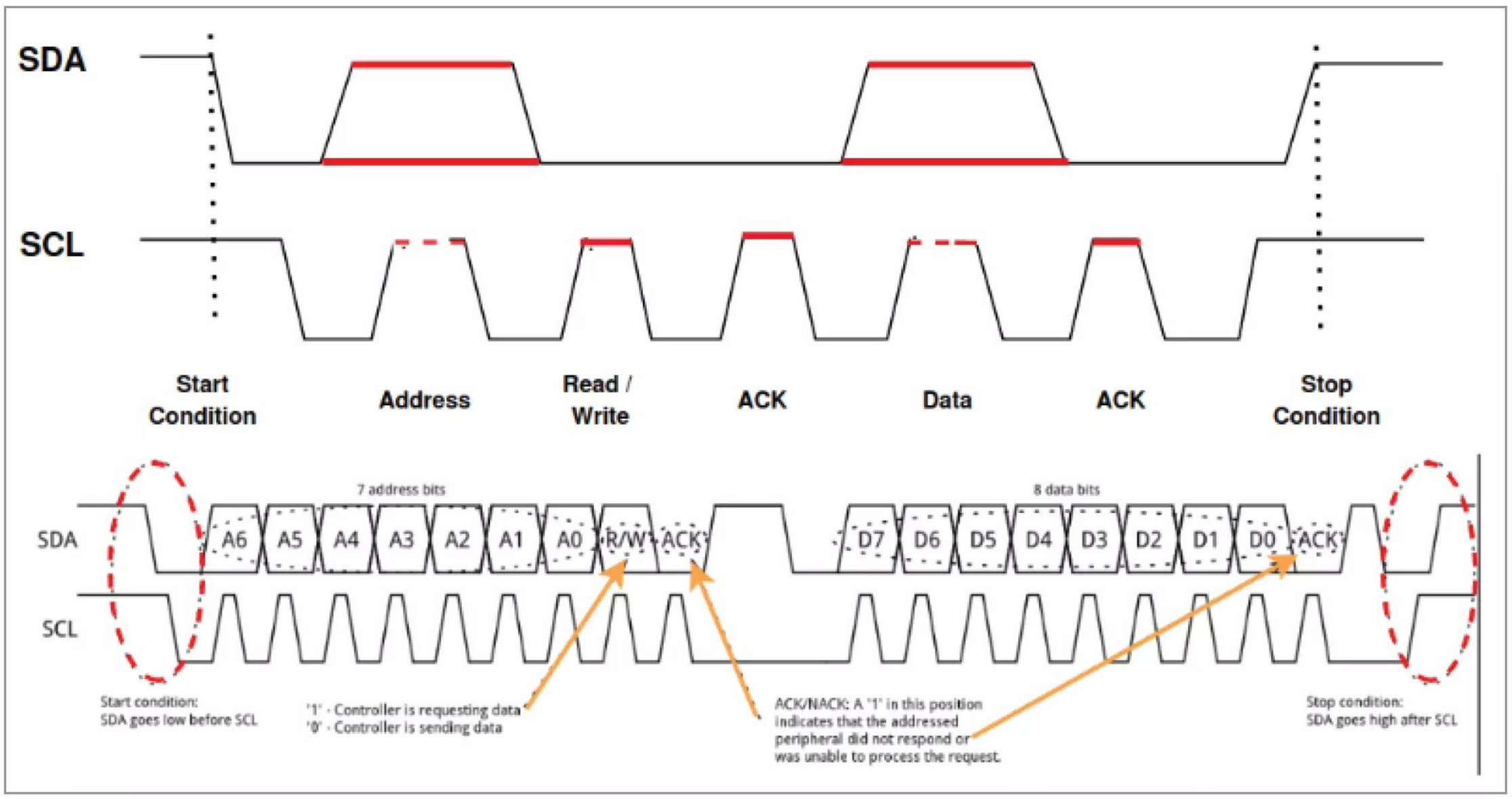


Figure 3: Steps in I2C communication

followed by a read/write bit. The read/write bit indicates whether it wants to read from or write to the slave.

Acknowledge bit (ACK): The addressed slave device responds by pulling the SDA line low during the next clock pulse (SCL). This confirms that the slave is ready to communicate.

Data transmission: The master or slave (depending on the read/write operation) sends data in 8-bit chunks. After each byte, an ACK is sent to confirm that the data has been received successfully.

Stop condition: When the transmission is complete, the master sends a stop condition by releasing the SDA line to HIGH while the SCL line is HIGH. This signals that the communication session has ended.

Repeated Start

Sometimes the master needs to first write something to a device (like a register address), and then immediately read data back from the same device.

If the master sends a STOP and then a new START, another master could take control of the bus in between. That would break the transaction. Instead, the master uses 'Repeated Start'. This is another START condition without sending STOP first. So the bus is not released.

This means that no other master can interrupt and the same device continues the conversation.

Clock stretching: When the slave needs more time

The master controls the clock (SCL) but some devices are slow. Examples are:

- EEPROM writing to flash
- Sensor doing internal conversion
- PMIC updating measurements

If the master keeps sending clocks, the slow device may not be ready.

As discussed earlier, I2C uses open-drain signals so anyone can pull the line LOW and nobody forces it HIGH.

During communication the master releases SCL (tries to make it HIGH). If slave is not ready, it pulls SCL LOW. The clock stays LOW and there is a pause in the communication. When slave is ready, it releases SCL. The pull-up resistor makes SCL HIGH, and the clock continues.

So the slave controls time, not the master.

Arbitration: When two masters talk at the same time

I2C supports multi-master systems. This means both the CPU controller and the BMC can be masters. Both are connected to the same bus.

When both devices start at the same time without arbitration, signals collide, data becomes garbage, and the bus crashes. So, I2C uses bit-level arbitration.

A key rule is that LOW always wins.

Because of open-drain:

- Sending '1' → device releases line (goes HIGH by pull-up)
- Sending '0' → device pulls line LOW

If one device sends '1' and another sends '0', the actual bus level equals LOW.

Here's an example of arbitration.

Two masters start at the same time: Master A and Master B both send START (as shown in Figure 5).

Step 1: Both send address bits:

Master A sends bit = 1 (releases SDA)

Master B sends bit = 0 (pulls SDA LOW)

Step 2: Bus becomes LOW. Both read the bus:

Master B: I sent 0 and I see 0 → OK

Master A: I sent 1 but I see 0 → I lost!

Step 3: Master A stops transmitting. It switches to listening mode.

Step 4: Master B continues alone. No data corruption, no reset and no crash.

This happens bit by bit, not byte by byte. So arbitration is fast and safe.

Limitations of I2C

While I2C is simple, modern systems like smartphones and servers are more demanding. Some of its limitations include:

Limited speed: Even Fast-mode Plus (1MHz) isn't enough for high-throughput sensors or large server systems that need real-time readings.

No in-band interrupts: Devices can't signal the master about urgent events without a separate GPIO interrupt pin.

Fixed addresses: Static addressing can cause conflicts when multiple identical sensors are used — common in large boards like server motherboards.

Higher power use: Pull-up resistors constantly draw current when lines are low, which can add up in energy-sensitive systems.

As embedded system complexity grew, particularly in modern servers with hundreds of sensors and monitoring ICs, these weaknesses drove the development of a more advanced replacement: I3C.

Introducing I3C: Smarter, faster, and more efficient

I3C stands for improved inter-integrated circuit. It keeps the same

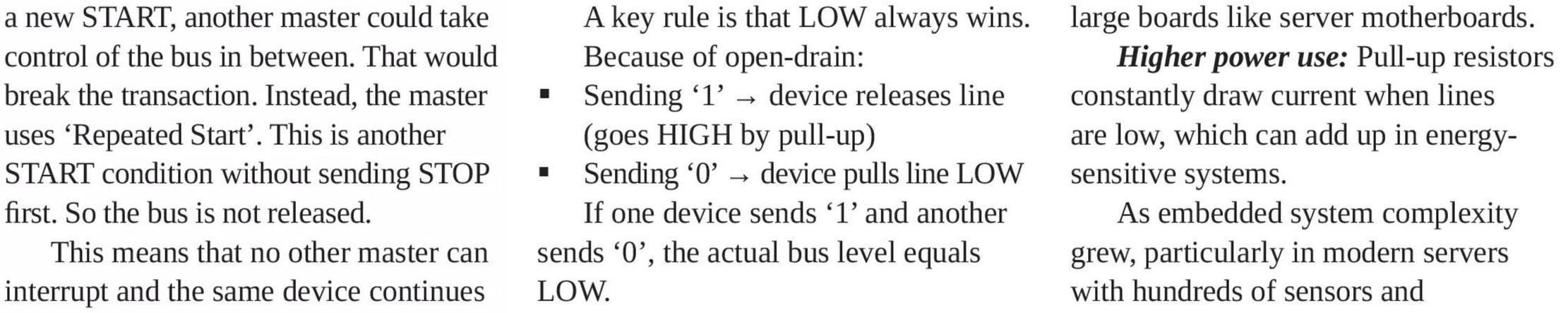


Figure 4: I2C Repeated Start

two-wire design but upgrades it for today's needs. It's backward compatible with I2C, which means older devices can still connect, but it offers major enhancements like faster communication speeds, Dynamic (runtime) Address Assignment, In-Band Interrupts, and lower power operation.

For server platforms, this means better performance for the BMC and faster sensor polling — all with fewer wires and less power draw. I3C is becoming an integral part of next-generation server management and data centre monitoring infrastructures.

I3C bus uses both open-drain and push-pull. Open-drain is used for initialisation and arbitration, while data transfer is used for push-pull.

In push-pull a device can actively drive the line both LOW and HIGH. It connects the line to the ground to send a logic 0 and connects it to the supply voltage to send a logic 1.

Push-pull outputs do not require pull-up resistors and are typically used in point-to-point or single-driver communication interfaces.

Open-drain to push-pull switching in I3C is as follows:

- The I3C bus always starts in open-drain mode to maintain backward compatibility with legacy I2C devices and to support arbitration during bus initialisation.
- Once dynamic addresses are assigned and only one target is selected for a private transfer, arbitration is no longer required.
- In this phase, the master and the selected I3C target may use push-pull signalling on the SDA line to achieve higher data rates and faster signal transitions.
- If legacy I2C devices are present on the bus, they do not respond to CCC (common command codes) such as ENTDAA. The master must continue to use open-drain signalling for bus management and any I2C-compatible transactions.

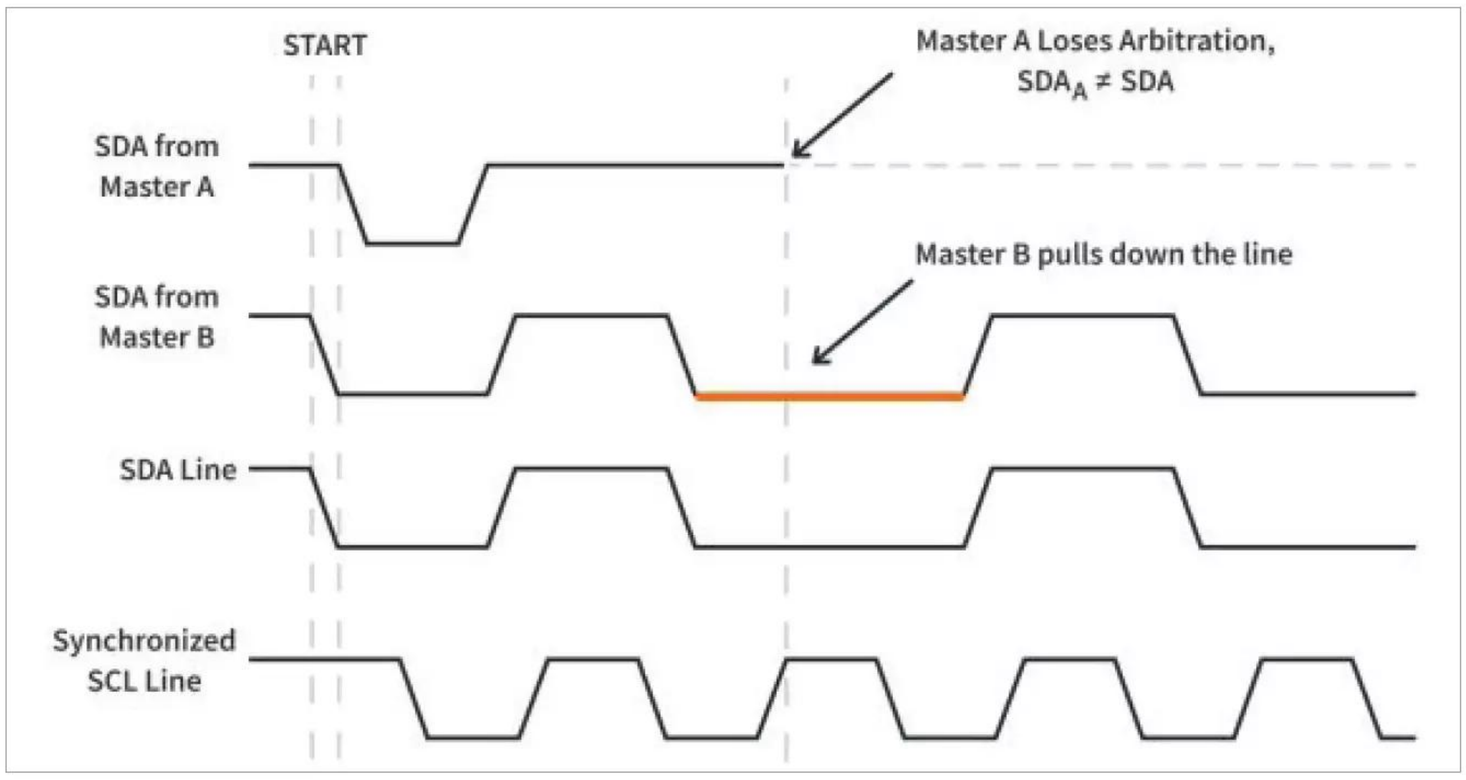


Figure 5: I2C arbitration

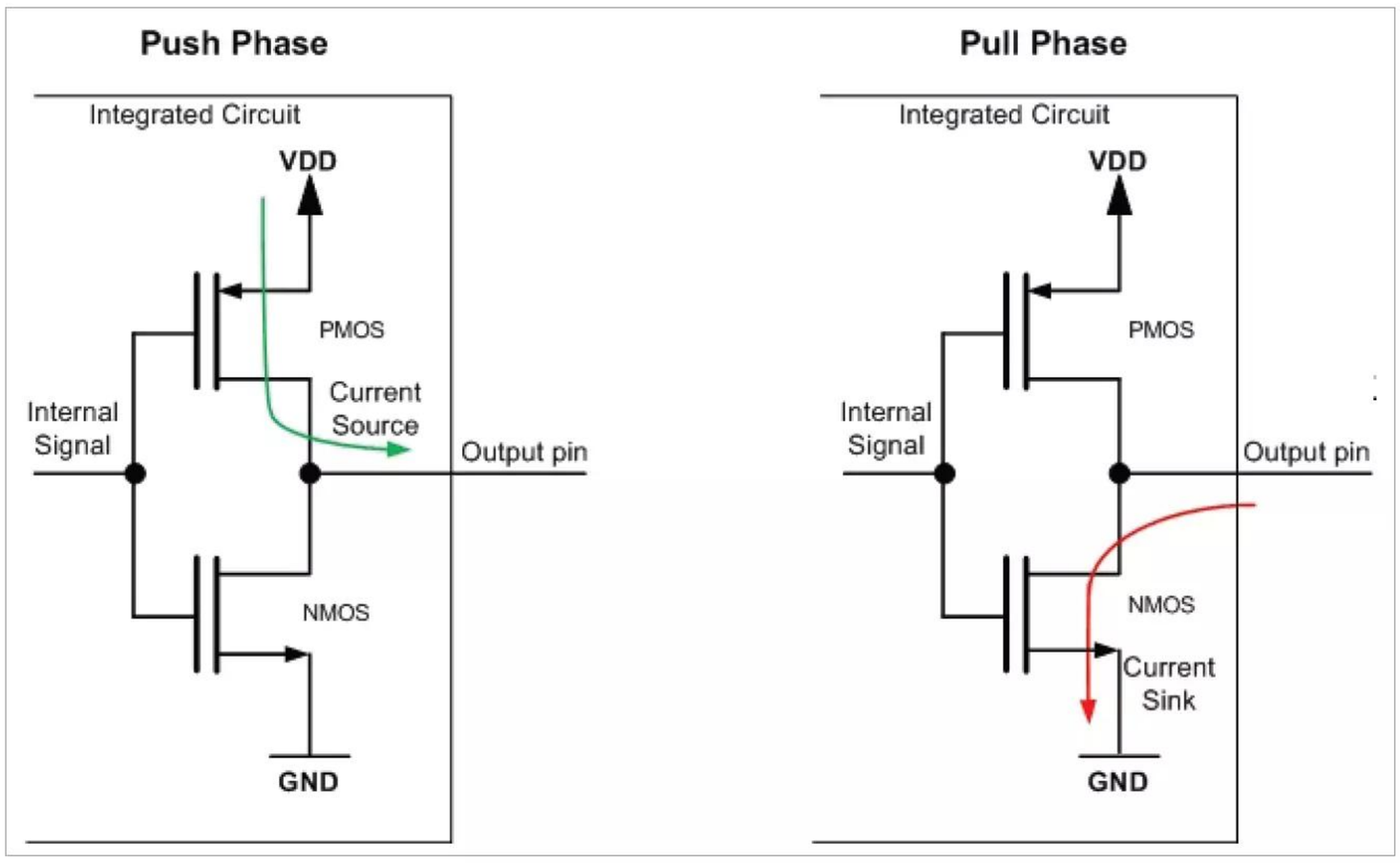


Figure 6: Push-pull

- However, push-pull signalling can still be used for private I3C transfers to I3C targets, while ensuring that electrical compatibility with I2C devices is maintained.

During ENTDAA (Enter Dynamic Address Assignment), devices do not yet have addresses. So for identifying the devices, they use PID (48 bits). PID contains:

- Vendor ID (who made the chip)
- Part ID (model of the chip)
- Instance ID (which copy of the chip it is)

All chips have a globally unique identity. This is how arbitration works safely.

Dynamic Address Assignment flow

- ***Master broadcasts ENTDAA:*** All devices without a dynamic address prepare to identify themselves.
- ***All unaddressed targets prepare to respond:*** Each target loads its 48-bit PID into shift registers. The device copies its internal 48-bit ID into hardware that can send one bit per clock on the SDA line and monitor the bus for arbitration, which is required for DAA in I3C.
- ***PID arbitration:*** The device with the lowest PID value wins. The master receives the full PID of the winning device.

- ***Master assigns dynamic address:*** 7-bit address + parity bit (to detect errors; parity = 1 if the number of 1's is even, and parity = 0 if the number of 1's is odd). The device accepts the address and stores it in the hardware register inside it (DADDR).
- ***Device ACKs and exits ENTDAA:*** The device does not participate again in DAA and uses the dynamic address from now on.
- ***ENTDAA continues for remaining devices:*** The same process continues until no devices respond anymore.
- ***ENTDAA ends:*** When the master sees no response during the ENTDAA phase, it knows that all devices have been assigned addresses.

In-Band Interrupts — Devices can call for help

In I2C, if a sensor has a problem, it cannot easily alert the master. It needs extra wires. In I3C, devices can send alerts directly on the same bus using In-Band Interrupts (IBI).

For example, if temperature suddenly rises in a server, the sensor sends IBI to BMC, which reacts immediately (increase fan, shutdown, alert admin). This is done without using extra wires, which is very useful in dense server boards.

Where I2C and I3C are being used today

Bus type	Common applications
I2C	EEPROM storage, temperature sensors, RTCs, fan controllers, basic power monitors, legacy server BMC communication
I3C	Sensor hubs in mobile devices, camera modules, SoC integration, server BMC-sensor communication (next-gen IPMI replacements), high-speed telemetry systems

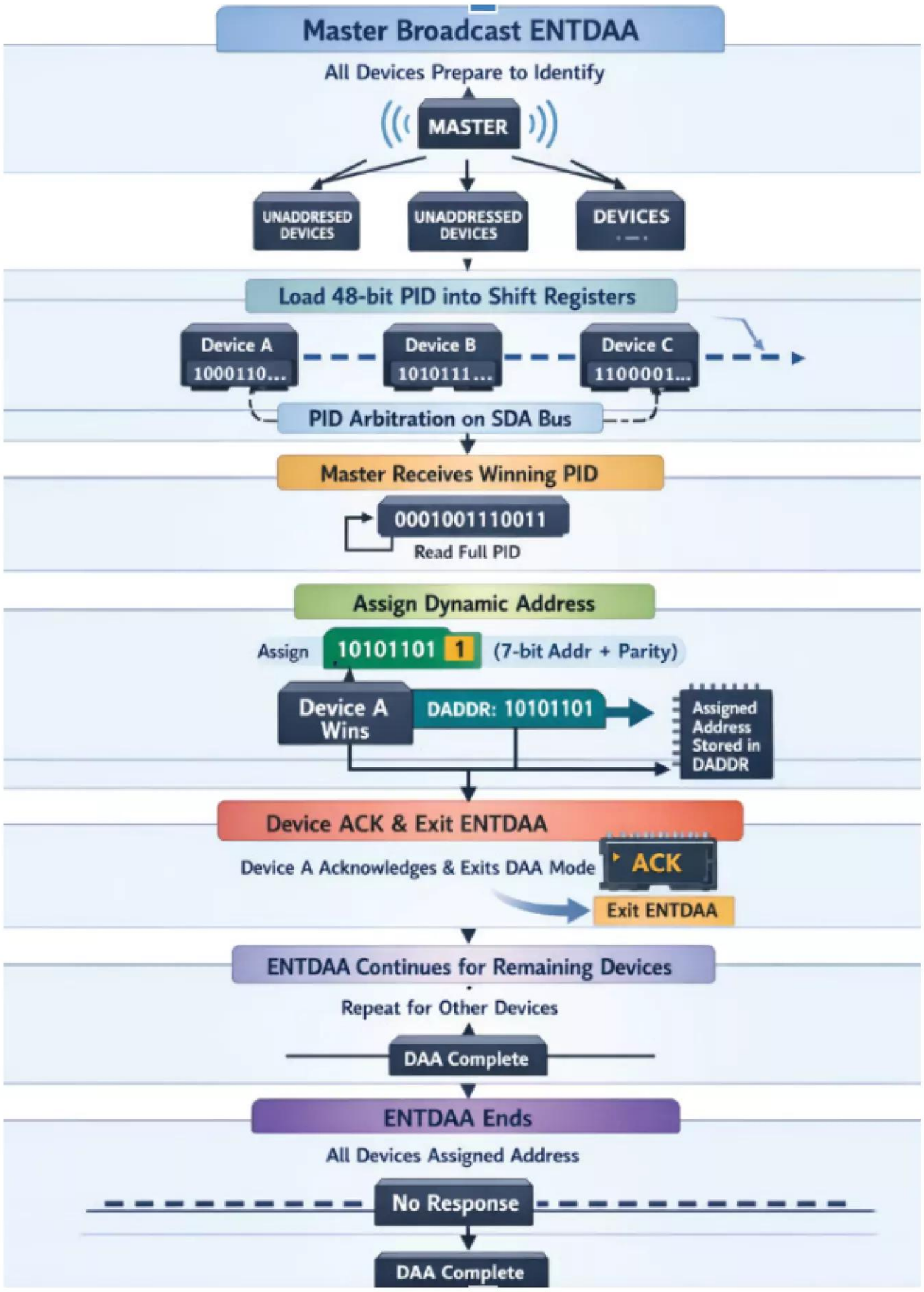


Figure 7: DAA flow in I3C

I2C became widely adopted because of its simplicity, low hardware cost, and reliability. It remains suitable for low-speed and low-complexity peripherals. However, modern embedded systems demand faster communication, lower power consumption, and better scalability.

I3C preserves the simplicity of two-wire communication while introducing features such as dynamic address assignment, in-band interrupts, and push-pull signalling, making it suitable for today's high-performance sensor-rich systems.

Seen together, I2C and I3C show how good standards grow over time. We don't throw away what already works; we build on it and extend it so it can handle the demands of newer, more advanced hardware. END

By: Priyadharshini S.

The author is an open source enthusiast working in an MNC.

INDIA'S AI CONFERENCE BUILT FOR DEVELOPERS

Learn how AI is actually built, shipped, and scaled:
from developers who do it every day.







2ND EDITION

aiDevCon INDIA

12-13 March 2026
Bengaluru

THEMES

- Agentic AI • Coding with AI
- DevOps & DevSec with AI • Data Management
- Open Source AI • Succeeding with AI



**Looking to hire, sell to,
or influence AI developers?**
Call +91-9811155335 or growmybiz@efy.in
to sponsor or exhibit



Statistics of AI Devcon 2025: **2,273** Attendees | **48** Speakers | **4** Workshops

R N I No. DELENG/2012/49440, Mailed on 27/28th of Advance month
Published on 27th February 2026

Posted at Magazine Post RMS Delhi 110006

Co-located with
INDIA ELECTRONICS WEEK

India's Only Conference For IoT Developers & Creators

IOTSHOW.IN
PROFIT FROM IOT

13-15 MAY 2026 • KTPO • BENGALURU

For more information on sponsoring, exhibiting or attending please call +91-9811155335 or growmybiz@efy.in

PROFIT FROM

