GRAFT: gRPC-Routed Agent Framework for Tasking in Edge and Personal Devices

Chinmay Shringi
New York University
New York, NY, USA
cs7810@nyu.edu

Alon Hillel-Tuch
New York University
New York, NY, USA
ah5647@nyu.edu

Sariya Rizwan
Pace University
New York, NY, USA
sl00399n@pace.edu

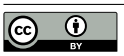
Abstract

GRAFT is a distributed edge orchestration system that routes structured tasks across heterogeneous personal devices, enabling off-grid workload completion without cloud services while keeping each worker within an explicit input budget. The system provides a sessioned gRPC control plane with device registration, capability-aware policy routing, compute-target inference dispatch (CPU, GPU, NPU), safety-bounded remote execution, and per-request telemetry. We demonstrate GRAFT through a multi-device PDF summarization workflow in which a coordinator partitions a document across four devices, each running a local SLM, and merges partial results through a staged execution plan with explicit synchronization barriers. Our evaluation on a four-device mesh covering silicon from Apple (laptop), Snapdragon X Elite (laptop), Samsung (mobile), and Arduino Q (embedded) hardware shows that all planned tasks execute on their intended devices, stage ordering is preserved, and end-to-end completion time is governed by the slowest parallel worker rather than total document size. New single-device baselines show that a capable laptop-class model can outperform the mesh on raw wall clock, but only the mesh achieves higher aggregate coverage while keeping every worker inside its native budget; a phone-class 1B model either covers substantially less input or degrades badly when pushed beyond budget. The live demonstration exposes routing decisions, per-task progress, device-level timing, and failure semantics in real time, contributing both a novel systems architecture for multi-device agent orchestration and a compelling interactive experience for conference attendees.

Keywords

Edge computing, distributed inference, device orchestration, heterogeneous computing, small language models, gRPC, agent frameworks, task routing, personal devices

ACM Reference Format:
Chinmay Shringi, Alon Hillel-Tuch, and Sariya Rizwan. 2026. GRAFT: gRPC-Routed Agent Framework for Tasking in Edge and Personal Devices. In *ACM Conference on AI and Agentic Systems (CAIS '26), May 26–29, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3786335.3813216



CAIS '26, San Jose, CA, USA

ACM ISBN 979-8-4007-2415-2/2026/05
https://doi.org/10.1145/3786335.3813216

1 Introduction

Personal devices now host diverse local compute resources, including CPUs, GPUs, NPUs, and on-device model runtimes, yet day-to-day orchestration remains fragmented [1][2]. A single small language model (SLM) running on one device can handle short prompts, but many practical tasks require processing inputs that exceed the context budget or inference capacity of constrained endpoints. Summarizing a 941-page novel such as *The Count of Monte Cristo* [3], for example, is feasible on a laptop only with aggressive truncation, while phone-class SLMs must trade off coverage against degraded over-budget execution. For small personal meshes, the real gap is lightweight local orchestration, not raw compute.

Existing agent frameworks provide strong workflow logic but generally assume a single host or a homogeneous service backend [4][5][6]. The underlying language-model stack is shaped by transformer architectures and in-context few-shot behavior [7][8]. Large-scale schedulers offer robust orchestration mechanisms [9][10][11], but they target data-center assumptions and containerized workloads rather than end-user tool semantics on personal hardware. GRAFT addresses this middle ground: a systems layer for structured task execution across a small heterogeneous mesh of personal devices, enabling budget-respecting off-grid completion of workloads whose desired coverage would otherwise force reduced input or degraded behavior on constrained single-device models.

GRAFT contributes three implementation-backed elements. A sessioned gRPC control plane with a device registry, capability model, and policy-based routing for heterogeneous personal-device meshes. A compute-target inference router that dispatches by CPU, GPU, or NPU target with explicit fallback behavior when an accelerator path is unavailable. Finally, it introduces safety and observability boundaries for remote tool execution through command allowlists, bounded file access, one-time transfer tickets, and per-request telemetry. The novelty is the combined execution contract: a single routing abstraction that spans heterogeneous inference targets and constrained multi-device tool execution with auditable failure semantics, not the use of gRPC transport by itself.

We evaluate GRAFT on a four-device heterogeneous deployment using PDF summarization as a representative workflow. PDF summarization is well-suited to this demonstration because the same aggregate coverage can be straightforward for a capable laptop-class model, yet destabilizing for a smaller phone-class model once its native budget is exceeded. We report timed execution results on 13-page and 1009-page instrumented runs, single-device baselines on the coordinator and on a phone-class 1B model, and additional demo-artifact results on two public-domain book workloads: *The Adventures of Sherlock Holmes* [12] (183 pages) and *The Count of Monte Cristo* [3] (941 pages). We measure target-device completion,

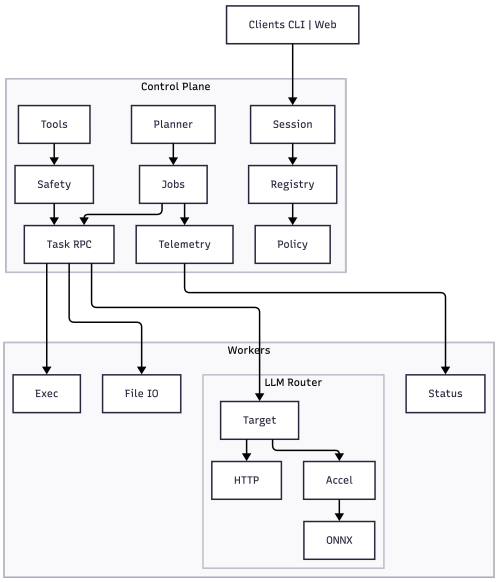


Figure 1: High-level architecture of the implementation artifact: sessioned gRPC control plane, device registry with policy routing, planning pathways, execution backends (remote commands, task execution, local inference, compute-target routing), and telemetry plus auxiliary channels.

per-task latency when available, stage ordering, total job outcome, and budget-respecting coverage. The paper emphasizes measurable system behavior and demo reproducibility rather than model-quality benchmarking. A demo video and interactive artifact are provided for camera-ready review.

2 System Design

2.1 Architecture Overview

GRAFT is organized as a unified server with a sessioned gRPC control plane, an HTTP API and web UI, and a bulk-transfer channel for large artifacts. The control plane maintains device registration, capability metadata, and routing policies. Clients submit routed commands, LLM tasks, or multi-step plans through stable request and response schemas. Devices execute locally or receive forwarded RPC calls, depending on routing policy and capability constraints.

2.2 Routing and Dispatch

Routing in GRAFT is policy-driven. Each task submission includes a routing policy that determines which device should execute the request. `BEST_AVAILABLE` selects the highest-capability device and is the default. PREFER_REMOTE favors a peer over the local coordinator, useful when the coordinator is reserved for merge duties. FORCE_DEVICE_ID pins a task to specific hardware. PREFER_LOCAL_MODEL and REQUIRE_LOCAL_MODEL select devices that already have the requested model loaded; the latter fails immediately rather than falling back. `REQUIRE_NPU` demands accelerator hardware and rejects the request outright if none is enrolled. In every case, policy evaluation returns both the selected device and, when constraints cannot be satisfied, a clear failure condition, making routing outcomes auditable at request granularity.

Execution dispatch is explicit. If the selected device is local, the request executes in-process; otherwise, the coordinator forwards the request over gRPC and returns a unified response envelope with selection metadata and end-to-end timing. The caller always knows which device ran the task, how long it took, and whether any fallback was applied.

2.3 Compute-Target Inference Router

The inference router exposes compute target as a first-class routing dimension: CPU, GPU, or NPU. Requests bind together device selection, model identifier, prompt payload, and compute target. CPU and GPU paths call the local chat or inference runtime over HTTP. NPU paths attempt accelerator-specific execution with an explicit fallback path when accelerator execution is unavailable or fails.

This router is what makes GRAFT an agent framework for local computation rather than a thin RPC wrapper. By treating the compute target as part of the routing contract, upstream planners and agent workflows can express hardware preferences declaratively without managing device-level runtime details. The contribution is not a new model architecture. It is the interface and execution contract that allows heterogeneous personal-device inference backends to be composed through one routing abstraction with explicit failure and fallback semantics, enabling accessible locally coordinated agent workflows that would otherwise require cloud orchestration.

2.4 Job Execution Semantics

A **job** is the durable runtime record created from a single client submission, storing a job identifier, execution plan, aggregate state (`PENDING`, `RUNNING`, `DONE`, or `FAILED`), per-task metadata, and any final reduced output. A **task** is the smallest executable unit: one typed operation with one input payload targeting one device. A **group** is an ordered synchronization barrier collecting tasks into one concurrent stage; the runtime does not advance to the next group until every task in the current group reaches a terminal state and any configured reduce step completes.

The number of parallel tasks in a group scales with the number of participating devices. In the four-device PDF workflow, Group 0 contains four parallel `PDF_SUMMARIZE` tasks (one per device) after capacity-weighted page partitioning. A `CONCAT` reduce step joins worker summaries, and Group 1 runs one `LLM_GENERATE` merge task on the coordinator. The job therefore reports $N + 1$ task records for N devices. Execution is asynchronous: the HTTP handler returns the job identifier immediately while the job manager advances groups in the background.

2.5 Safety Boundaries

Remote command execution is constrained by an allowlist and argument validation. File access paths are bounded to prevent traversal and arbitrary host reads. Bulk artifact download uses one-time tickets with expiration, reducing persistent attack surface. Tool-level file reads are range-bounded to limit accidental over-exposure. These controls focus on practical risk reduction for personal-device orchestration rather than full sandboxing. The implementation also provides a hardened transport profile with TLS, bearer API_KEY enforcement, and pairing-gated discovery; see docs/HARDENED_MODE.md.

2.6 Telemetry

GRAFT emits per-request duration for routed calls and maintains historical device metrics via periodic polling. Telemetry is tied to routing outcomes so that policy compliance, forwarding overhead, and failure cases can be diagnosed with concrete traces. This is necessary for demo credibility because edge deployments exhibit unstable network and device behavior.

3 Evaluation

3.1 Setup

The evaluation focuses on the four-device heterogeneous mesh used for the PDF-summarization demo: one Apple-silicon macOS coordinator, one Windows 11 ARM64 Snapdragon X Elite worker, one Samsung Galaxy S25 Ultra Android 16 worker, and one Arduino UNO Q / QRB2210 Debian worker. Discovery support includes LAN broadcast plus seed peers. Control-plane transport uses gRPC, and web endpoints use HTTP on port 8080. In the reported runs, all four devices were on the same Wi-Fi network. The coordinator communicated with Windows at `10.20.57.106`, Samsung at `10.20.58.133`, and Arduino at `10.20.92.41` over gRPC; the demo transport settings were `TLS_MODE=skip` and `AUTH_MODE=none`. The quantitative evidence in this revision combines two instrumented end-to-end jobs recorded on 2026-03-14, a 13-page input (job `e29a8c42...`) and a 1009-page input (job `9eb50b86...`). It also includes two book-length demo jobs recorded on 2026-03-13: *The Adventures of Sherlock Holmes* [12] (183 pages; job `dfe22d64...`) and *The Count of Monte Cristo* [3] (941 pages; job `a421988f...`). Appendix Table 10 shows that while a capable laptop-class model can beat the mesh on raw wall clock, the mesh achieves higher aggregate coverage without exceeding native device budgets.

3.2 Distributed-Scale Comparison

We use the 13-page instrumented job as a small-input baseline against the 1009-page instrumented job under the same two-group execution plan and the same per-task prompt cap. End-to-end time increased from 24.6 s to 36.9 s, while merge latency changed only from 1.4 s to 1.5 s. The main difference appears in Group 0, where the slowest worker increased from 23.2 s to 35.4 s. Worker summary output increased from 8,806 characters to 9,567 characters, and the final merged summary increased from 246 to 274 characters. Because each worker truncates extracted text before inference, runtime does not scale linearly with page count.

3.3 Routing Policy Compliance

Compliance is evaluated at the execution-plan level. Each job creates N explicit worker tasks in Group 0 (one per device) and one explicit merge task in Group 1. Across the two recorded runs, all 8 worker tasks executed on their assigned target devices (Mac, Windows, Samsung, and Arduino), and both merge tasks executed on the coordinator. No task was misrouted to an unintended device, and the Group 1 merge never started before all Group 0 workers had completed. In other words, both device-level routing assignments and the inter-group synchronization barrier operated as specified by the execution plan.

3.4 Controlled LLM Routing Comparison

The comparison mechanism isolates system behavior, not answer quality. The same `PDF_SUMMARIZE` task kind was dispatched to all four devices in both runs, producing 8/8 worker responses. Samsung was fastest in both jobs (5.3 s and 5.5 s). Mac required 11.2 s and 18.4 s, Windows 23.2 s and 13.2 s, and Arduino 22.5 s and 35.4 s. Completion order shifted from Samsung, Mac, Arduino, Windows in the first job to Samsung, Windows, Mac, Arduino in the second.

Worker summary sizes varied as well: 2,679 and 3,794 characters on Mac, 2,439 and 1,089 on Windows, 1,068 and 1,001 on Samsung, and 2,620 and 3,683 on Arduino. Because devices used different models, inference engines, and hardware classes, these numbers characterize execution latency and response throughput rather than model quality. Latency variation across runs also reflects the fact that each device allocates memory dynamically under its host operating system. The Windows laptop and Mac coordinator, for instance, run full desktop operating systems with numerous background services competing for CPU, GPU, and memory resources; their inference times are therefore sensitive to ambient system load in ways that dedicated accelerator hardware would not be.

3.5 Pipeline Execution

Both instrumented jobs used the same two-group plan. Group 0 performed four parallel `PDF_SUMMARIZE` tasks after page partitioning: 7/2/2/2 pages in the 13-page run, corresponding to 53.8%, 15.4%, 15.4%, and 15.4% across Mac, Windows, Samsung, and Arduino, and 505/168/168/168 pages in the 1009-page run, corresponding to 50.0%, 16.7%, 16.7%, and 16.7%. Group 1 then ran one `LLM_GENERATE` merge task on the coordinator. The 13-page job completed in 24.6 s and the 1009-page job in 36.9 s, with 10/10 total tasks succeeding across both runs. In both cases, the merge stage started only after every worker in Group 0 completed, and total wall-clock time was determined by the slowest worker in the parallel stage.

The same staged workflow was also exercised on two public-domain books from Project Gutenberg, *The Adventures of Sherlock Holmes* [12] and *The Count of Monte Cristo* [3]. The Sherlock run partitioned 183 pages as 93/30/30/30 across Mac, Windows, Samsung, and Arduino, produced 6,674 characters of worker output, and ended with a 3,536-character merged summary. The Monte Cristo run partitioned 941 pages as 473/156/156/156, produced 6,018 characters of worker output, and ended with a 3,198-character merged summary. Both book runs completed with 5/5 successful tasks and 0 failures. The current book-result artifact does not expose per-task timestamps, so these runs extend the evidence for workflow scalability and correct staged execution rather than timing analysis.

3.6 Failure and Safety

Quantitatively, the two instrumented PDF jobs completed with 10/10 successful tasks, 0 task failures, 0 observed worker misroutes, and 0 observed early merge starts. The two book-length demo jobs each completed with 5/5 successful tasks and 0 failures, bringing the total across all four reported jobs to 20/20 successful tasks. What the runs do show is barrier correctness in the instrumented logs and durable plan execution in the book-length artifact: the merge stage never started early in the timed runs, and all reported jobs reached `DONE` with complete task sets. Safety controls remain

implementation-level mechanisms in this artifact, including command allowlists, bounded file access, and one-time transfer tickets.

3.7 Limitations

The primary goal of this evaluation is to demonstrate end-to-end system functionality: that GRAFT can partition a real workload, route tasks to heterogeneous devices according to policy, enforce synchronization barriers, and merge results, all within a personal-device mesh and without cloud services. It is a system demonstration, not a large-scale benchmark.

The evaluation combines two instrumented timing runs with two book-length artifact runs and five post-review baseline measurements rather than a repeated trial set with statistical confidence intervals. Devices use different models and runtimes, cross-device answer quality is not directly comparable. The PDF summarization tasks truncate extracted text to a fixed prompt budget before inference, so the experiment validates distribution, routing, and staged reduction more directly than full-context document understanding. Each device runs under its own operating system with variable background load, reported latencies reflect real-world operating conditions rather than controlled benchmarking environments. The reported runs use demo transport settings (`TLS_MODE=skip`, `AUTH_MODE=none`) rather than the hardened deployment profile described in `docs/HARDENED_MODE.md`, and we do not report WAN behavior in this revision.

The current implementation also applies limited prompt engineering to the summarization and merge stages, and the chunking strategy partitions pages by capacity-weighted count without regard for semantic boundaries such as chapter or section breaks. As a result, worker summaries are functional but not as cohesive as they could be: a shard boundary may split a narrative arc or argument mid-flow, and the merge prompt does not yet compensate for this fragmentation. Improving summary coherence is a tractable engineering problem involving better chunking heuristics, context-window-aware shard sizing, overlap regions between adjacent shards, and more targeted prompt design for both the per-shard summarization and the final merge step. The post-review artifact analysis in Table 11 is useful but still limited: it covers only the saved book-length outputs, and a future revision should pair it with human-rated coherence and faithfulness scores and with failure-suite reruns under the hardened profile. Likewise, the baseline measurements in Table 10 are informative about budget-respecting coverage but do not yet constitute a repeated benchmark across multiple device classes.

4 Demo Scenario

The live demo centers on the PDF workflow. A user uploads either *The Adventures of Sherlock Holmes* [12] (183 pages) or *The Count of Monte Cristo* [3] (941 pages) through the web interface, after which the system shows extracted page count, capacity-weighted page allocation, and the two-group plan: four parallel `PDF_SUMMARIZE` tasks followed by one merge task. As execution proceeds, the UI exposes assigned device, per-task duration when available, current group, and final merged output, making the synchronization barrier between Group 0 and Group 1 visible to the audience.

We also include constrained routing requests in the demo to show fail-fast behavior for unsatisfied policies such as `REQUIRE_NPU`. The demonstration video follows the same end-to-end flow. An installable artifact and reproducibility notes are provided.[1] [2] This demo emphasizes both dimensions in the CAIS call: technical contribution and compelling interactive experience.

5 Related Work

LLM-agent research motivates structured planning, tool invocation, and multi-step execution [4][13][5][6], but these works assume a single host or uniform API backend rather than defining the multi-device routing, safety, and failure semantics required for orchestration across personal devices. General schedulers provide robust control-plane, placement, and fault-handling principles [9][10][11], yet target data-center-scale clusters with homogeneous, containerized workloads. Personal-device meshes differ fundamentally: nodes are heterogeneous, connectivity is intermittent, devices run consumer operating systems with unpredictable background load, and the workload is an end-user tool invocation rather than a container. Edge computing literature motivates heterogeneous deployment close to users [1][2], but many practical systems remain cloud-mediated. GRAFT operates entirely within the local mesh, requiring no cloud endpoint for discovery, routing, or execution.

GRAFT is positioned as a lightweight execution substrate beneath existing agent frameworks. Where a planner decides *what* to do, GRAFT handles *where* and *how*: selecting devices, enforcing compute-target preferences, gating execution through safety boundaries, and providing telemetry for an inherently unstable personal-device environment.

6 Conclusion & Future Work

GRAFT is a policy-driven execution substrate for heterogeneous personal-device meshes, combining a sessioned gRPC control plane, capability-aware routing with six policy modes, compute-target inference dispatch, constrained remote actions, and per-request telemetry. In the four-device PDF workflow, the two instrumented runs completed in 24.6 s and 36.9 s, all planned tasks succeeded, and all worker tasks executed on their intended devices while preserving the two-group synchronization barrier. Two additional book-length demonstration runs on *The Adventures of Sherlock Holmes* [12] and *The Count of Monte Cristo* [3] also finished with 5/5 successful tasks each, preserving the same staged execution contract at 183 and 941 pages. A capable single device can be faster, but the mesh scales coverage without exceeding native device budgets. These results demonstrate that agent frameworks can use GRAFT as a systems backend for multi-device workflows without re-implementing routing, safety checks, or runtime observability.

Future work includes larger meshes to characterize scaling behavior, WAN validation with NAT traversal and intermittent connectivity, calibrated cost models incorporating energy and thermal state, full NPU throughput benchmarking across accelerator families, tighter integration with upstream agent planners, improved output coherence through semantically aware chunking, context-window-aware shard sizing, shard overlap, and targeted prompt engineering. Controlled failure injection research is warranted.

[1] Demo video URL: https://youtu.be/uT27mgHBe9A.
[2] Github Url: https://github.com/ChinmayShringi/graft-analysis

Acknowledgments

The authors thank Qualcomm Incorporated (San Diego, CA) for their support and collaboration in providing the hardware infrastructure used to evaluate GRAFT.

References

[1] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30-39. DOI: https://doi.org/10.1109/MC.2017.9

[2] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637-646. DOI: https://doi.org/10.1109/JIOT.2016.2579198

[3] Alexandre Dumas and Auguste Maquet. 1844. *The Count of Monte Cristo*. Project Gutenberg, eBook #1184. Released January 1, 1998. URL: https://www.gutenberg.org/ebooks/1184

[4] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large Language Model Based Multi-agents: A Survey of Progress and Challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*. IJCAI, 8045-8053. DOI: https://doi.org/10.24963/ijcai.2024/890

[5] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR 2023)*. URL: https://openreview.net/forum?id=WE_vluYUL-X

[6] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*. URL: https://openreview.net/forum?id=Yacmpz84TH

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS 2017)*. URL: https://proceedings.neurips.cc/paper/7181-attention-is-all-you-need

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[9] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys 2015)*. ACM, Article 18. DOI: https://doi.org/10.1145/2741948.2741964

[10] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys 2013)*. ACM, 351-364. DOI: https://doi.org/10.1145/2465351.2465386

[11] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018)*. USENIX. URL: https://www.usenix.org/conference/osdi18/presentation/moritz

[12] Arthur Conan Doyle. 1892. *The Adventures of Sherlock Holmes*. Project Gutenberg, eBook #1661. Released March 1, 1999. URL: https://www.gutenberg.org/ebooks/1661

[13] Vishal Pallagani, Bharath Muppasani, Biplav Srivastava, Francesca Rossi, Lior Horesh, Keerthiram Murugesan, Andrea Loreggia, Francesco Fabiano, Rony Joseph, and Yathin Kethepalli. 2023. Plansformer Tool: Demonstrating Generation of Symbolic Plans Using Transformers. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)*. IJCAI, 7531-7535. DOI: https://doi.org/10.24963/ijcai.2023/839

[14] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2004)*. USENIX. URL: https://www.usenix.org/conference/osdi-04/mapreduce-simplified-data-processing-large-clusters

[15] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*. ACM, 29-43. DOI: https://doi.org/10.1145/945445.945450

[16] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*. USENIX. URL: https://www.usenix.org/conference/osdi-06/bigtable-distributed-storage-system-structured-data

[17] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Alex Lakshman, Avinash Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*. ACM, 205-220. DOI: https://doi.org/10.1145/1294261.1294281

[18] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35-40. DOI: https://doi.org/10.1145/1773912.1773922

[19] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2012. Spanner: Google's Globally-Distributed Database. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2012)*. USENIX. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

[20] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2011)*. USENIX. URL: https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center

[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2012)*. USENIX. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

[22] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 2014)*. USENIX, 305-319. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[23] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation Using GPT-4 with Better Human Alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*. ACL, 2511-2522. DOI: https://doi.org/10.18653/v1/2023.emnlp-main.153

[24] Liyan Tang, Philippe Laban, and Greg Durrett. 2024. MiniCheck: Efficient Fact-Checking of LLMs on Grounding Documents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*. ACL. URL: https://aclanthology.org/2024.emnlp-main.499/

[25] Yuheng Zha, Yichi Yang, Ruichen Li, and Zhiting Hu. 2023. AlignScore: Evaluating Factual Consistency with a Unified Alignment Function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023)*. ACL, 11328-11348. DOI: https://doi.org/10.18653/v1/2023.acl-long.634

A Appendix

A.1 Four-device PDF summarization testbed

Device	Platform and hardware	Representative runtime across reported jobs	Role
Mac	macOS / arm64; Apple-silicon MacBook Pro	`gemma-3-4b` via LM Studio	Coordinator, uploader, merge worker
Windows	Windows 11 / arm64; Dell Latitude 7455, Snapdragon X Elite	llama-3.2-3b-qnn INT4 via QNN NPU	Remote summary worker
Samsung	Android 16 / arm64; Galaxy S25 Ultra, Snapdragon 8 Elite	`gemma-3-4b` on-device (timed runs); `Llama 3.2 1B` via Genie SDK (book runs)	Remote summary worker
Arduino	Debian 13 / arm64; UNO Q, Dragonwing QRB2210	`gemma-3-4b` on-device (timed runs); `gemma-3-4b` via LM Studio (book runs)	Remote summary worker

Table 1: Representative run-time configuration across the timed and book-length jobs.

The coordinator communicated with Windows, Samsung, and Arduino over the same Wi-Fi network. The table above summarizes the run-time configuration reported across the current evaluation and demo jobs. *Note: Any device can assume the role of Coordinator, uploader, and merge worker.*

A.2 Network and transport configuration

Hop	Protocol and endpoint	Notes
User → Mac	HTTP JSON; TCP :8080	PDF upload and job polling
Mac → Windows	gRPC; Wi-Fi LAN, `10.20.57.106:50051`	Direct worker RPC
Mac → Samsung	gRPC; Wi-Fi LAN, `10.20.58.133:50051`	Direct worker RPC
Mac → Arduino	gRPC; Wi-Fi LAN, `10.20.92.41:50051`	Direct worker RPC
All workers	Demo settings; `TLS_MODE=skip`, `AUTH_MODE=none`	Demo-only transport profile

Table 2: Observed network configuration shared by the reported jobs.

Note: For demonstration purposes TLS was disabled. The framework is functional with TLS enabled.

A.3 Book-length PDF demonstration runs

The two book-length demonstration runs use Project Gutenberg editions of *The Count of Monte Cristo* [3] and *The Adventures of Sherlock Holmes* [12]. Both runs followed the same two-group execution plan as the instrumented timed evaluation but report only allocation and output artifacts, not per-task timestamps.

Book	Pages	Mac	Windows	Samsung	Arduino
Monte Cristo	941	473 (50.3%)	156 (16.6%)	156 (16.6%)	156 (16.6%)
Sherlock Holmes	183	93 (50.8%)	30 (16.4%)	30 (16.4%)	30 (16.4%)

Table 3: Page allocation in the two book-length PDF runs.

Book	Groups	Success	Failures	Worker output	Final output
Monte Cristo	2	5	0	6,018 chars	3,198 chars
Sherlock Holmes	2	5	0	6,674 chars	3,536 chars

Table 4: Job-level outcomes from the two book-length PDF runs.

A.4 Instrumented timed PDF runs

Job	Timing summary	Output summary
13-page PDF	Job `e29a8c42...`; 13 pages; total 24.6 s; Group 0 bottleneck 23.2 s; merge 1.4 s	Worker output 8,806 chars; final output 246 chars
1009-page PDF	Job `9eb50b86...`; 1009 pages; total 36.9 s; Group 0 bottleneck 35.4 s; merge 1.5 s	Worker output 9,567 chars; final output 274 chars

Table 5: Job-level outcomes from the two instrumented timed PDF runs.

Job	Device	Pages	Share
13-page PDF	Mac	1–7	53.8%
13-page PDF	Windows	8–9	15.4%
13-page PDF	Samsung	10–11	15.4%
13-page PDF	Arduino	12–13	15.4%
1009-page PDF	Mac	1–505	50.0%
1009-page PDF	Windows	506–673	16.7%
1009-page PDF	Samsung	674–841	16.7%
1009-page PDF	Arduino	842–1009	16.7%

Table 6: Observed page allocation for each timed run.

Device	Pages	Time (s)	Order	Output size
Samsung	10–11	5.3	1	1,068 chars
Mac	1–7	11.2	2	2,679 chars
Arduino	12–13	22.5	3	2,620 chars
Windows	8–9	23.2	4	2,439 chars
Mac merge	all sections	1.4	5	246 chars

Table 7: 13-page PDF per-task timings, output sizes, and completion order.

Device	Pages	Time (s)	Order	Output size
Samsung	674–841	5.5	1	1,001 chars
Windows	506–673	13.2	2	1,089 chars
Mac	1–505	18.4	3	3,794 chars
Arduino	842–1009	35.4	4	3,683 chars
Mac merge	all sections	1.5	5	274 chars

Table 8: 1009-page PDF per-task timings, output sizes, and completion order.

Observed outcome	13-page PDF	1009-page PDF
Worker tasks on intended devices	4/4	4/4
Merge task on coordinator	yes	yes
Early merge start	0	0
Task failures	0	0
Injected policy-rejection tests	not run	not run
Injected allowlist-denial tests	not run	not run

Table 9: Observed routing and failure outcomes in the instrumented timed logs.

In both jobs, Samsung completed first, the merge stage ran on the Mac, and total wall-clock time was determined by the slowest worker in the parallel summarization stage.

A.5 Pseudocode for compute-target router behavior

```
procedure ORCHESTRATE_LLM(req):
    t0 <- now()

    if req.compute == "cpu":
        out, err <- HTTP_GENERATE(model=req.model, prompt=req.input,
        ↪  stream=false)
    else if req.compute == "gpu":
        out, err <- HTTP_GENERATE(model=req.model, prompt=req.input,
        ↪  stream=false)
    else if req.compute == "npu":
        out, err <- PY_ACCELERATED_INFER(model=req.model, input=req.input)
        if err != nil:
            out, err <- PY_ONNX_INFER_WITH_ACCEL_EP(model=req.model,
            ↪  input=req.input)
    else:
        out <- ""
        err <- "unsupported compute target"

    dt <- millis_since(t0)
    if err != nil:
        return Response{device_id=req.device_id, compute=req.compute,
        ↪  model=req.model,
                        output="", error=str(err), duration_ms=dt}
    return Response{device_id=req.device_id, compute=req.compute,
    ↪  model=req.model,
                    output=out, error="", duration_ms=dt}
```

A.6 Pseudocode for Group-by-group orchestration

```
procedure EXECUTE_JOB(plan, routing, timeouts):
    results <- empty list
    for group in plan.groups in order:
        group_out <- parallel_map(group.tasks, lambda task:
            dev <- SELECT_DEVICE(task.device_id, routing.policy)
            ctx <- with_timeout(timeouts.task)
            return RUN_TASK_RPC(dev, task, ctx)
        )
        append results with group_out
        if any(task_result in group_out is failure):
            mark group failed
            break

    final <- REDUCE(results, plan.reduce or CONCAT)
    return final
```

A.7 13-page PDF test

Run metadata. The 13-page run was recorded on 2026-03-14 under job identifier `e29a8c42-3f4c-46b2-b491-e87c63d1a130`. The input document was the 13-page GRAFT research paper graft-paper-demo.pdf. The job reached state `DONE` with all five tasks succeeding.

Execution timeline. The run began at 17:50:53 with Group 0, which launched four parallel `PDF_SUMMARIZE` tasks. Samsung finished first at 17:50:58 after 5.3 s. The Mac completed at 17:51:04 after 11.2 s. Arduino completed at 17:51:16 after 22.5 s, and Windows completed at 17:51:16 after 23.2 s. Group 1 then started immediately on the Mac, ran the merge task, and finished at 17:51:18 after 1.4 s. The full job therefore completed in 24.6 s.

Output coverage. The Mac summary for pages 1–7 produced 2,679 characters and focused on GRAFT architecture, peer-to-peer discovery, gRPC plus TLS routing, Protocol Buffers serialization, device capability reporting, and proportional workload distribution. The Windows summary for pages 8–9 produced 2,439 characters and emphasized the experimental setup, methodology, and findings about heterogeneous network feasibility. The Samsung summary for pages 10–11 produced 1,068 characters and focused on benchmark results and comparison with distributed inference frameworks. The Arduino summary for pages 12–13 produced 2,620 characters and emphasized limitations, conclusions, speedup, data locality, and privacy. The final merged summary produced 246 characters.

Observed behavior. The run log states that all four devices performed real LLM inference rather than mock execution. Samsung was the fastest worker, Windows was the slowest worker at 23.2 s, and total wall-clock time followed that slowest parallel task. The Windows worker is reported as running `llama-3.2-3b-qnn` on the QNN NPU at roughly 17 tok/s, while Arduino completed successfully on Dragonwing QRB2210 hardware.

A.8 1k-page PDF test

Run metadata. The 1009-page run was recorded on 2026-03-14 under job identifier `9eb50b86-612f-4317-92db-57b87745498d`. The input document was `graft-1000p.pdf`, reported as 3.2 MB. The job reached state `DONE` with all five tasks succeeding.

Execution timeline. The run began at 18:16:42 with Group 0, which launched four parallel `PDF_SUMMARIZE` tasks. Samsung finished first at 18:16:47 after 5.5 s. Windows completed at 18:16:55 after 13.2 s. The Mac completed at 18:17:00 after 18.4 s. Arduino completed at 18:17:17 after 35.4 s. Group 1 then started on the Mac, ran the merge task, and finished at 18:17:19 after 1.5 s. The full job therefore completed in 36.9 s.

Output coverage. The Mac summary for pages 1–505 produced 3,794 characters and covered GRAFT architecture, peer-to-peer discovery, gRPC plus TLS routing, Protocol Buffers serialization, mesh networking, model compression techniques, and NPU hardware acceleration. The Windows summary for pages 506–673 produced 1,089 characters and emphasized the WebRTC protocol stack, ONNX interoperability, and peer-to-peer communication standards. The Samsung summary for pages 674–841 produced 1,001 characters and focused on proportional workload distribution and capacity-weighted page allocation. The Arduino summary for pages 842–1009 produced 3,683 characters and emphasized capacity-weighted distribution, model optimization, gRPC versus REST performance, federated learning, and NPU plus DSP specialization. The final merged summary produced 274 characters.

Observed behavior. The run log again states that all four devices performed real LLM inference rather than mock execution. Samsung was the fastest worker, Arduino was the slowest worker at 35.4 s, and total wall-clock time followed that slowest parallel task. The Windows worker is reported as running `llama-3.2-3b-qnn` on the QNN NPU at roughly 17 tok/s, and the run log explicitly notes that task prompts were truncated at 4,000 characters to fit device context budgets.

A.9 *The Count of Monte Cristo* book test

Run metadata. This run was recorded on 2026-03-13 under job identifier `a421988f-8d1a-4732-945b-19f0b69888be`. The input document was `the_count_of_monte_cristo.pdf`, a Project Gutenberg edition of *The Count of Monte Cristo* [3]. The PDF had 941 pages. The job used four devices and a two-group execution plan consisting of Group 0 parallel summarization and Group 1 final merge. The final state was `DONE` with five successful tasks and zero failures.

Work allocation and models. The Mac processed pages 1–473, which corresponded to 50.3% of the document. Windows processed pages 474–629, Samsung processed pages 630–785, and Arduino processed pages 786–941, with each of those three workers receiving 156 pages or 16.6% of the document. The Mac used `gemma-3-4b` through LM Studio for both its worker segment and the final merge. Windows used `Llama 3.2 3B INT4` through QNN on the Snapdragon X Elite Hexagon NPU. Samsung used `Llama 3.2 1B` through the Qualcomm Genie SDK on the Snapdragon 8 Elite Hexagon NPU. Arduino used `gemma-3-4b` through LM Studio. In this run, all workers were described as participating in the same Wi-Fi network, with Windows at `10.20.57.106`, Samsung at `10.20.58.133`, and Arduino at `10.20.92.41`.

Output coverage. The Mac summary produced 2,392 characters and covered the opening aboard the *Pharaon*, Captain Leclere's death, Edmond Dantès's early characterization, and the conspiracy involving Fernand, Danglars, and Villefort that leads to Dantès's imprisonment. The Windows summary produced 1,389 characters and covered the Count's re-entry into Parisian society, Danglars's speculative financial empire, the Andrea Cavalcanti deception, and rising tension among the Morcerf, Danglars, and Villefort families. The Samsung summary produced 1,024 characters and focused on Fernand de Morcerf's public disgrace and suicide, the Villefort poisoning crisis, and Caderousse's death at Benedetto's hands. The Arduino summary produced 1,213 characters and covered Eugénie Danglars's refusal to marry, Benedetto's courtroom exposure as Villefort's son, Danglars's capture by Luigi Vampa, and the Count's departure with Haydée. The final merged summary produced 3,198 characters and synthesized the full novel as a four-part progression from wrongful imprisonment to revenge, ruin, mercy, and moral closure.

Observed behavior. The run preserved the intended group structure: four worker summaries in Group 0 followed by one merge task on the Mac in Group 1. The artifact reports no task failures. The Windows worker note states that input tokens were capped at 1024 to avoid QNN context overflow and that MAX_OUTPUT_TOKENS=256 was used. The Samsung worker note states that its text input was capped at 2000 characters because of the smaller context budget of the 1B model. The Arduino worker note states that max tokens=256 was used to simulate a constrained edge device. The current submission artifact does not include per-task wall-clock timestamps for this run, so this appendix records allocation, model assignment, and output characteristics rather than completion order.

A.10 *The Adventures of Sherlock Holmes* book test

Run metadata. This run was recorded on 2026-03-13 under job identifier `dfe22d64-65d3-4ac7-9a6b-963598383149`. The input document was the_adventures_of_sherlock_holmes.pdf, a Project Gutenberg edition of *The Adventures of Sherlock Holmes* [12]. The PDF had 183 pages. As in the Monte Cristo run, the job used four devices and a two-group execution plan with Group 0 parallel summarization and Group 1 final merge. The final state was `DONE` with five successful tasks and zero failures.

Work allocation and models. The Mac processed pages 1–93, or 50.8% of the document. Windows processed pages 94–123, Samsung processed pages 124–153, and Arduino processed pages 154–183, with each of those workers receiving 30 pages or 16.4% of the document. The model assignments matched the Monte Cristo run: the Mac used `gemma-3-4b` through LM Studio, Windows used `Llama 3.2 3B INT4` on the Snapdragon X Elite QNN stack, Samsung used `Llama 3.2 1B` through the Qualcomm Genie SDK, and Arduino used `gemma-3-4b` through LM Studio. The workers were again described as being reachable over the same Wi-Fi network, with Windows at `10.20.57.106`, Samsung at `10.20.58.133`, and Arduino at `10.20.92.41`.

Output coverage. The Mac summary produced 3,133 characters and covered *A Scandal in Bohemia*, *The Red-Headed League*, *A Case of Identity*, *The Boscombe Valley Mystery*, *The Five Orange Pips*, *The Man with the Twisted Lip*, and the beginning of *The Blue Carbuncle*. The Windows summary produced 1,247 characters and covered

the conclusion of *The Blue Carbuncle*, *The Speckled Band*, and the beginning of *The Engineer's Thumb*. The Samsung summary produced 987 characters and covered the conclusion of *The Engineer's Thumb*, *The Noble Bachelor*, and the beginning of *The Beryl Coronet*. The Arduino summary produced 1,307 characters and covered the conclusion of *The Beryl Coronet* and *The Copper Beeches*. The final merged summary produced 3,536 characters and described the collection as a sequence of twelve cases centered on Holmes's deductive method, the Holmes-Watson partnership, and recurring themes of greed, justice, and mercy.

Observed behavior. This run again preserved the intended group structure, with four worker summaries followed by a merge on the Mac. The artifact reports five successful tasks and zero failures. The Windows worker note again states that input tokens were capped at 1024 to avoid QNN context overflow. The Samsung worker note states that its text input was capped at 2000 characters because of the smaller context budget of the 1B model. The Arduino worker note states that `MAX_OUTPUT_TOKENS=256` was used to simulate a constrained edge device. As with the Monte Cristo run, the current artifact does not include per-task wall-clock timestamps or completion order for this book-length evaluation, so the appendix records only the evidence directly present in the result files.

A.11 Single-device Coverage Baselines

We ran single-device baselines on the coordinator and on a phone-class 1B model to separate raw latency from aggregate coverage. Table 10 shows that the Mac coordinator running `gemma-3-4b` summarized a 16 KB prompt in 9.4 s, compared with 36.9 s for the four-device mesh, while the default 4 KB single-device cap is also fast, 10.4 s on the 13-page PDF and 9.8 s on the 1000-page synthetic PDF, but covers only one quarter of the mesh's 16 KB total coverage. The phone-class 1B model makes the constraint clearer: at its native 2 KB budget it runs in 7.2 s but covers only one eighth of the mesh's aggregate coverage, while at 8 KB, still only half of the mesh's coverage, runtime rises to 61.1 s and output degrades into regurgitation with visible token corruption. GRAFT's value is therefore not raw speedup over the strongest single device, but budget-respecting coverage scaling across heterogeneous devices.

Profile	Model	Coverage	Wall clock	Outcome
N=1	`gemma-3-4b`	4 KB	10.4 s	clean summary
N=1	`gemma-3-4b`	16 KB	9.4 s	clean summary
N=1	`TinyLlama 1.1B`	2 KB	7.2 s	clean summary
N=1	`TinyLlama 1.1B`	8 KB	61.1 s	degenerate output
N=4 mesh	mixed	16 KB	36.9 s	clean merge

Table 10: Single-device and mesh baselines using the post-review artifact runs. Coverage denotes total prompt budget exposed to the summarizer across the run; the mesh row corresponds to the reported 1009-page timed run, while the long single-device baseline used the 1000-page artifact PDF.

A.12 Artifact-side Output Quality

We analyzed the saved book-length summaries using three complementary metrics: G-Eval, MiniCheck, and AlignScore [23][24][25]. We restrict this analysis to the two book-length runs because the timed PDF jobs do not retain the raw worker text needed for comparable scoring. Table 11 reports the merge-stage results. The merged summaries are readable and coherent overall, while grounding-sensitive metrics remain challenging on book-length inputs. The per-worker artifact results also show stronger grounding scores for Mac and Arduino than for Windows and Samsung, which is consistent with their larger effective budgets and longer outputs. Full per-worker scores are included in `docs/output_quality_metrics.json`.

Book	MiniCheck	AlignScore	G-Cons	G-Coh
Sherlock Holmes	0.07	0.32	4.0	4.3
Monte Cristo	0.09	0.45	4.7	5.0

Table 11: Merge-stage quality metrics on the two book-length artifact runs. G-Cons and G-Coh denote G-Eval consistency and coherence on a 1–5 scale.

A.13 Supplementary evaluation tables

Run	Worker routes	Merge target	Barrier kept
13-page PDF	4/4	Mac	yes
1009-page PDF	4/4	Mac	yes

Table 12: Observed routing compliance in the two instrumented timed PDF jobs.

Case	Expected contract	Artifact check
REQUIRE_NPU, no NPU device	Admission fails fast with no worker dispatch and an explicit policy-unsatisfied error	require_npu_no_npu.sh
Allowlist denial	Non-allowlisted command is rejected without execution	allowlist_denial.sh
Mid-group worker disconnect	Affected task fails explicitly and the job reports per-task failure state	mid_group_disconnect.sh

Table 13: Artifact-side negative-path coverage included with the submission.

Profile	TLS	Access control
Demo	off (TLS_MODE=skip)	off (AUTH_MODE=none)
Hardened	on by default	bearer API_KEY, pairing-gated discovery

Table 14: Demo versus hardened deployment profiles.