



中国科学院高能物理研究所
Institute of High Energy Physics, Chinese Academy of Sciences

17th Workshop of the FCPPN/L

France–China Particle Physics Network and Laboratory

📍 Lyon, France | 📅 29 June – 3 July 2026

AutoDeploy: HEP ML Deployment on Heterogeneous Hardware



Mustofa A. Ahmed

Institute of High Energy Physics, Chinese Academy of Sciences

On behalf of the IHEP Computing Center team



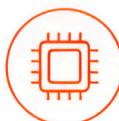
Model
Checkpoint



AutoDeploy



User
Objectives



Target
Hardware



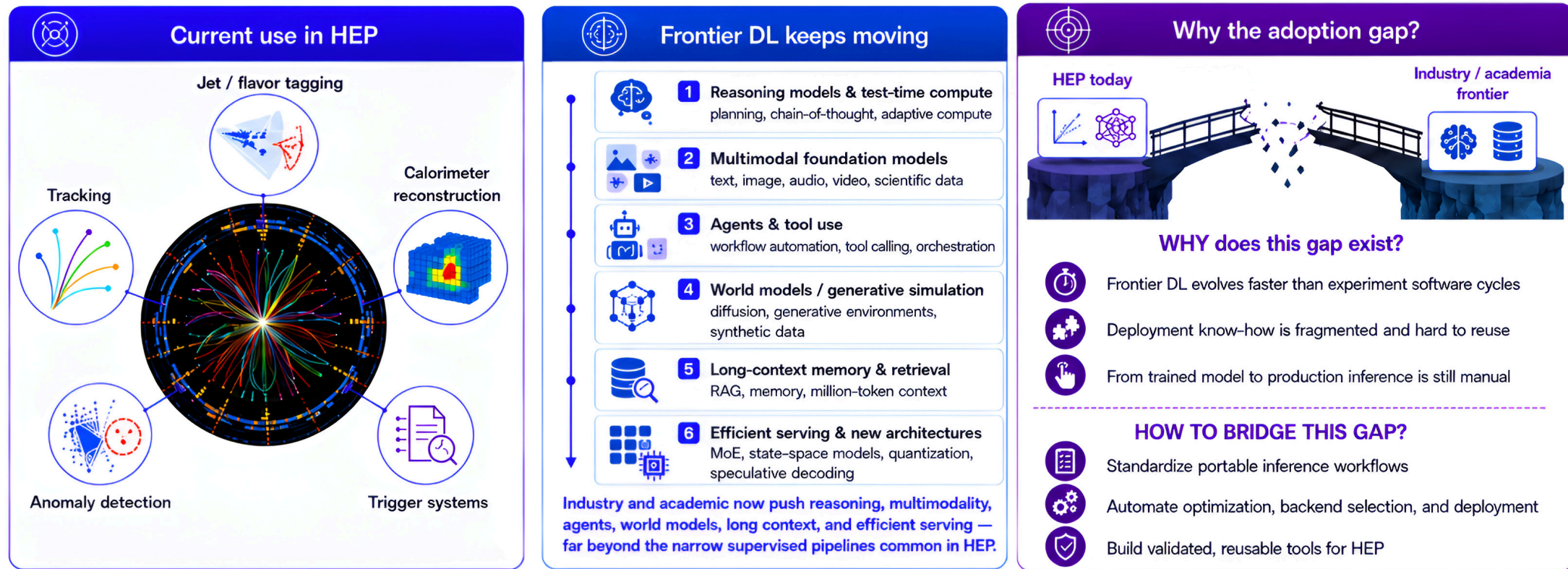
Best
Pipeline





HEP Uses ML Today — But Frontier Adoption Still Lags

ML is already valuable in high-energy physics, but turning frontier models into reliable inference systems remains slow.



HEP does not lack ML models; it lacks a scalable path from frontier models to validated deployment.

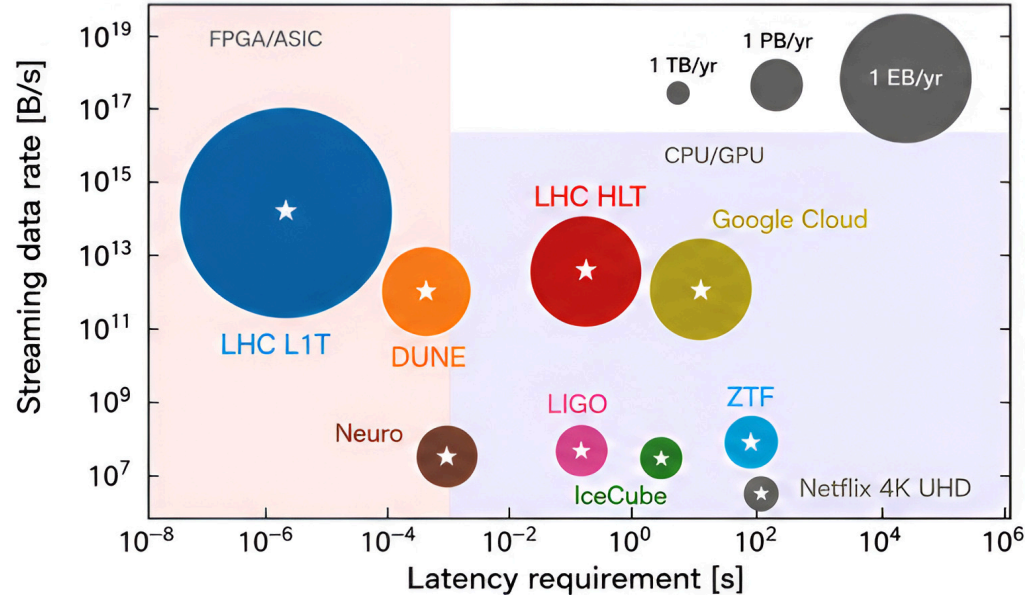


Why Deployment Becomes the Bottleneck in HEP

Unlike ordinary ML serving, HEP inference must satisfy extreme timing, data-rate, hardware, and physics-validation constraints.

Representative operating regimes

A3D3 Institute



HEP trigger and online systems operate in unusually demanding latency and streaming-rate regimes.

Source: A3D3 Institute (2022), <https://a3d3.ai/>; also shown in Indico: <https://indico.global/event/12246/contributions/109491/>



Microsecond latency

- Trigger paths can require μs -scale decisions.
- Slow inference may be unusable, not merely inconvenient.



Huge data rates

- Online systems process continuous detector streams.
- High throughput is mandatory; buffering is limited.



Fixed hardware budgets

- CPU / GPU / FPGA targets impose resource, memory, and power limits.
- Operator support and implementation details depend on the backend.



Physics fidelity & validation

- Accuracy loss can become physics inefficiency or systematic bias.
- Results must be reproducible, auditable, and stable over long experiment lifetimes.



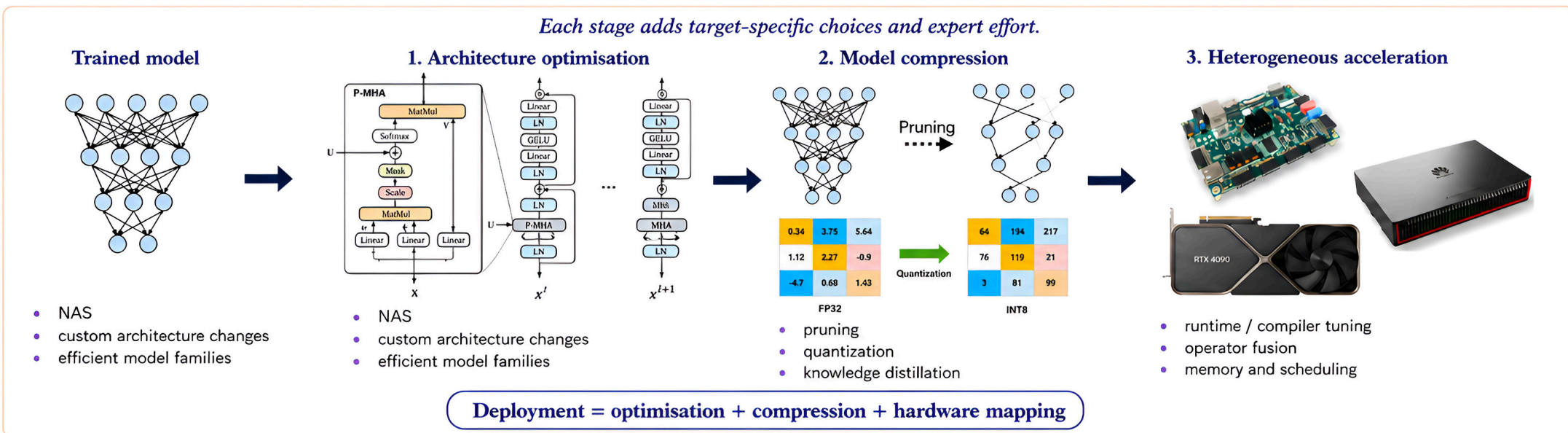
A trained model is not yet a deployable inference system.



In HEP, deployment candidates must pass timing, resource, physics-fidelity, and reproducibility gates.

Deployment is a workflow, not one step

In practice, deployment spans architecture choices, compression, and hardware-specific tuning — often repeated manual decisions rather than a single conversion step.



Not just format conversion

Real deployment typically involves redesign, approximation, and system-level tuning.



Manual and iterative

Different models and targets often require repeated optimisation loops and expert intervention.



System constraints dominate

The final solution must satisfy latency, memory, power, throughput, and hardware compatibility.

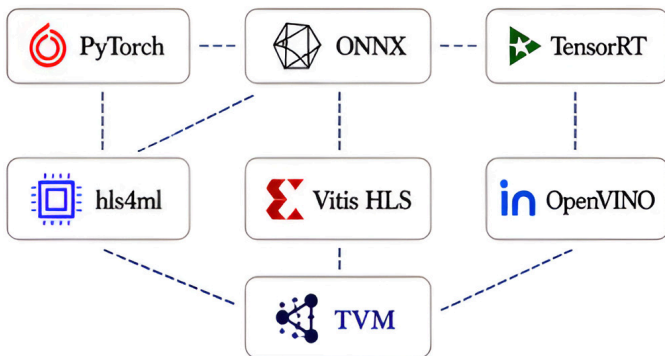


The bottleneck is not exporting a checkpoint — it is navigating the full workflow from model design to hardware-ready inference.

Deployment Is Fragmented Across Tools, Knowledge, and Process

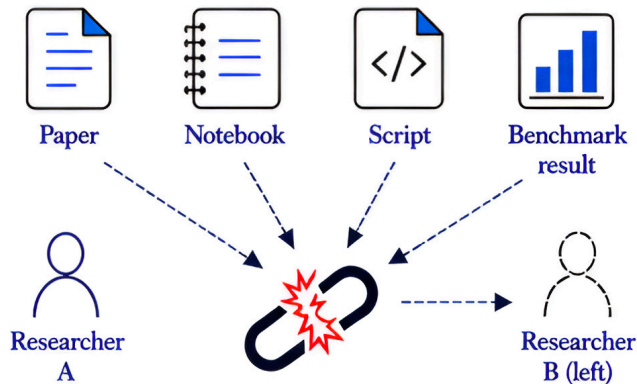
The bottleneck is not one missing tool; it is a disconnected deployment ecosystem.

1. Tool fragmentation



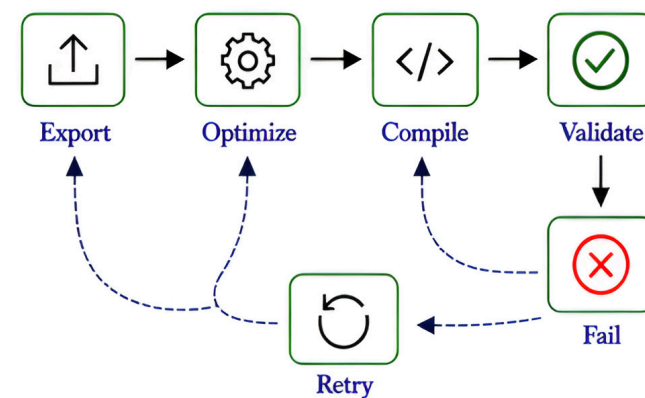
- Different hardware families require different toolchains
- Format conversion and operator support are inconsistent
- Optimizations that work on one stack may fail on another

2. Knowledge fragmentation



- Results are stored in papers, scripts, and private notes
- Success and failure cases are rarely structured for reuse
- When people leave, key deployment know-how often leaves too

3. Process fragmentation



- Teams stitch together local scripts and one-off fixes
- Reproducibility and comparison across runs are difficult
- Trial-and-error dominates instead of systematic search

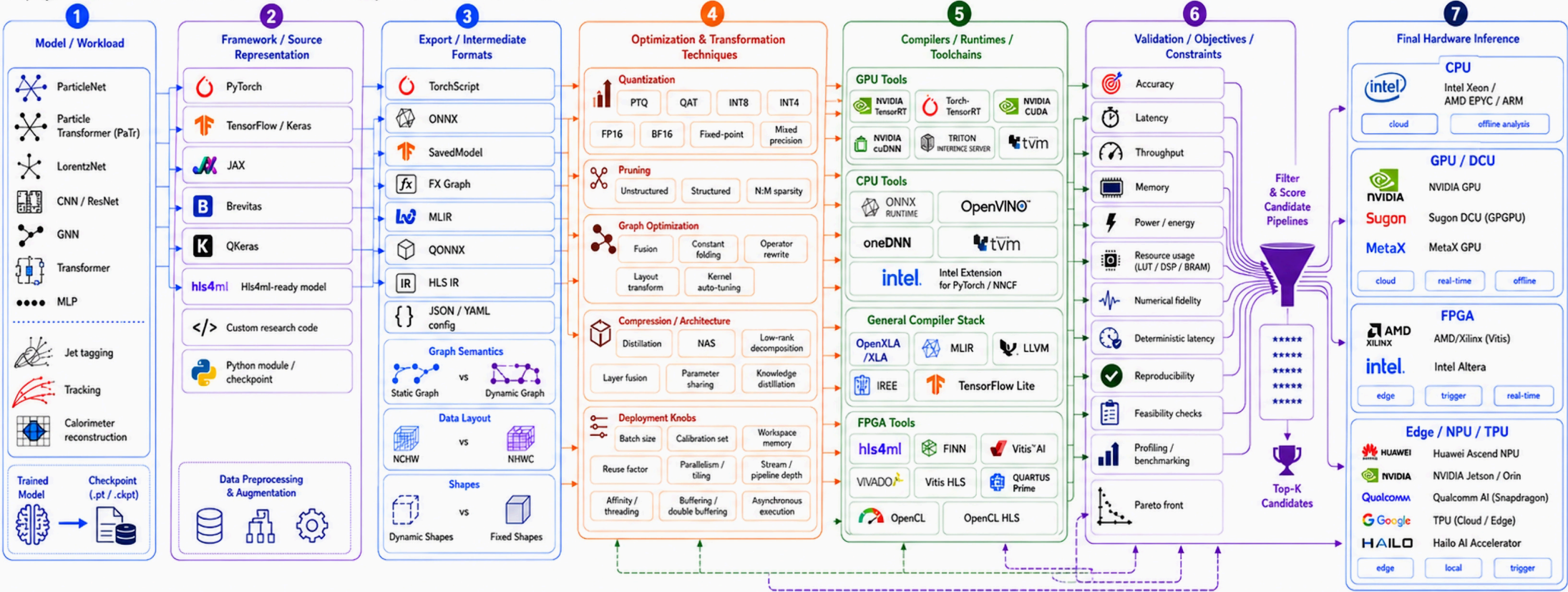
Fragmentation turns deployment into **repeated local effort** instead of reusable institutional knowledge.



The core problem is systemic: the ecosystem is fragmented across tools, knowledge, and process.

Huge Deployment Search Space

Deployment is a constrained search over models, formats, optimizations, toolchains, and hardware.

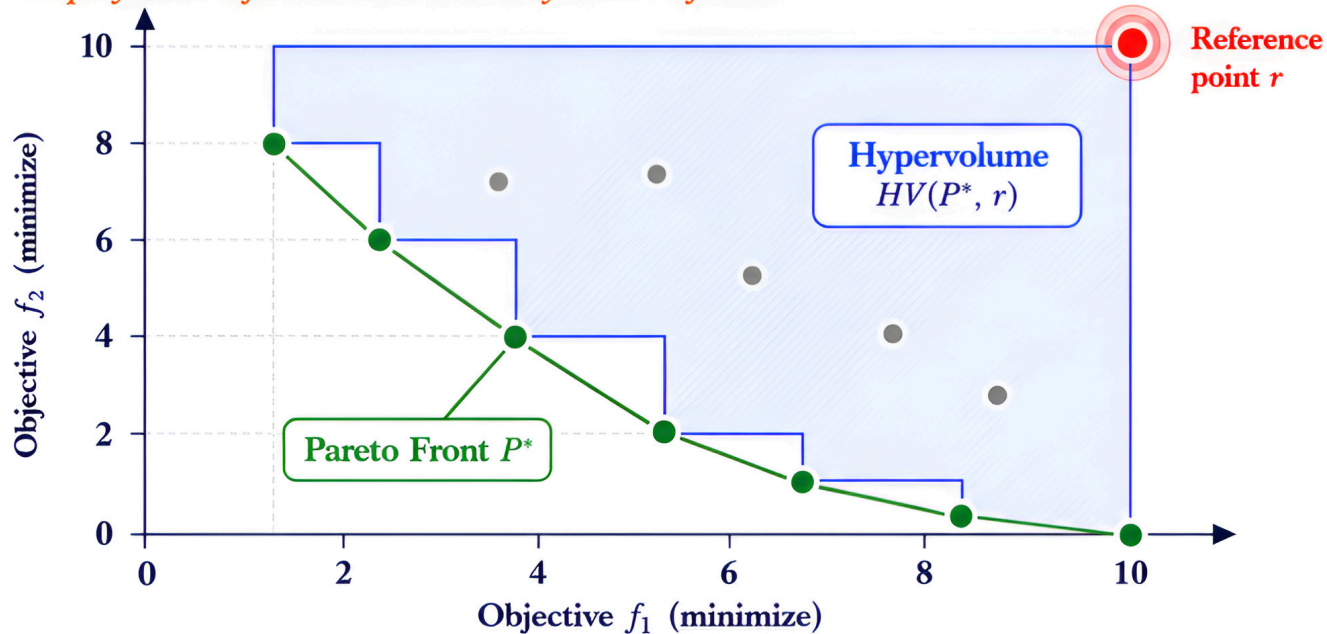


One model can map to many deployment pipelines — manual trial-and-error does not scale.

No Single Best Deployment: Pareto Trade-offs

Deployment is multi-objective: latency, throughput, resource usage, power, and accuracy trade off under target-specific constraints.

Deployment objectives are inherently multi-objective



Multiple objectives

Lower is better on all objectives, but gains in one often come at the expense of another.



Pareto optimal candidates

A candidate is useful when no other option is better in all objectives at once.



Choice depends on context

Trigger, offline analysis, FPGA, GPU, and cloud deployments may prefer different trade-offs.



Search, then select

Return a frontier of strong candidates and choose according to experiment priorities.

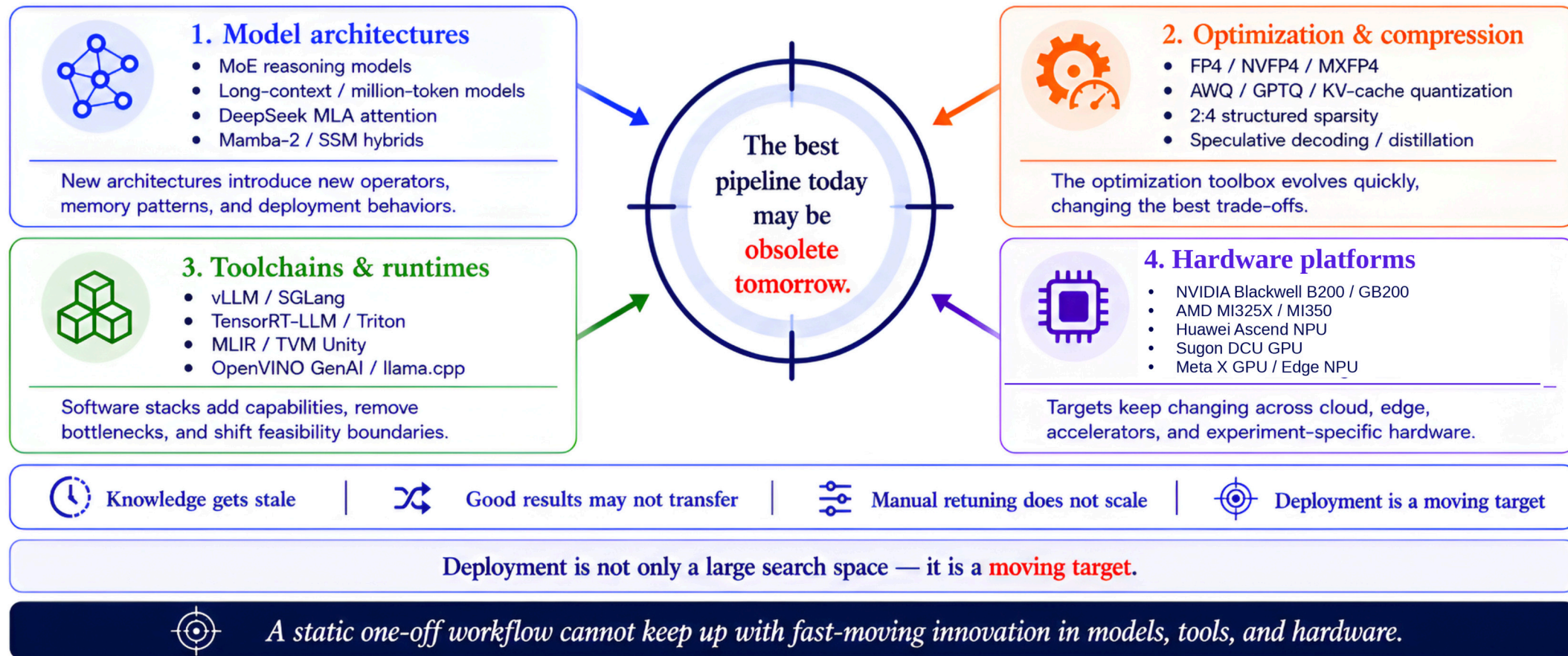
best deployment $\neq$ single winner

best deployment = context-dependent choice on a Pareto frontier

Optimization is not about one universal optimum — it is about navigating trade-offs under constraints.

Moving target problem: models, tools, hardware keep changing

Fast innovation continuously changes what is deployable, what is optimal, and what knowledge remains useful.





IHEP Computing Center Mission & AutoDeploy Scope

We develop practical algorithms and tools for HEP researchers across experiments and computing environments.



IHEP Computing Center Mission

Develop new algorithms and tools that advance HEP physics research across many experiments.



AutoDeploy Goals



Build on what exists

Reuse existing compilers, runtimes, and vendor toolchains.



Take advantage of HEP collaboration

Leverage shared pipelines, failures, and experience into reusable knowledge.



Bridge the gap

Connect ML in HEP with ML practice in industry and academia.



AutoDeploy Scope



Easy to use

Accessible even to researchers with limited deployment or ML background.



From individuals to collaborations

Support personal experiments, small labs, and large collaborations alike.



Across deployment regimes

Useful for offline analysis, edge deployment, online systems, and real-time trigger pipelines.



AutoDeploy Non-goals



Not a replacement ecosystem

AutoDeploy does not replace TensorRT, TVM, hls4ml, Vitis, ONNX Runtime, or experiment frameworks.



System role

It selects, composes, executes, validates, and remembers pipelines built from existing tools.



Focus boundary

Its focus is selection, composition, validation, and evidence reuse — not becoming another monolithic compiler or runtime.



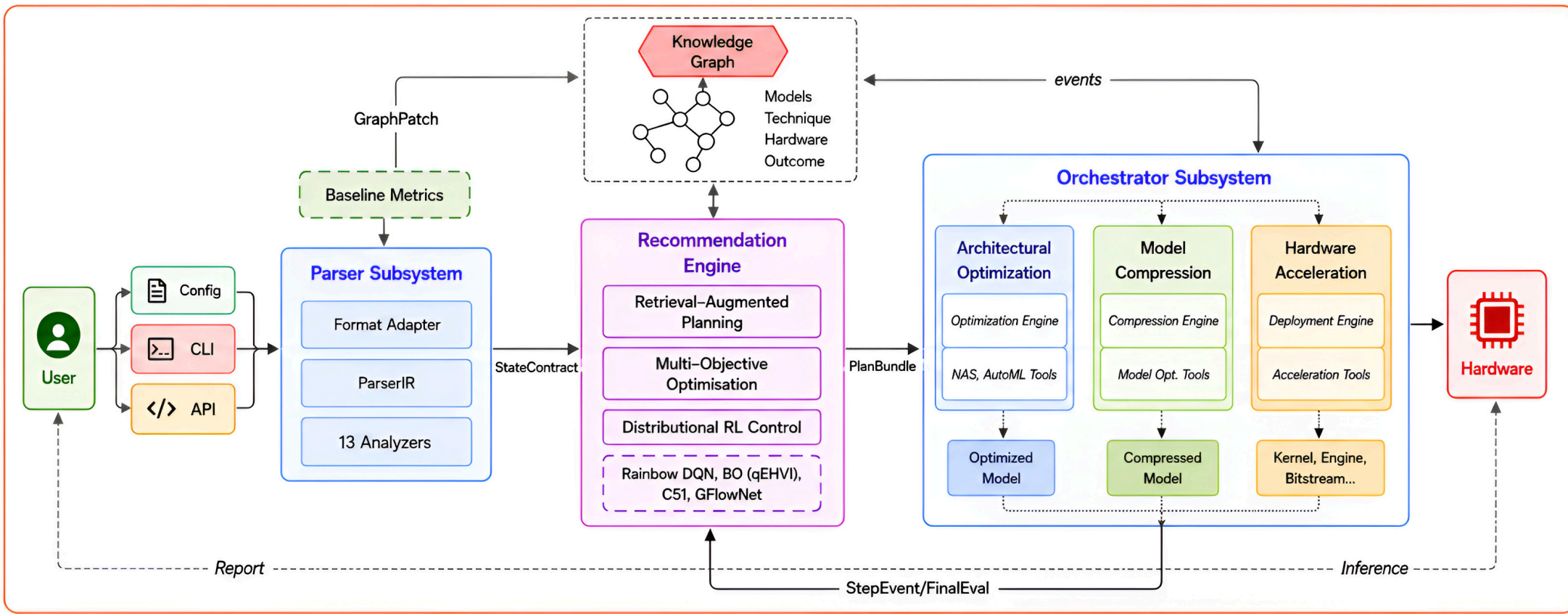
AutoDeploy is an intelligence layer over existing toolchains — enabling reusable, collaboration-friendly deployment workflows for HEP.





AutoDeploy System Architecture

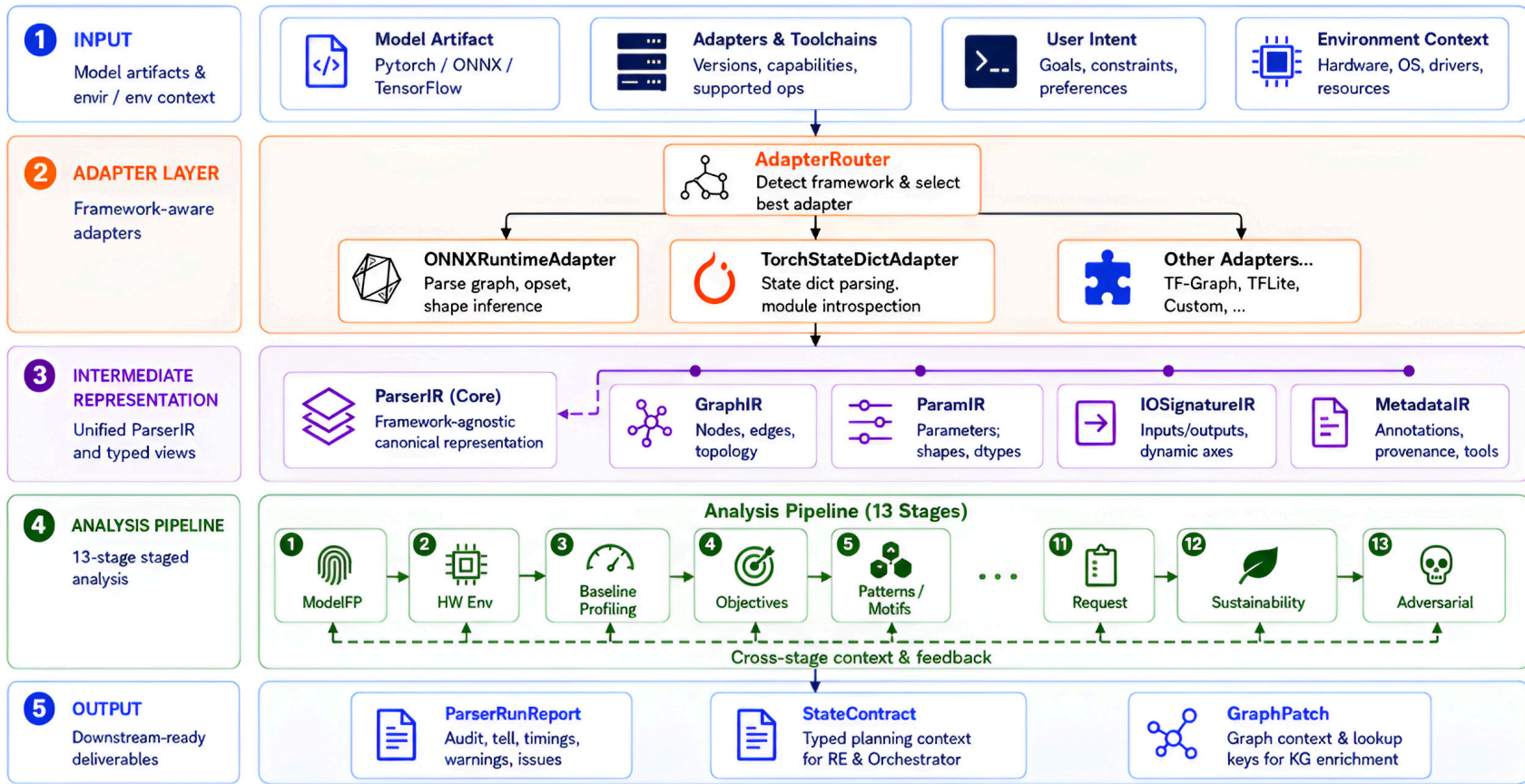
A closed-loop workflow connecting parsing, recommendation, execution, and knowledge accumulation.





Parser Subsystem Architecture

Transforms model artifacts and environment context into rich, analysis-ready representations.



Parser Subsystem at a Glance

The Parser is the first subsystem in AutoDeploy. It parses model artifacts and environment context, performs a 13-stage analysis, and produces structured outputs that drive planning, execution, and learning.

Key Design Principles

- ✓ **Framework-Agnostic**
Unified ParserIR decouples analysis from any specific framework.
- ✓ **Typed & Schema-Validated**
All interfaces are versioned and JSON-serialisable.
- ✓ **Deterministic & Idempotent**
Same inputs → same outputs; safe to re-run and cache.
- ✓ **Separation of Concerns**
Adapters, IR, analysis, and outputs are cleanly separated.
- ✓ **Extensible & Evolvable**
Add adapters, stages, or views without breaking contracts.

Legend (flows)

- Data / context flow
- Adapter selection flow
- IR derivation flow
- Analysis feedback loop
- Output delivery



The parser converts raw model artifacts and deployment context into structured, typed outputs that enable reliable planning, execution, and continuous learning.





StateContract & GraphPatch: Parser Outputs

Typed planning context and knowledge-graph updates produced by the Parser Subsystem.

StateContract (example)

v1.0.0

```
{
  "_type": "StateContract",
  "_version": "1.0.0",
  "schema_version": "1.0.0",
  ▶ "model": { 17 keys },
  ▶ "hardware": { 37 keys },
  ▶ "objectives": { 14 keys },
  ▶ "baseline": { 34 keys },
  ▶ "patterns": { 31 keys },
  ▶ "compatibility": { 17 keys },
  ▶ "workload": { 17 keys },
  ▶ "security": { 10 keys },
  ▶ "kg_signals": { 3 keys },
  ▶ "rl_features": { 10 keys },
  ▶ "request": { 13 keys },
  ▶ "preferences": { 14 keys },
  ▶ "sustainability": { 9 keys },
  ▶ "adversarial": { 13 keys },
  "parser_version": "1.0.0",
  "parser_timestamp": "2026-01-31T14:32:22.020397+00:00",
  "parser_duration_seconds": 10.087217092514038,
  "warnings": [ 1 items ],
  "errors": [],
  "metadata": { 16 keys },
  "target_hardware_urn": "hw:gpu:nvidia:geforce:3060:6:pci:v1"
}
```

GraphPatch (example)

v1.0.0

```
{
  "_type": "GraphPatch",
  "_version": "1.0.0",
  "schema_version": "1.0.0",
  "patch_id": "patch:run:pipeline:20260131_143213_592:56e9:v1:aa91e7d57955",
  "patch_kind": "initial_parse",
  "origin_mode": "admin",
  "source_system": "parser",
  "produced_at": "2026-01-31T14:32:22.020397+00:00",
  "parser_version": "1.0.0",
  "model_urn": "model:particle_transformer:hep:jet_tagging:v1",
  "graph_snapshot_hash":
    "1fbc258042a6445d66b30e8e684390c9a7b00198c12b096f9a622a180",
  ▶ "nodes": [ 55 items ],
  ▶ "relationships": [ 38 items ],
  ▶ "deletes": [],
  ▶ "tags": {},
  ▶ "annotations": { 4 keys }
}
```

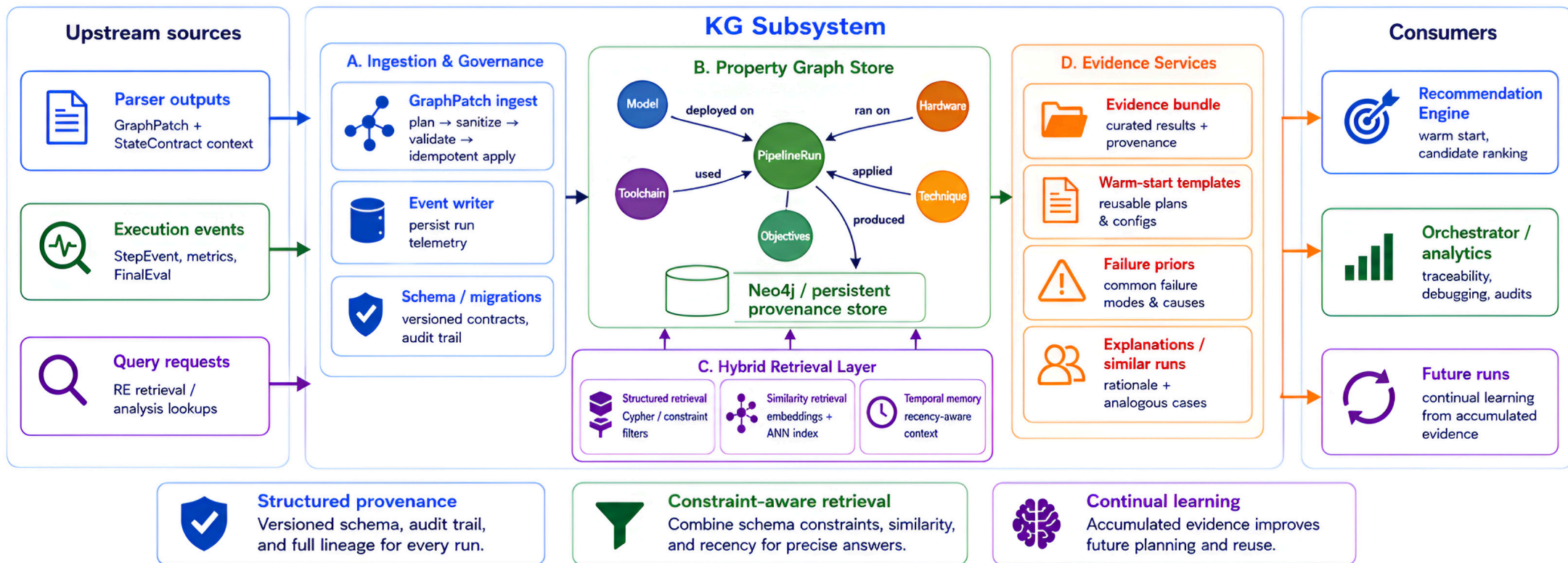


StateContract carries complete, typed planning context. GraphPatch captures minimal, incremental updates to the knowledge graph.



Knowledge Graph as Reusable Deployment Memory

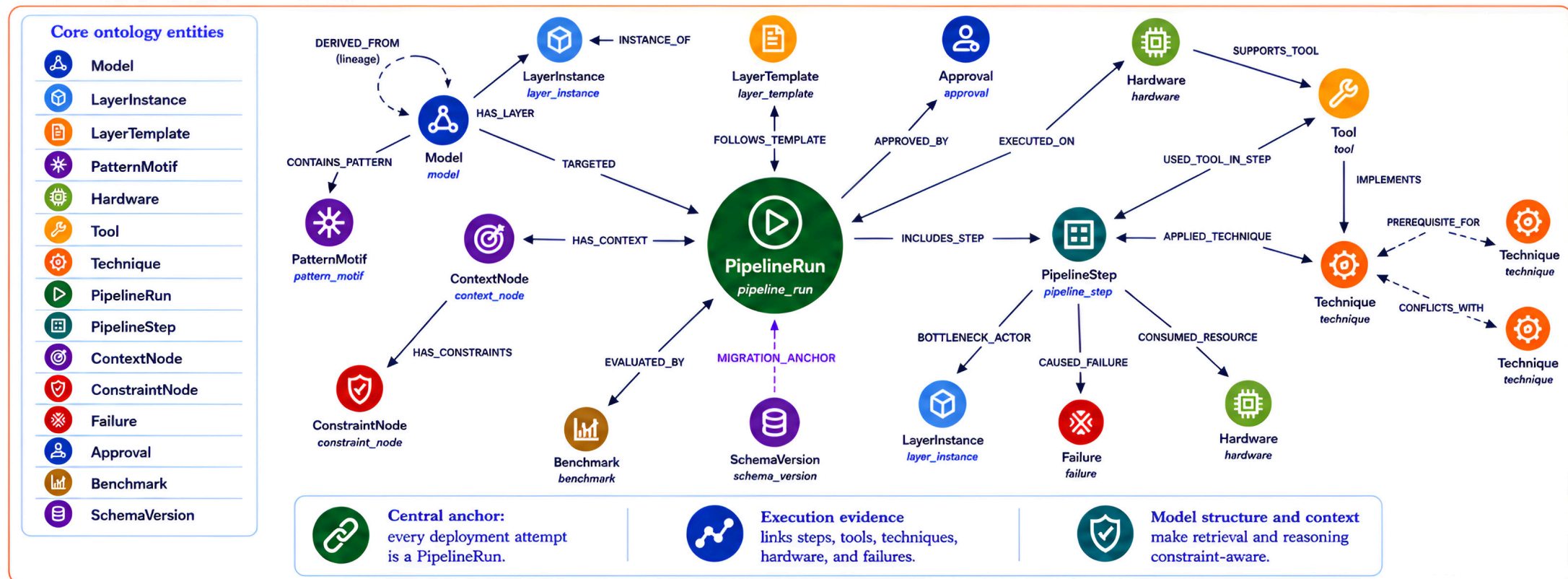
After parsing and execution, the KG stores structured evidence so deployment experience becomes searchable, reusable, and useful future planning.



The KG makes deployment experience reusable: every run becomes structured evidence for planning, filtering, and learning.

Pipeline-Centric KG Ontology

The Knowledge Graph schema is centered on *PipelineRun*, linking model structure, execution context, tools, techniques, and step-level evidence.

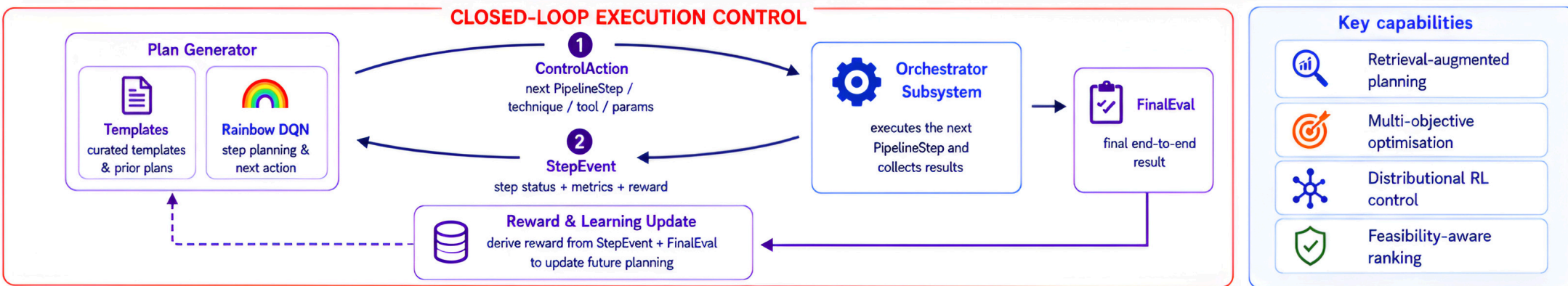
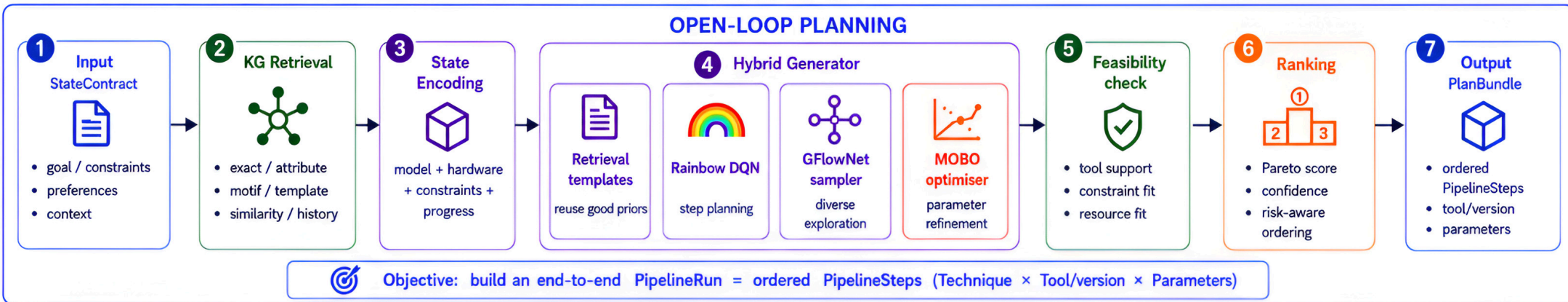


AutoDeploy uses a pipeline-centric ontology so every run, step, constraint, and outcome becomes reusable evidence.



Recommendation Engine Subsystem

Evidence-guided planning and closed-loop control for end-to-end deployment pipelines.



Key capabilities

- Retrieval-augmented planning
- Multi-objective optimisation
- Distributional RL control
- Feasibility-aware ranking



The Recommendation Engine converts structured context and past evidence into ranked, feasible deployment pipelines, then adapts online using execution feedback.





Orchestrator Subsystem

Contract-first execution, deterministic step control, multi-tier isolation, telemetry, and rollback safety.

Isolation Tiers

strongest → weakest



1 Container
OCI/Docker • pinned env



2 Namespace
Linux namespaces • cgroups



3 Bubblewrap
user-space sandbox



4 Virtualenv
Python / tool wrapper



5 Host
direct exec • policy-gated

strongest
available tier
per step

Run PlanBundle

Validate Bundle

Emit RunStart

IDs, budgets, step integrity,
dependency order

for each step s in primary plan



Preflight resolve ToolRunner • verify hardware/driver • check tool availability



Policy Gate [PRE]
untrusted URNs • no-network • host-tier deny



Budget Gate [PRE]
wall-time estimate • budget exhausted



[Container]

[Container]

[Namespace]

[Virtualenv]



Quality Gate [POST]
accuracy • validity • latency • resource flow



Budget Gate [POST]
step timeout • disk cap exceeded

StepEvent

GraphPatch
(idempotent)

FinalEval (optional)

Emit RunEnd + final KG patch

Recommendation Engine (RE)



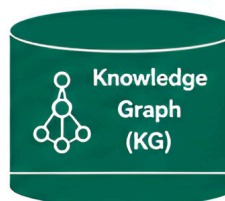
StepEvent
(preflight_fail)

StepEvent
(gate_fail)

ControlAction
continue / retry /
replace / terminate

StepEvent
(post_gate_fail)

StepEvent



Contract-first: validate before execution



Per-step telemetry and KG patches



Checkpointed rollback; no silent repair





AutoDeploy Interfaces

Configuration-first setup, command-line automation, and interactive chat provide three complementary entry points into AutoDeploy.

AUTODEPLOY

Intelligent Model Deployment & Optimization Platform

Version: 1.0.0+beta.20260117.1

Config (YAML)

```
1 schema_version: "1.0.0"
2 model: Part.pt
3 hardware: nvidia_v100
4 objectives:
5   latency: 0.45
6   min_accuracy: 0.93
```

- Declarative run specification
- Reproducible and versioned
- Machine-readable objectives

CLI

```
⚡ AUTODEPLOY deployment • optimization • diagnostics
MODES: run chat validate admin

USAGE  autodeploy [-h] [-V] [-v] [-q] [--skip-validation] [-vv] COMMAND ...
HINT   autodeploy <command> -h for command help
COMMANDS
  run      Run a deployment from a config (user mode)
  chat     Interactive mode (user)
  validate Validate environment and exit
  recommend Suggest proven deployment pipelines
  doctor   Collect diagnostics (prints; optionally writes a file)
  admin    Admin commands (internal use only)
OPTIONS
  -h, --help      show this help message and exit
  -V, --version   print version information and exit
  -v, --verbose   verbose output (includes debug + progress)
  -q, --quiet     errors/warnings only
  --skip-validation skip validation checks (advanced/CI)
  -vv, --dev-verbose developer verbose output (TEMP: enabled for all modes)
```

Automation, diagnostics, and power-user workflows

ChatMode

```
🌈 Welcome to AutoDeploy ChatMode!
1 INFO I'll help you deploy your AI model
  with intelligent optimization.
1 INFO Tell me about your model, target
  hardware, and deployment objectives.
👤 You: Hi!
⚡ AutoDeploy: Hello! 😊 Welcome to our
AutoDeploy conversation. I'm here to help
you deploy your AI models with advanced
optimization techniques. How can I assist
you today?
```

- Natural-language guidance
- Clarifies model, hardware, and objectives
- Guides the next deployment step

Config for reproducibility

CLI for automation

Chat for interaction

AutoDeploy exposes the same deployment system through configuration, command-line workflows, and conversational assistance.



</>

AutoDeploy

Intelligent Model Deployment & Optimization Platform

Version: 1.0.0+beta.20260117.1

Thank You!!

MUSTOFA@IHEP.AC.CN