Naught Fusion: Workload-Level Fusion of Local and Remote Heterogeneous Compute

Benedict Blessing Gbogr
Naught
bbg@junglegrid.dev

Version 0.1 Preprint – September 8, 2026

Abstract

Modern applications can access heterogeneous compute beyond the local machine, including remote CPUs and on-demand GPUs. Yet a faster external accelerator does not necessarily yield a faster workload: resource acquisition, data movement, runtime startup, coordination, and availability can outweigh the accelerator's computational advantage. This paper presents *Naught Fusion Alpha*, an experimental runtime for placing stages of a single workload across local and external heterogeneous resources.

Fusion represents workloads as explicit dependency graphs, filters resources by capability, and uses a deterministic planner based on measured or profiled computation, intermediate-object locality, transfer cost, startup, and coordination. Immutable content-addressed objects preserve same-run state across execution domains, while bulk data and executable payloads are staged independently from compact control messages.

We evaluate Alpha with a three-stage workload whose compute-intensive middle stage can execute locally or externally. A small-kernel microbenchmark (1,024 elements, intensity 4) took 42.0 ms locally but 118.3 s through a managed NVIDIA L4 path. At 250,000 elements and intensity 64, an external L4 kernel-family microbenchmark completed in 134.0 s versus roughly 145–147 s for the local implementation. The latter comparison is a performance-profile proxy rather than a same-input distributed run: the isolated GPU profiler generates its input remotely and therefore excludes Fusion's real input transport.

Alpha also completed a verified same-run local-to-GPU-to-local execution. Stage A produced a 5,032,313-byte immutable object locally; the exact object was staged and verified remotely by run identifier, size, and SHA-256; a provider-routed NVIDIA GeForce RTX 5090 executed CUDA Stage B; the returned artifact was consumed by local Stage C; and final correctness passed. Independently instrumented components of this path total 132.94 s, but the original total timer also included post-run local reference computation and some pre-submission staging was not timed separately. We therefore make no complete end-to-end speedup claim for the first successful run.

The Alpha results establish feasibility and expose the key systems boundary: the value of external acceleration is determined by the complete path required to acquire, reach, and use it, not by accelerator capability alone.

1 Introduction

A computer no longer needs to perform all computation on hardware physically attached to it. Cloud platforms, GPU services, edge machines, clusters, and network-accessible accelerators can expose substantially more capability than a local device. For a heterogeneous workload, this suggests a useful execution model: keep inexpensive or locality-sensitive stages local while recruiting external hardware for stages that benefit from it.

The difficult question is *when* external compute should participate.

A remote GPU may execute a numerical kernel faster than a local CPU, but using that GPU may require capacity discovery, provisioning, data transfer, runtime startup, execution, artifact retrieval, and local continuation. Thus the relevant comparison is not simply

$$T_{\text{GPU compute}} < T_{\text{CPU compute}},$$

but

$$T_{\text{remote path}} < T_{\text{local path}}.$$

When acquisition or transport dominates, a nominally faster accelerator can be a worse placement.

Distributed runtimes such as Ray provide tasks and distributed objects [1]; StarPU schedules tasks across heterogeneous processors using performance models and locality [2]; Legion and Realm expose explicit computation and data placement [3, 4]; rCUDA virtualizes remote GPU access [5]; and split-execution systems partition computation between device and cloud [6, 7]. More recently, Prism performs graph-level CPU/GPU disaggregation over an RDMA-connected datacenter fabric [8], while ORION models variability and cold starts in serverless DAG scheduling [9].

Naught Fusion studies a narrower boundary at the intersection of these mechanisms:

Under what measurable conditions should a workload already executing locally recruit external heterogeneous compute, when acquiring and making that resource ready is itself part of the placement cost?

We built *Naught Fusion Alpha*, a deliberately small explicit-DAG runtime. The evaluation uses three dependent stages,

$$A \to B \to C,$$

where A and C execute locally and B can execute locally or externally.

The experiments establish three main observations. First, external acceleration can be catastrophically unattractive for small work: a 1,024-element, intensity-4 kernel took 42.0 ms locally versus 118.3 s through a managed L4 path. Second, a compute-heavy kernel-family profile can become externally competitive: at 250,000 elements and intensity 64, the managed L4 profiler completed in 134.0 s versus roughly 145–147 s locally. This second measurement is explicitly a profiling proxy, not a same-input Fusion measurement, because the isolated GPU profiler generates its input on the remote side. Third, the complete system successfully executed one same-run workload as local A → external GPU B → local C while preserving exact intermediate-object identity and final numerical correctness.

The first successful full run also exposed an instrumentation error: its recorded 288.3 s total included the local correctness reference executed after Fusion. We therefore exclude that field from performance claims. Known heterogeneous-path components total 132.94 s, but uninstrumented pre-submission staging prevents a valid complete speedup or slowdown conclusion.

This paper makes three contributions:

1. **A workload-level local/external execution model.** Dependent stages execute across local and dynamically acquired external resources while preserving explicit immutable intermediate state.
2. **An explainable placement model.** Capability filtering is separated from optimization, and candidate cost can include computation, data movement, locality, startup, and coordination.
3. **An Alpha feasibility study.** We demonstrate a verified same-run local-to-GPU-to-local execution and document measured resource-readiness, transport, provider-routing, and instrumentation effects that determine the placement boundary.

We do not claim universal speedup, planner optimality, automatic partitioning of arbitrary programs, or production readiness.

2 Motivation and Research Questions

2.1 End-to-end placement cost

For a local stage, a simplified cost is

$$T_{\text{local}} = T_{\text{compute,local}} + T_{\text{local overhead}}.$$

For external execution,

$$\begin{aligned} T_{\text{remote}} = {} & T_{\text{input}} + T_{\text{queue}} + T_{\text{provision}} \\ & + T_{\text{runtime}} + T_{\text{compute}} + T_{\text{output}} \\ & + T_{\text{artifact}} + T_{\text{coordination}}. \end{aligned}$$

Only one term is directly improved by accelerator speed. Remote execution is worthwhile only when the compute advantage exceeds the rest of the path.

2.2 Resource readiness

The Alpha experiments observed completed GPU jobs whose job-creation-to-command-start delay ranged from 84 s to 343 s, plus attempts that never reached useful execution before cancellation. This motivates a resource descriptor closer to

$$R = (\text{capability}, \text{location}, \text{locality}, \text{readiness}, \text{availability})$$

than simply "GPU model."

2.3 Research questions

We evaluate four questions:

- **RQ1:** Can managed external acceleration be much worse than local execution for small work?
- **RQ2:** Does the same kernel family reach a regime where an external GPU provider path becomes competitive with local execution?
- **RQ3:** Can a deterministic planner change its selection from local to external in response to measured profiles rather than hard-coded thresholds?
- **RQ4:** Can one workload execute correctly across local and external heterogeneous resources while preserving the exact same-run intermediate state?

3 System Model

3.1 Workload graph

A workload is a directed acyclic graph

$$W = (S, D),$$

where S is the stage set and $D \subseteq S \times S$ is the dependency relation. Placement is a function

$$\pi : S \to R.$$

For the successful Alpha run,

$$\pi(A) = r_{\text{local}}, \quad \pi(B) = r_{\text{external}}, \quad \pi(C) = r_{\text{local}}.$$

Alpha requires explicit stage boundaries; it does not infer partitions from arbitrary source code.

3.2 Immutable intermediate objects

Stages communicate through content-addressed immutable objects. For object o,

$$id(o) = SHA256(bytes(o)).$$

Metadata can include object size, known locations, producer, and workload run identifier. If a candidate resource already holds an input object, the planner can treat it as a locality hit; otherwise the input contributes transfer bytes.

Same-run identity is stronger than numerical equivalence. The remote Stage-B program verifies the expected run identifier, byte length, and SHA-256 of the exact Stage-A object before CUDA execution.

3.3 Capability and placement

For stage s and candidate r, eligibility is

$$E(s, r) \in \{0, 1\}.$$

Alpha checks requirements such as GPU presence, vendor, memory, and allowed location before performance scoring. The viable set is

$$R_s = \{r \in R \mid E(s, r) = 1\}.$$

For each viable candidate, the planner estimates

$$\hat{T}(s, r) = T_{\text{compute}} + T_{\text{input}} + T_{\text{startup}} + T_{\text{output}} + T_{\text{coordination}},$$

and chooses

$$\pi(s) = \arg\min_{r \in R_s} \hat{T}(s, r).$$

The model is intentionally deterministic and auditable.

3.4 Planned versus actual hardware

Provider routing can substitute physical devices. We therefore distinguish the planner candidate r_{plan} from the actual provisioned hardware r_{actual}. In the first complete successful run, the planner candidate was named `jg-gpu-l4`, but the provider artifact reported an NVIDIA GeForce RTX 5090.

4 Design and Implementation

Alpha is implemented primarily in Node.js. The core includes a workload IR, local execution backend, deterministic planner, content-addressed store, remote Runner protocol, external GPU adapter, correctness checker, profiling utilities, and real/replay/synthetic experiment provenance.

4.1 Benchmark workload

The evaluated workload contains three stages. Stage A generates a centered numerical representation. For deterministic input length N, the local input is generated as

$$v_i = \frac{17i \bmod 101}{100}.$$

Stage A computes the mean μ and serializes centered values

$$x_i = v_i - \mu$$

with explicit schema fields including format version, representation, dtype, count, values, and mean.

Stage B repeatedly applies

$$x \leftarrow \sin(x) + \cos(0.5x)$$

for $250I$ iterations, where I is compute intensity, and emits a compact checksum summary. The local implementation uses JavaScript double precision; the CUDA implementation uses PyTorch float32.

Stage C consumes the returned Stage-B summary and produces the final local result.

4.2 Control, data, and executable transport

The first full-run attempt embedded an approximately 5 MB Stage-A object in the control request and was rejected with HTTP 413. After moving input bytes to provider-native staged storage, another attempt embedded the Python Stage-B program in command arguments and exceeded a 4,096-character command limit.

The successful design separates:

- **control:** run ID, resource requirements, object/script references, short command;
- **data:** Stage-A bytes and returned artifacts;
- **executable:** separately staged Stage-B Python/CUDA program.

4.3 Remote verification and artifact return

The provider-side program checks the exact input hash, byte length, format version, and run identity. It records CUDA availability and physical GPU identity in a JSON artifact. The artifact is retrieved through the provider API, inserted into the local CAS, and consumed by Stage C.

4.4 Generic runtime versus first-GPU harness

The generic DAG engine currently supports local and Runner-based execution. The first managed-GPU full success is orchestrated by a dedicated `run-fusion-first.js` harness that reuses the workload, planner, CAS, GPU

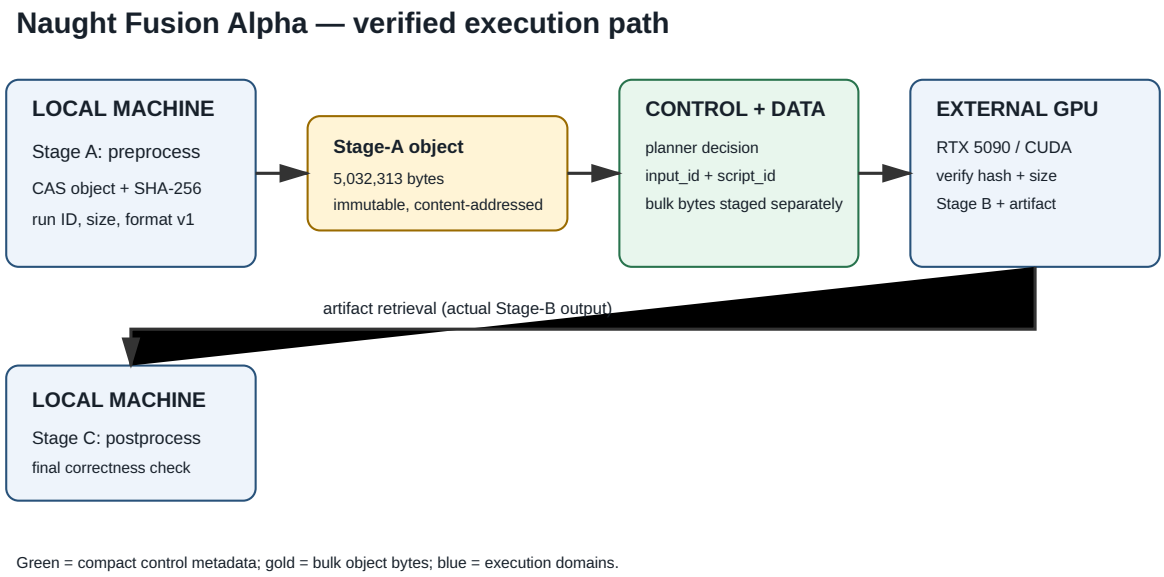

Figure 1: Naught Fusion Alpha architecture and verified execution path. Control metadata is kept compact while bulk intermediate data and executable payloads are staged separately.

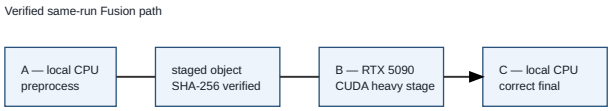

Figure 2: The first verified same-run heterogeneous chain: local Stage A, exact staged object, real GPU Stage B, and local Stage C.

backend, and correctness logic. We therefore claim integrated Alpha semantics, not a production-transparent provider backend.

5 Experimental Methodology

5.1 Environments

The local machine executes the planner, CAS, Stage A, local reference Stage B, Stage C, and experiment harness. A separate CPU Runner provides 6 Intel Haswell vCPUs, approximately 11 GiB RAM, Node.js 22.22.1, and no GPU. Real accelerator experiments use provider-routed NVIDIA GPUs in the `eu-central` region; principal observed devices include NVIDIA L4 and RTX 5090.

The evidence corpus contains 58 raw records, including CPU controls, real GPU profiling jobs, provider failures, partial integration attempts, and the first complete heterogeneous success. Real, replayed, and synthetic records are labeled separately; only real records support empirical claims.

5.2 Kernel-family profile anchors

The isolated GPU profiler is useful but must be interpreted carefully. It constructs an input tensor remotely using `torch.arange(size)/100` rather than consuming the Stage-A object. Thus it measures the same kernel family, element count, and iteration count, but it excludes Fusion's actual input transfer and does not execute on identical starting values.

We therefore use the 1,024/4 and 250,000/64 GPU records as *performance-profile anchors*, not as full same-input distributed Stage-B experiments.

5.3 Correctness

Transport integrity is exact: hashes and byte lengths must match. Floating-point semantic correctness uses

$$|a - e| \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max(|a|, |e|).$$

The full-success harness uses absolute tolerance 10^{-2} and relative tolerance 10^{-6}.

5.4 Retry and failure policy

Paid provider experiments use zero automatic retries. Startup and full-execution deadlines trigger cancellation. Failed records remain in the evidence corpus rather than being silently removed.

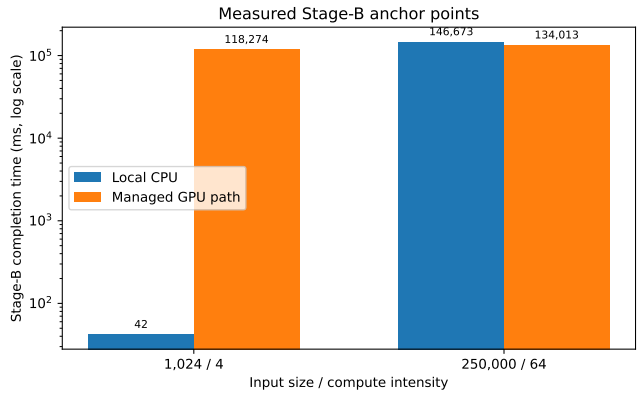

Figure 3: Measured local and managed-GPU kernel-family profile anchors. The remote profiler uses the same element count and iteration count but generates input remotely; these are performance-profile measurements, not same-input Fusion runs.

6 Evaluation

6.1 Remote CPU control

The real-VPS CPU control corpus shows that external execution is not automatically useful. Median latencies over the archived control set were approximately 49.1 ms for local-only, 3,249.2 ms for remote-only, 1,130.0 ms for naive heavy offload, 56.4 ms for the compute-centric baseline, and 50.7 ms for Fusion. Fusion selected zero remote heavy stages in this environment.

6.2 GPU performance-profile anchors

Figure 3 shows the two isolated kernel-family anchors on a logarithmic scale.

At 1,024 elements and intensity 4, five local runs produced a median of 42.0089 ms. The managed L4 profiler required 118,274 ms, including approximately 92 s between job creation and command start. The external path was roughly 2,815× slower.

At 250,000 elements and intensity 64, one local profile recorded 146,673.3 ms (the separately recorded full local-only workload later measured Stage B at 145,079.7 ms). The managed L4 profiler completed in 134,013 ms, including approximately 109 s between job creation and command start. This indicates that the same compute shape can reach a regime where the managed GPU provider path is competitive with the local implementation, despite a large cold-start penalty.

Because the profiler generates remote input rather than consuming the exact Stage-A object, we do *not* interpret the 12.66 s difference as an end-to-end Fusion speedup.

6.3 Planner behavior

The planner was fed measured profiles for the heavy configuration and selected the external candidate. In the first complete successful run, its explanation predicted 146,673.3 ms locally and 134,018.0 ms externally. However, the predicted transfer time for the approximately 5 MB intermediate was only about 5 ms, whereas the observed local-to-Runner copy took 17.25 s. The first Alpha planner therefore underestimated part of the real transport path substantially.

This result supports a narrower claim than planner optimality: *the planner changes placement in response to measured profiles and explicit locality/transfer terms, but its current topology model is not yet calibrated well enough to establish optimal placement on the complete external path.*

6.4 First verified same-run heterogeneous execution

The successful run identifier was

`fusion-first-1788793479345-8e48cb83`.

Stage A produced an immutable 5,032,313-byte object with SHA-256

`a885f5b097bda283412a5e87497830e3612549062`
`a0cf3db80a78acb741d146c`

The exact object was copied to the Runner, staged with the provider, mounted, and verified remotely. The provider assigned an NVIDIA GeForce RTX 5090. CUDA Stage B completed and returned a 429-byte artifact. Stage C consumed that artifact locally.

The GPU checksum was 416,863.375 and the local reference checksum was 416,863.34522097587, an absolute difference of 0.029779 and relative difference of approximately 7.14×10^{-8}. Final correctness passed.

This establishes the core feasibility result: one explicit workload crossed local and real external GPU execution domains and returned to local execution while preserving same-run object identity.

6.5 Timing audit of the full success

The original raw record reports `total_ms`=288,321.3 ms. Source inspection shows that the timer started before Stage A but stopped only after the harness executed an additional local Stage-B and Stage-C reference for correctness. Thus

$$T_{\text{recorded}} = T_{\text{Fusion}} + T_{\text{reference}} + T_{\text{validation}},$$

so 288.3 s is not Fusion latency and is excluded from performance claims.

The independently timed components of the heterogeneous path are shown in Table 1.

The known subtotal is about 12.24 s below the separate local-only workload. However, provider input staging, script staging, estimate handling, and some pre-submission coordination were not timed independently. The frozen Alpha evidence therefore cannot classify the

Table 1: Independently instrumented components of the first complete successful run.

Component	Time
Local Stage A	113.6 ms
Local → Runner object copy	17,249.5 ms
Provider submission → artifact	115,571.0 ms
Local Stage C	5.1 ms
Known subtotal	132,939.2 ms
Separate local-only workload	145,179.1 ms

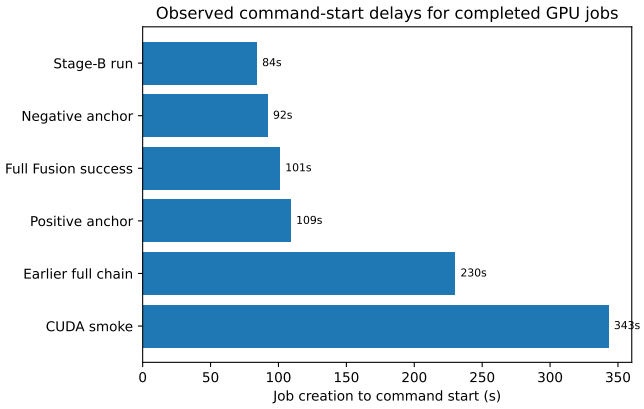

Figure 4: Job creation to command start for completed real GPU jobs with usable timestamps. These are heterogeneous experiments, not repeated samples of one identical workload; the figure demonstrates readiness variation rather than a calibrated distribution.

complete heterogeneous run as either a speedup or a slowdown.

6.6 Resource readiness

Figure 4 summarizes command-start delays for completed GPU jobs with usable timing evidence. The observed range is 84–343 s. Additional attempts remained in a starting state until cancellation, including records with 289 and 412 billable seconds without an artifact.

The central implication is that resource readiness is not background noise: it can be minutes long and therefore directly changes placement economics.

7 Failure Analysis and Lessons

Table 2 summarizes the principal failures that changed the Alpha design.

The semantic mismatch is especially important. Exact byte hashes prove that both sides possess the same object, but not that both runtimes assign those bytes the same meaning. Alpha therefore added explicit fields such as format version, representation, dtype, mean, and count.

The timing-scope error similarly shows that a correct distributed execution can still produce an invalid scientific conclusion if metric boundaries are wrong. Future experiments should derive aggregate timing from structured event timestamps rather than one broad timer.

8 Discussion

8.1 Workload-level fusion versus device virtualization

Remote-GPU virtualization preserves a device-like abstraction. Fusion instead places explicit workload stages. The coarser abstraction sacrifices transparency but exposes information that becomes essential when resource acquisition and state movement are large relative to compute.

8.2 Cold and warm resources are different regimes

For a cold resource,

$$T_{\text{remote,cold}} = T_{\text{acquire}} + T_{\text{stage}} + T_{\text{compute}} + T_{\text{return}},$$

whereas an already-ready resource may approach

$$T_{\text{remote,warm}} = T_{\text{stage}} + T_{\text{compute}} + T_{\text{return}}.$$

Given the observed minute-scale startup delays, a controlled warm-versus-cold experiment is the highest-value next measurement.

8.3 From stage placement to subgraph placement

Alpha has only one placement-flexible stage. Larger DAGs require joint placement because greedy stage-level choices can introduce unnecessary object movement. A future objective is

$$\Pi^* = \arg\min_{\Pi} T(W, \Pi),$$

where Π assigns resources to the entire graph or subgraphs.

8.4 Placement under uncertainty

The current planner uses point estimates, but provider readiness is variable. A future model can treat completion as a random variable $T_r \sim D_r$ and optimize expected latency, tail latency, deadline probability, or a multi-objective function combining latency, monetary cost, and failure risk.

8.5 What Alpha establishes

Alpha establishes: (1) one verified same-run local-to-GPU-to-local execution; (2) a real small-workload regime in which external execution is overwhelmingly unattractive; (3) a compute-heavy kernel-family profile in which a managed GPU path becomes competitive with local execution;

Table 2: Principal Alpha failure classes and resulting systems lessons.

Failure	Immediate cause	Lesson
HTTP 413	Multi-megabyte object embedded in job request	Bulk state must be referenced/staged independently from compact control metadata.
Command-size rejection	Python executable embedded in command arguments	Substantial executable payloads need their own transport path.
Semantic mismatch	JavaScript/Python disagreed on whether Stage-A values were raw or centered	Content identity must be paired with an explicit semantic schema.
Host-side logging failure	Local code crashed after a valid GPU artifact returned	Remote-stage success is not workload success; downstream continuation matters.
Startup timeouts	Provider accepted jobs that did not become usable before deadlines	Readiness and failure probability belong in the resource model.
Hardware substitution	L4-derived candidate yielded RTX 5090	Planned resource class and actual physical hardware must be recorded separately.
Timing-scope error	Local correctness reference ran inside broad total timer	Experimental instrumentation is part of the system's correctness boundary.

and (4) direct evidence that readiness, transport, provider routing, and measurement boundaries affect placement.

Alpha does not establish full end-to-end speedup, general planner optimality, a measured decision surface, or production reliability.

9 Related Work

Distributed and heterogeneous runtimes. Ray, StarPU, Legion, and Realm establish distributed task, locality, and heterogeneous scheduling mechanisms [1, 2, 3, 4]. Fusion does not claim these mechanisms as new. Its Alpha setting focuses on a local workload considering a resource that may not yet be part of a ready execution fabric.

Remote accelerators and split execution. rCUDA virtualizes remote CUDA access [5]; Neurosurgeon and related edge/cloud systems model computation/communication tradeoffs in split inference [6, 7]. Fusion uses explicit workload stages rather than accelerator calls or DNN layers and includes resource acquisition/readiness in the placement path.

GPU disaggregation. Prism is particularly close conceptually: it automatically partitions recommendation models into CPU- and GPU-intensive subgraphs and schedules them across RDMA-connected CPU and heterogeneous GPU pools [8]. The distinction is not graph-level CPU/GPU partitioning. Prism operates over a provisioned datacenter fabric; Fusion Alpha studies whether a workload already executing locally should enter an external execution path whose acquisition and readiness are themselves part of the cost.

Cold-start-aware DAG scheduling. ORION explicitly models runtime variability, dependencies, cold starts, and prewarming in serverless DAG scheduling [9]. Fusion asks a complementary local-versus-external question: how should such readiness costs affect whether a stage recruits the external environment at all?

The novelty boundary is therefore narrow: Naught Fusion combines established task, object, remote-execution, and startup-aware ideas at the boundary where *acquiring the external heterogeneous resource is itself part of the placement action.*

10 Limitations

The strongest limitations are the small number of real GPU measurements, one verified complete success, an incorrectly scoped full-run timer, a remote profiling anchor that generates input remotely rather than consuming the real Stage-A object, an optimistic transfer prediction in Planner v0, cold-start-dominated provider behavior, hardware substitution, one principal provider route, one synthetic workload family, and explicit rather than automatically discovered stage boundaries.

The full successful run therefore supports feasibility, not a speedup claim. The 134.0 s heavy L4 profiler supports a performance-profile observation, not a same-input end-to-end remote Stage-B result. Likewise, the plan-

ner's external selection demonstrates responsiveness to measured profiles, not established optimality.

The highest-priority next experiments are: (1) correctly instrumented repeated full Fusion runs; (2) repeated local baselines; (3) a same-input provider-backed Stage-B experiment that stages the actual Stage-A object; (4) warm-versus-cold execution; (5) targeted near-break-even sampling; and (6) at least one additional workload or external backend.

11 Conclusion

Naught Fusion Alpha investigates when a workload already executing locally should recruit external heterogeneous compute. The first prototype demonstrates a complete same-run local-to-GPU-to-local workload with exact intermediate-object verification and correct downstream continuation. It also shows why accelerator speed alone is an insufficient placement criterion: provider startup can take minutes, real transport can differ dramatically from optimistic models, provider hardware can be substituted, and data/executable staging is a first-class part of execution.

The strongest current result is therefore architectural and empirical rather than a speedup claim:

The value of external acceleration is determined by the complete path required to use it.

Naught Fusion Alpha establishes a minimal working system for studying that boundary. The next phase is to measure the boundary rigorously with correctly timed same-input repetitions, warm and cold resources, and richer workload families.

References

[1] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, 2018, pp. 561–577. URL: https://www.usenix.org/conference/osdi18/presentation/moritz.

[2] Cédric Augonnet et al. "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures". In: *Concurrency and Computation: Practice and Experience* 23.2 (2011), pp. 187–198. DOI: 10.1002/cpe.1631.

[3] Michael Bauer et al. "Legion: Expressing Locality and Independence with Logical Regions". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. 2012.

[4] Sean Treichler, Michael Bauer, and Alex Aiken. "Realm: An Event-Based Low-Level Runtime for Distributed Memory Architectures". In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 2014. DOI: 10.1145/2628071.2628084.

[5] Carlos Reaño et al. "CU2rCU: Towards the Complete rCUDA Remote GPU Virtualization and Sharing Solution". In: *2012 19th International Conference on High Performance Computing (HiPC)*. 2012, pp. 1–10. DOI: 10.1109/HiPC.2012.6507485.

[6] Yiping Kang et al. "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge". In: *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2017. DOI: 10.1145/3037697.3037744.

[7] En Li et al. "Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing". In: *arXiv preprint arXiv:1910.05316* (2019). URL: https://arxiv.org/abs/1910.05316.

[8] Lingyun Yang et al. "GPU-Disaggregated Serving for Deep Learning Recommendation Models at Scale". In: *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, 2025, pp. 847–863. URL: https://www.usenix.org/conference/nsdi25/presentation/yang.

[9] Ashraf Mahgoub et al. "ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, 2022, pp. 303–320. URL: https://www.usenix.org/conference/osdi22/presentation/mahgoub.