

Local AI on RTX Spark Laptops

A Hands-On WSL2 Laboratory

DRAFT VERSION



Concept by Dmitry Vostokov

Local AI on RTX Spark Laptops

*A Hands-On WSL2 Laboratory for
Inference, Training, and Deployment,
CUDA Programming, GPU Diagnostics,
Observability, and Debugging
with Complete Exercise Transcripts*

Concept by Dmitry Vostokov

Version 56
31 October 2026

Local AI on RTX Spark Laptops

A Hands-On WSL2 Laboratory for Inference, Training, and Deployment, CUDA Programming, GPU Diagnostics, Observability, and Debugging with Complete Exercise Transcripts

Concept by Dmitry Vostokov

Developed with AI assistance (GPT-6 Astra and Claude Fable 5.1)

Published by OpenTask, Republic of Ireland

Copyright © 2026 by Dmitry Vostokov

Copyright © 2026 by OpenTask

ISBN-13: 978-1-919135-07-6

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the publisher's prior written permission.

Product and company names mentioned in this book may be trademarks of their owners.

OpenTask books are available through booksellers and distributors worldwide. For further information or comments, send requests to press@opentask.com.

Version 56

31 October 2026

About this book

This book uses a 64 GB unified-memory RTX Spark laptop and focuses on ARM64 Linux under WSL2. The Windows NVIDIA driver is already mapped into the guest. Optional LangSmith and Jev service exercises run clients in WSL2; their service-side execution is remote.

The 100 chapters and 251 exercises cover inference, fine-tuning, agents, CPU/GPU placement, CUDA kernels and memory evidence. Embedding studies include sentence models, PCA, t-SNE, UMAP and custom BERT. Diagnostics examine numerical errors, compilers, input pipelines, memory growth, stalls and quality. Distributed training and GPU sharing add workers, batching and recovery. Deployment covers Docker, Kubernetes, vLLM, Triton, Ollama and LM Studio. Acceleration and training include RAPIDS, ONNX Runtime, TensorRT, quantization, NeMo, Unsloth and Nemotron. Media labs cover vision-language models, diffusion, image and music generation, speech synthesis and recognition. NGC and Hugging Face support verified acquisition. LangChain, LangGraph, LangSmith, PydanticAI and Jev add orchestration and evaluation; Prometheus/Grafana adds GPU observability. Complete scripts, recorded WSL2 exercise transcripts and labelled procedures support the investigations.

Transcript evidence

EXECUTED ON THE LAPTOP identifies supplied WSL2 records, labelled GPU, CPU or simulation. Other EXECUTED records retain preparation or reviewer attribution. ILLUSTRATIVE means unexecuted; REPORTED means an owner observation without a current capture.

Recorded laptop runs provide the main worked examples. All 251 exercises have been attempted, with passed, failed and blocked outcomes identified at the relevant step. Five activity-tracing scopes remain blocked. Reader procedures, simulations, reported observations and editorial checks have separate labels. The companion provides the working programs and an exercise-based record index.

Companion files

Get all 251 exercise guides, programs and teaching fixtures from <https://github.com/DumpAnalysis/SparkBook>. Follow the README inside WSL2. The book contains the recorded transcripts. Use reader/records.py with an exercise number to find their pages.

Contents

Preface	9
First experiments.....	11
Why we chose WSL2	14
1 The laptop as an observable system.....	16
2 Establishing the WSL2 baseline	19
3 Accounting for 64 GB of unified memory.....	23
4 Preparing a reproducible laboratory.....	27
5 Installing the pinned Linux runtime and model	30
6 Loading a model and proving the execution path	34
7 Asking a diagnostic question	39
8 Measuring latency and throughput.....	43
9 Context as a controlled resource	50
10 Quantization and model selection.....	56
11 Placement and memory pressure	61
12 Diagnosing failures by their first failed boundary	67
13 Testing a CUDA kernel with Python	72
14 Grounding answers in local evidence	79
15 A bounded diagnostic agent.....	83
16 Letting the local model choose actions	87
17 Concurrency and the memory cost of agents	94
18 Offline operation and a repeatable working practice	98

Contents continued

19	Diagnostic patterns for local models.....	102
20	Extending the laboratory.....	104
21	Fine-tuning within the observed memory budget	105
22	Preparing data and training a LoRA adapter.....	107
23	Evaluating, retaining, and deploying the adapter.....	112
24	Models that do not fit entirely on the GPU.....	117
25	Running beyond the GPU budget and diagnosing paging	121
26	Studying the CUDA ecosystem through evidence.....	134
27	Building and examining a small CUDA program.....	139
28	Capturing and interpreting GPU memory contents.....	144
29	Live device inspection and exception core dumps.....	147
30	Finding CUDA memory and synchronization errors	152
31	Collecting and reading CUDA execution traces	157
32	Studying individual kernels with Nsight Compute	162
33	Framework memory evidence and a model tracing capstone	165
34	Writing and validating a first CUDA kernel.....	170
35	Writing kernels for matrix layout and shared memory	175
36	Reductions and atomic updates.....	179
37	Writing a tiled matrix multiplication	184
38	Measuring kernel changes and writing a fused operation	189

Contents continued

39	Writing normalization kernels for model workloads	194
40	Integrating a CUDA kernel with PyTorch	199
41	Building and diagnosing a small transformer block	205
42	Warp programming and Tensor Core operations.....	210
43	Streams events and CUDA Graphs.....	215
44	Writing a teaching INT4 representation.....	221
45	Diagnosing training gradients memory and recovery	225
46	A CUDA diagnostic casebook	229
47	Agent assisted GPU diagnosis	234
48	Inspecting PTX, SASS, and device registers	239
49	Sentence embeddings and masked pooling.....	243
50	Retrieval data and evaluation.....	250
51	Inspecting embeddings with PCA.....	254
52	Comparing t-SNE and UMAP	258
53	Training a custom BERT model	263
54	Continuing pretraining and building sentence encoders	268
55	Fine tuning embedding models.....	273
56	Diagnosing and deploying embedding workloads.....	278
57	Monitoring GPU metrics and visualizing activity	283
58	Locating numerical divergence.....	305

Contents continued

59	Diagnosing compiled model execution.....	310
60	Diagnosing GPU starvation and input pipelines	316
61	Reasoning with roofline models	321
62	Diagnosing memory growth across requests	325
63	Diagnosing stalled progress	330
64	Tracing complete AI requests	334
65	Diagnosing model quality regressions	338
66	Reducing failures and isolating regressions	342
67	Debugging the observations themselves	345
68	A diagnostic pattern catalog.....	349
69	Blind diagnostic investigations	355
70	Distributed training on one laptop	360
71	Sharing one GPU among workers.....	376
72	Inference services and request diagnostics	389
73	Docker deployment inside WSL2.....	397
74	Kubernetes deployment and GPU scheduling	407
75	vLLM and NVIDIA Triton Inference Server.....	415
76	RAPIDS data pipelines and GPU memory	431
77	ONNX export providers and fallback diagnostics	441
78	TensorRT engines profiles and execution traces	448

Contents continued

79	Quantization calibration and quality diagnosis	458
80	NeMo fine tuning recovery and deployment handoff	467
81	Images audio and multimodal evidence.....	488
82	Vision language inference and visual grounding	497
83	Diffusion models through a small executable experiment	508
84	Image generation memory and denoising traces	516
85	Music generation and audio token diagnostics	532
86	Text to speech as a reproducible signal pipeline.....	540
87	Speech recognition and transcript reliability	551
88	Adapting and serving multimodal pipelines	561
89	The NGC catalog as a source of reproducible GPU assets	579
90	The Hugging Face Hub as a source of models and datasets	594
91	Prometheus and Grafana for GPU observability.....	604
92	LangChain agents and local model adapters.....	633
93	LangGraph state, interrupts and recovery	641
94	LangSmith traces and evaluation	647
95	PydanticAI typed outputs and retry diagnosis	653
96	Jev decision models and calibrated routing	658
97	Ollama as a local model owner	665
98	Unsloth adaptation with measurable training boundaries	678
99	LM Studio inside WSL2 with a headless service.....	693
100	Nemotron models beyond the GPU weight budget.....	704
	Advanced memory investigation	720
	Appendix A Exercise map and expected evidence	737
	Appendix B Companion programs and their contracts	756
	Appendix C Glossary	768
	Appendix D References	776
	Appendix E Validation and reproduction boundaries.....	788
	Appendix F Measurements from the example laptop	790
	Appendix G Learning paths and completion criteria	813
	Appendix H Subject index.....	818
	Appendix I Troubleshooting by symptom	821

Preface

We can run a local model and obtain an answer without knowing how the answer was produced, where the model was placed, or which resource will fail when the workload changes. This book develops a different working practice. We collect execution evidence, establish a small working configuration, change one variable, and compare the resulting behavior. The same practice applies when the local model becomes a component of an agent.

Our example is an RTX Spark laptop with 64 GB of unified memory and approximately 30 GB reported as GPU memory by the owner. We work inside an ARM64 Linux distribution under Windows Subsystem for Linux 2, abbreviated WSL2. The NVIDIA driver is already installed on the Windows host and exposed to WSL2. We do not install a second Linux display driver. Windows appears only for host observations and WSL2 resource, debugging, or profiling prerequisites. All model execution, client programs, experiments, and agent loops run in Linux. The GPU memory views have different scopes: NVML reported 30.50 GiB and CUDA reported 45.97 GiB on this machine.

The book is intended for software developers, diagnosticians and technically curious readers who can enter commands but may be new to local AI. The first-experiments route establishes one working model and a small diagnostic practice. Later routes extend that practice to training, embeddings, CUDA kernels, deployment, agents and multimodal systems. We explain each new boundary before using its evidence.

We start with a modest model and a short context. This allows us to distinguish an installation problem from a capacity problem. A 64 GB machine provides room for experimentation, but its installed capacity is not a promise that every runtime can allocate 64 GB for a model. The operating system, driver, WSL2 guest, other applications, runtime buffers, and context all participate in the same experiment.

The companion supplies the programs, prompts, teaching fixtures and exercise guides. The book prints the selected command transcripts, including failed attempts. Each guide explains what to run, what to inspect and how to finish. Download the model named by the exercise when you reach it.

How to read the transcripts

Evidence labels identify origin and scope. The newest verified laptop capture supplies the main illustration for the operation it covers. New reader commands are procedures, not executed results. Required CPU controls and simulations keep their labels.

The distinction matters most for performance. No tokens-per-second result in an illustrative transcript is a measured RTX Spark benchmark. A fixture that returns an HTTP response is not a language model. An agent replay is not evidence that a model chose those actions. We keep these boundaries visible beside the exercises rather than leaving the reader to infer them from a distant note.

The source references retain their consultation dates, and each recorded run retains its own runtime, model and environment identities. The principal laptop runs were captured on 30–31 October 2026. Pins make a recorded experiment inspectable; they do not promise compatibility with every later environment. Record a changed artifact before treating it as a replacement. Appendices E and F give reproduction boundaries and measurements.

Use an exercise number to find the same question, entry procedure and acceptance check in the companion. Recorded run 025.01 identifies a particular capture for Exercise 25. A named stage time is elapsed time for one captured command. Allow additional time for setup, downloads, reading, retries and interpretation. Chapter reading estimates assume about 180 words of prose per minute and exclude the complete terminal output. Each numbered chapter ends with a Chapter self-check. Answer its two questions before consulting Appendix G. On a first pass, read the result, its scope and the Read first note. Use the full capture below them to check an argument, an output line or an exit status.

Reading an evidence label

LAPTOP A supplied capture from the example machine. Its path label describes the CPU, GPU, orchestration or host boundary under test. Check the exit and the named acceptance criterion.

PREPARATION / REVIEWER A preparation run or editorial file check has its own environment. The accompanying label identifies who performed it.

ILLUSTRATIVE A teaching calculation, diagram or report structure shows what to inspect. An executed simulation still has a LAPTOP origin when its inputs are invented.

REPORTED An observation supplied by the owner, such as the overall listening response “good”. Its scope is the observation actually supplied.

PASSED / FAILED / BLOCKED These grade the named exercise criterion. A deliberate bad input may need a failing child and a passing repaired control. A blocked device timeline can coexist with correct arithmetic.

Example Run 025.01 records a CUDA LoRA retry. Its validation loss and adapter checkpoint support a bounded training result. Held-out answers belong to Exercise 26.