TURN SMALL MODELS INTO POWERFUL LOCAL AI AGENTS

FINE-TUNING SMALL LANGUAGE MODELS FOR LOCAL AI AGENTS

A PRACTICAL GUIDE TO BUILDING SPECIALIZED AGENTIC SYSTEMS ON YOUR OWN HARDWARE

SELECT THE RIGHT MODEL

FINE-TUNE EFFICIENTLY

BUILD AGENTIC CAPABILITIES

DEPLOY ON YOUR OWN HARDWARE

PRACTICAL EXAMPLES • REPRODUCIBLE CODE
REAL CONFIGURATIONS • EVIDENCE-BASED RESULTS

EDWIN R. LANGFORD

Fine-Tuning Small Language Models for Local AI Agents

A Practical Guide to Building Specialized Agentic Systems on Your Own Hardware

Steve Publications

This book is available at
https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents

This version was published on 2026-08-12

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

A Practical Guide to Building Specialized Agentic Systems on Your Own Hardware

This book teaches software engineers and ML practitioners how to select, fine-tune, evaluate, and deploy small language models as capable local AI agents. You will learn the foundations of transformer architectures and tokenization, master supervised fine-tuning and parameter-efficient methods like LoRA and QLoRA, build agentic capabilities for tool use and structured output, and ship production-ready systems on consumer hardware. Every chapter includes reproducible code, concrete configurations, and evidence-based comparisons so you can move from an unmodified base model to a specialized agent running entirely on your own machine.

Introduction: Why Fine-Tune Small Models Locally?

In 2023 and early 2024, the dominant narrative in artificial intelligence was simple: bigger is better. The race for larger models produced systems with hundreds of billions of parameters, trained on trillions of tokens, deployed only behind expensive cloud APIs. If you wanted to use a capable language model, you sent your data to someone else's servers and hoped their uptime matched your deadline.

That narrative has changed. A new generation of small language models, typically in the 1B to 8B parameter range, now delivers practical capabilities for specialized tasks while running entirely on local hardware. An 8B-parameter model fine-tuned on domain-specific data can outperform a prompted 70B general-purpose model on that domain, and it does so with latency measured in milliseconds instead of seconds, zero egress costs, and complete data privacy.

This book is about making that shift concrete. It covers the full lifecycle of building local agentic systems with fine-tuned small models: from understanding what a transformer actually does internally, through selecting base models and preparing training datasets, to running fine-tuning jobs on consumer GPUs, evaluating results rigorously, and deploying production services that integrate tools, memory, and structured outputs.

The central thesis is straightforward. Fine-tuned small language models are not compromised large models. They are different tools optimized for different constraints. When you need a system that runs fast, stays private, costs nothing per inference, and specializes in your domain, a well-fine-tuned small model beats a prompted giant every time. The challenge is knowing how to make it work.

This introduction sets expectations for the journey ahead. The book assumes you are comfortable with programming and basic software engineering concepts but does not require prior expertise in language model fine-tuning. Every major concept is explained from first principles before diving into implementation. Every code example is complete and runnable. Every recommendation is evidence-based rather than aspirational.

By the end, you will be able to take a production requirement, select an appropriate small base model, prepare or discover training data, run a fine-tuning job on your available hardware, evaluate whether the result actually improved, deploy it as a local service with tool access and structured outputs, and maintain it over time. This is not theoretical knowledge. It is a practical skill set for building real systems.

Chapter 1: Foundations of Small Language Models

Before fine-tuning anything, you need to understand what a small language model actually is, how it processes text internally, and why its size determines both its capabilities and its resource requirements. This chapter establishes the technical foundation for everything that follows. No concept here is decorative. Each one directly affects your decisions about which models to use, how to prepare data, and how to train effectively.

What Is a "Small" Language Model?

The term small language model, or SLM, has no single formal definition. Different organizations draw the boundary at different points. Hugging Face describes SLMs as ranging from 1 million to 10 billion parameters [1]. IBM uses similar language: millions to a few billion parameters. Microsoft Azure defines them simply as models with fewer parameters and simpler architectures than large language models, optimized for efficiency rather than scale.

For the purposes of this book, small means roughly 1B to 8B parameters. This range captures the sweet spot for local agentic use:

- Models below 1B parameters (such as SmolLM2 at 135M or Qwen2.5-Coder at 1.5B) are useful for extremely constrained environments like mobile devices and embedded systems, but they often lack the reasoning depth needed for complex agentic tasks.
- Models in the 3B to 8B range (Llama 3.1 8B, Qwen2.5 7B, Gemma 2 9B) are the workhorses of local deployment. They fit on consumer GPUs when quantized and deliver instruction-following, tool-use, and reasoning capabilities sufficient for most practical applications.
- Models above 8B but below roughly 14B (such as Gemma 2 27B or Qwen2.5 14B) push beyond the typical local deployment boundary unless you have workstation-class hardware with multiple GPUs.

The distinction between small and large is not just about parameter count. It is about deployment context. A model is small if it can run on hardware you own: a laptop GPU, a desktop rig, or a modest server. If it requires a rented cluster of H100s, it is large regardless of its parameter count.

This definition matters because it shapes every design decision in this book. When we talk about fine-tuning small models for local agents, we are talking about systems that fit on hardware you control, run with latency you can measure directly, and operate without ongoing API costs. The constraints of smallness are not limitations to work around. They are the defining characteristics of a different class of system.

Transformer Architecture Refresher

Every modern language model, small or large, uses some variant of the transformer architecture introduced by Vaswani et al. in 2017 [2]. You do not need to understand every mathematical detail to fine-tune one effectively, but you do need enough architectural intuition to reason about what happens during training and why certain configurations work better than others.

At its core, a transformer is a stack of identical layers, each performing two main operations: attention and feed-forward computation. The input text is first converted into numerical vectors called tokens through a process we will cover in detail shortly. These token embeddings flow through the layer stack, with each layer refining the representation before passing it to the next.

The attention mechanism is what makes transformers powerful. When processing a sequence of tokens, each token can attend to every other token in the sequence, computing how much each one should influence its own representation. This allows the model to capture long-range dependencies: a word at the beginning of a paragraph can directly influence how a word at the end is understood, without information degrading through sequential processing as it would in recurrent neural networks.

More specifically, attention works through three learned projections for each token: query, key, and value vectors. For each position, the model computes the dot product between its query vector and every key vector in the sequence, scales these scores by the square root of the dimension to prevent numerical instability, applies a softmax function to produce attention weights, and then uses those weights to compute a weighted sum of all value vectors.

The result is an updated representation that incorporates relevant information from across the entire context window [3].

Multi-head attention repeats this process multiple times in parallel with different learned projections, allowing the model to attend to different kinds of relationships simultaneously: syntactic structure in one head, semantic similarity in another, positional patterns in a third. The outputs of all heads are concatenated and projected back to the original dimension.

After attention, each token representation passes through a feed-forward network: typically two linear layers with a nonlinearity such as GELU between them. This is where the model applies learned transformations to the attended representation, extracting features and building higher-level abstractions. The intermediate dimension of this network is usually larger than the hidden dimension (often 4x), giving the model capacity to compute complex functions before projecting back down.

Residual connections wrap both the attention and feed-forward operations. Instead of replacing each token's representation with the output of a layer, the model adds the layer's output to its input: output equals input plus layer(input). This allows gradients to flow directly through many layers during training, preventing the vanishing gradient problem that plagued earlier architectures.

Layer normalization stabilizes activations by normalizing each token's vector across features before and after each sublayer. Positional encoding injects information about each token's position in the sequence, since attention itself is permutation-invariant and would treat reordered tokens identically without it. Modern models use various positional encoding schemes: sinusoidal encodings in the original transformer, learned positional embeddings in many GPT-style models, rotary positional embeddings (RoPE) in Llama and Mistral families, and alibi-based relative position bias in some architectures.

For fine-tuning purposes, the key insight is this: every token representation that flows through your model during training passes through dozens of these attention-plus-feed-forward layers, each one transforming it based on patterns learned from billions of training tokens. When you fine-tune, you are adjusting those learned transformations so they better serve your specific task distribution. Understanding this flow helps explain why data quality matters more than hyperparameter tuning and why catastrophic forgetting happens when you train too aggressively on narrow data.

Tokenization and Vocabulary

Before a transformer can process text, the text must be converted into numerical tokens. This conversion is not trivial, and the tokenizer used during pretraining fundamentally shapes what the model can learn and how efficiently it processes different kinds of input.

Modern language models use subword tokenization rather than character-level or word-level tokenization. Character-level tokenization would require sequences thousands of tokens long for typical text, creating computational overhead and making it hard for the model to capture meaningful linguistic units. Word-level tokenization would require enormous vocabularies to handle rare words and out-of-vocabulary terms gracefully. Subword tokenization strikes a balance: common words are single tokens, rare words decompose into recognizable subunits, and truly unknown strings still tokenize predictably.

The three dominant subword algorithms are Byte Pair Encoding (BPE), WordPiece, and the Unigram language model. All three start with a character-level vocabulary and iteratively build up subword units, but they differ in how they decide which merges or units to keep.

BPE works bottom-up by repeatedly merging the most frequent adjacent token pairs in the training corpus until reaching a target vocabulary size. GPT-2, Llama 3, and many other decoder-only models use BPE-based tokenizers. Llama 3's tokenizer specifically is based on tiktoken, a fast Rust implementation that operates on raw bytes rather than Unicode characters, making it robust to any input encoding [4].

WordPiece, used by BERT and related encoder models, selects each merge to maximize the corpus likelihood under a unigram language model rather than raw frequency. The practical difference from BPE is small for most purposes.

The Unigram algorithm works top-down: it starts with an overcomplete vocabulary of all possible subword units and iteratively prunes the least-likely ones using expectation-maximization until reaching the target size. T5 and mBART use Unigram tokenizers.

SentencePiece is not itself a tokenization algorithm but a framework that implements BPE, WordPiece, and Unigram with language-independent training. It preprocesses text by replacing spaces with a special visible character before tokenization, preserving word boundary information without requiring

language-specific rules about what constitutes a word. Many models use SentencePiece-trained tokenizers [5].

Vocabulary size is an important architectural choice. Larger vocabularies mean fewer tokens per sequence, reducing computational cost during both training and inference. However, larger vocabularies also mean larger embedding tables that consume more memory and require more training data to learn effectively. Llama 3 uses a vocabulary of roughly 128K tokens, significantly larger than GPT-2's 50K but smaller than some specialized tokenizers.

For fine-tuning, the tokenizer matters in three ways:

First, you generally cannot change the tokenizer after pretraining without retraining the embedding layer from scratch. Fine-tuning adjusts weights assuming a fixed vocabulary and tokenization behavior. If your domain uses many domain-specific terms that tokenize into dozens of subword fragments, the model has to learn their meaning from scattered pieces rather than coherent units.

Second, the number of tokens per training example directly affects batch size, memory usage, and training time. A dataset with verbose examples in a language that tokenizes inefficiently will train slower than a concise dataset in a well-supported language, even if the semantic information content is identical.

Third, tokenizer behavior affects how you format training data. Special tokens such as end-of-sequence markers, padding tokens, and task-specific control tokens must be used consistently with the model's expectations. Different model families use different chat templates that arrange system messages, user prompts, and assistant responses using specific token sequences. Getting these wrong leads to training instability or models that fail to follow instructions at inference time.

Context Windows and Memory

The context window is the maximum number of tokens a model can process in a single forward pass. It determines how much prior information the model can attend to when generating each new token. Llama 3.1 8B supports 128K tokens natively. Qwen2.5 models also offer 128K context across their size range. Some newer models claim support for million-token contexts, though practical quality at those lengths remains an open question.

Context length has a direct and linear impact on memory usage during inference through the key-value cache, or KV cache. Every time the model processes a token, it computes key and value vectors for that position in every attention layer. Rather than recomputing these vectors each time a new token is generated, the model caches them. This cache grows with every token in the context window.

The memory required by the KV cache follows a precise formula:

KV bytes equals 2 times layers times kv_heads times head_dim times sequence_length times bytes_per_element

For Llama 3.1 8B with 32 layers, 32 attention heads, and 128-dimensional heads running at FP16 precision (2 bytes per element), a 32K-token context requires roughly 4.3 GB of VRAM just for the KV cache [6]. This is nearly as much memory as the model weights themselves when quantized to Q4 (roughly 4.9 GB). At 128K tokens, the KV cache alone would require about 17 GB, exceeding what many consumer GPUs can provide.

This linear relationship between context length and memory usage means that supporting a large theoretical context window does not guarantee you can use it in practice on your hardware. A model that supports 128K context but runs on a 12 GB GPU may be limited to roughly 14K tokens in practice after accounting for weights and overhead [7].

Several techniques mitigate KV cache pressure:

Grouped-query attention (GQA) reduces the number of key-value heads relative to query heads. Llama 3.1 uses GQA with 8 key-value groups, reducing KV cache memory by a factor of 4 compared to multi-head attention with equal heads and values. Multi-query attention (MQA) takes this further by using a single shared set of key-value heads, but at the cost of some quality degradation.

KV cache quantization stores cached keys and values at lower precision than the model weights. Ollama supports 4-bit KV cache quantization via environment variables, reducing cache memory by roughly 75 percent [8]. The quality impact is typically small for most applications.

PagedAttention, introduced in vLLM, treats the KV cache like virtual memory: it allocates cache entries in fixed-size blocks rather than contiguous arrays, eliminating fragmentation and allowing more efficient packing of multiple sequences with different lengths into the same GPU memory [9]. This alone can

increase effective throughput by 2x to 24x under concurrent workloads.

For fine-tuning, context length decisions affect both training cost and model behavior. Training on longer sequences requires more VRAM per batch due to larger activation tensors during attention computation (which scale quadratically with sequence length unless techniques like flash attention are used). If your use case does not require long contexts, training with shorter max lengths lets you use larger batch sizes or fit the model on smaller hardware.

However, if your agent will regularly process long documents or multi-turn conversations, training with appropriate context lengths matters. A model trained only on short sequences may fail to attend effectively to distant tokens even if its architecture supports longer windows. Positional encoding generalization is not guaranteed beyond the lengths seen during training.

Embeddings and Representations

Embeddings are the numerical representations that transformers actually compute with. The word embedding layer maps each token ID to a dense vector in a high-dimensional space, typically 4096 dimensions for an 8B-parameter model. These vectors are learned during pretraining to encode semantic information: similar tokens end up close together in this space, and arithmetic relationships can emerge (king minus man plus woman approximates queen).

During forward propagation, each token's representation evolves as it passes through transformer layers. Early layers tend to capture local syntactic patterns and surface features. Middle layers build more abstract semantic representations. Later layers encode task-specific information and prepare outputs for the final projection back to token probabilities. This hierarchical structure is not rigid but represents a general trend observed through mechanistic interpretability research [10].

Fine-tuning changes these embeddings and all subsequent layer weights in ways that reflect your training distribution. When you fine-tune on medical text, the model's representations of terms like prognosis or contraindication become more nuanced and contextually appropriate for that domain. When you fine-tune for tool use, the model learns to associate certain input patterns with structured function-call outputs rather than free-form prose.

The embedding layer itself is often a significant portion of total model parameters. For Llama 3.1 8B with a 128K vocabulary and 4096-dimensional em-

beddings, the embedding table alone contains roughly 524 million parameters, about 6 percent of the total. During full fine-tuning, these parameters update along with everything else. During LoRA-based fine-tuning, the embedding layer is typically frozen unless explicitly targeted.

Understanding embeddings helps explain several practical phenomena:

Catastrophic forgetting happens when fine-tuning on narrow data causes representations to shift away from previously learned patterns. If you train aggressively on code examples, the model's ability to write natural prose may degrade because the shared representation space has been reorganized to prioritize coding patterns.

Transfer learning works because early-layer representations capture general linguistic structure that is useful across domains. A model pretrained on web text already understands syntax, basic semantics, and world knowledge. Fine-tuning primarily adjusts later layers to specialize this general foundation for specific tasks, which is why relatively small amounts of task-specific data can produce large improvements.

Domain vocabulary matters because if your domain uses terms not well-represented in the pretraining vocabulary, the embedding layer has no good starting point for representing them. Continued pretraining on domain-corpus text before supervised fine-tuning can help the model learn better embeddings for domain-specific terminology.

The Size–Capability–Cost Trade-off Curve

The relationship between model size, capability, and cost is not linear and not uniform across tasks. Understanding this trade-off curve helps you decide when a small model is sufficient and when it is not.

Scaling laws describe how model performance improves with size, data, and compute. The original scaling law from Kaplan et al. at OpenAI showed that test loss decreases as a power law of model parameters, training tokens, and total compute [11]. The Chinchilla paper by DeepMind refined this finding, showing that existing models were significantly undertrained: for compute-optimal training, model size and training data should scale approximately equally, with roughly 20 tokens trained per parameter [12].

These laws imply that larger models are inherently more capable given sufficient training. However, they also imply diminishing returns. Going from

7B to 70B parameters improves performance substantially, but going from 70B to 405B provides smaller relative gains for much larger absolute cost increases.

For small models specifically, recent benchmarks show impressive capability at the 7B to 8B range:

Llama 3.1 8B achieves roughly 66 percent on MMLU (a broad knowledge benchmark) and nearly 80 percent on GSM8K (grade-school math reasoning) [13]. Qwen2.5 7B reaches 74 percent on MMLU, nearly 50 percent on MATH, and 58 percent on HumanEval (coding) [14]. These scores are not frontier-level but they are genuinely useful for many practical tasks.

The key insight for local agentic systems is specialization. A fine-tuned 7B model trained extensively on your domain's data, tools, and interaction patterns can outperform a prompted 70B general-purpose model on that specific domain. The large model has broader knowledge but no deep understanding of your particular context. The small model, properly trained, encodes your domain directly in its weights.

This does not mean small models are universally better. They have real limitations:

- Narrower general knowledge: they may fail on questions outside their training distribution that a larger model would answer from pretraining alone.
- Weaker reasoning on novel problems: complex multi-step reasoning that requires synthesizing unfamiliar concepts is harder for smaller models.
- Less robust instruction following: edge cases in prompt formatting or ambiguous instructions are more likely to produce poor outputs.
- Higher sensitivity to data quality: with fewer parameters to absorb noise, bad training data degrades small models more severely.

The trade-off decision framework is straightforward:

Use a small fine-tuned model when your task has a well-defined domain, stable requirements, and benefits from low latency, privacy, or cost constraints. Use a large prompted model (via API) when you need broad general knowledge, must handle highly variable inputs, or cannot invest in building training data and evaluation pipelines. Use both when you can: fine-tune a small model for your core domain tasks and route edge cases to a larger model.

The rest of this book shows you how to make the small-model path work in practice.

Chapter 2: The Local AI Ecosystem and Model Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Major Small Language Model Families

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Base Models Versus Instruction-Tuned Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Quantization Formats and Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Inference Engines and Runtimes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Hardware Realities: GPUs, CPUs, and Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Selecting Your Stack: A Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 3: Data Strategy for Fine-Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Fine-Tuning Versus Prompting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Fine-Tuning Versus RAG

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Fine-Tuning Versus Continued Pretraining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Fine-Tuning Versus Distillation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

The Data Quality Hierarchy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Licensing, Provenance, and Legal Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 4: Building Fine-Tuning Datasets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Discovering Existing Datasets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Instruction Dataset Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Building Domain-Specific Datasets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Tool-Use and Function-Calling Datasets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Synthetic Data Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Dataset Cleaning, Validation, and Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 5: Supervised Fine-Tuning Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

How Supervised Fine-Tuning Works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Setting Up Your Training Environment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

The Hugging Face Ecosystem for Fine-Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Configuring a Training Run

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Estimating Resource Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Running Your First Fine-Tuning Job

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 6: Parameter-Efficient Fine-Tuning Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Why Parameter-Efficient Fine-Tuning?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

LoRA: Low-Rank Adaptation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

QLoRA: Quantized Low-Rank Adaptation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Other Adapter Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Configuring PEFT for Different Tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Merging and Exporting Adapters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 7: Training Operations, Experiment Management, and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Hyperparameter Selection Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Training Stability and Convergence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Experiment Tracking and Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Checkpoint Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Debugging Common Failure Modes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Reproducibility Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 8: Evaluating Fine-Tuned Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Why Systematic Evaluation Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Automated Benchmark Suites

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Task-Specific Evaluation Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Tool-Calling and Structured Output Evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Regression Testing and Ablation Studies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Human Evaluation When It Counts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 9: Specializing Models for Agentic Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

What Makes a Model "Agentic"?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Training for Tool Use and Function Calling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Training for Structured Outputs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Reasoning Specialization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Routing and Intent Classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Multi-Agent Training Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 10: Building Local AI Agents with Fine-Tuned Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

The Agent Loop Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Tool Interfaces and MCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Memory and State Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Retrieval-Augmented Generation Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Constrained Decoding and Output Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Security, Sandboxing, and Permissions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 11: Production Deployment and Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Model Packaging and Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Serving Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Latency and Throughput Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Memory Optimization in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Monitoring and Observability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Performance Profiling and Bottleneck Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Chapter 12: Long-Term Maintenance, Security, and Governance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Iterative Fine-Tuning and Continual Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Catastrophic Forgetting: Causes and Mitigation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Dataset Drift and Model Decay

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Security Auditing Fine-Tuned Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Model Provenance and Compliance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Organizational Practices for Model Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

Conclusion: The Framework for Local Agentic Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/fine-tuningsmalllanguagemodelsforlocalaiagents.