140509_21.md README

Model Quantization and Fine-tuning Platform

Summary: Develop a platform that enables efficient model quantization and fine-tuning for deploying large language models on resource-constrained environments. Problem Statement: Large language models require significant computational resources, limiting their deployment in edge environments. Your task is to create a platform that automates model quantization, fine-tuning, and optimization for specific use cases while maintaining performance quality. The system should support various quantization techniques, provide performance benchmarking, and enable easy deployment to different hardware configurations. Steps: â€¢ Design automated quantization pipeline supporting multiple techniques (INT8, INT4, dynamic) â€¢ Implement fine-tuning workflows with parameter-efficient methods (LoRA, QLoRA) â€¢ Create performance benchmarking suite measuring accuracy, speed, and memory usage â€¢ Build deployment optimization for different hardware targets (CPU, GPU, mobile) â€¢ Develop model comparison and selection tools based on constraints â€¢ Include monitoring and quality assessment for quantized models Suggested Data Requirements: â€¢ Pre-trained model checkpoints and configuration files â€¢ Domain-specific fine-tuning datasets â€¢ Hardware performance benchmarks and constraints â€¢ Quality evaluation datasets for model comparison Themes: GenAI & its techniques, Quantization, Fine-tuning The steps and data requirements outlined above are intended solely as reference points to assist you in conceptualising your solution. PRD (Product Requirements Document) Product Vision and Goals To democratize LLM deployment on edge devices by automating optimization, reducing model size by 4x-8x while retaining >95% accuracy. Goals: Support 10+ quantization methods, integrate with 5 hardware types, and provide one-click deployment. Target Audience and Stakeholders

Primary Users: ML engineers, mobile app developers, IoT specialists. Stakeholders: Hardware vendors for benchmarks, end-users for inference. Personas: An edge AI developer optimizing GPT-J for Raspberry Pi.

Key Features and Functionality

Auto-quantization with technique selection. PEFT (Parameter-Efficient Fine-Tuning) workflows. Multi-metric benchmarking dashboard. Hardware-specific exporters (e.g., TFLite for mobile). Model selector with constraint-based ranking. Post-deployment monitoring for drift.

Business Requirements

Open-source with enterprise edition for cloud integration. Integration with Hugging Face Hub for model loading.

Success Metrics

Efficiency: >2x speed-up on target hardware. User Adoption: 1000+ downloads in first year. Quality: Perplexity <5% increase post-quantization.

Assumptions, Risks, and Dependencies

Assumptions: Users have basic PyTorch knowledge. Risks: Accuracy loss in quantization; mitigate with calibration datasets. Dependencies: Libraries like bitsandbytes for QLoRA, public models from HF.

Out of Scope

Custom hardware acceleration (e.g., FPGA design). Online learning during inference.

FRD (Functional Requirements Document) System Modules and Requirements

Quantization Pipeline (FR-001):

Input: Model checkpoint, calibration data. Functionality: Support PTQ (Post-Training Quant), QAT; techniques: static INT8, dynamic, FP16. Output: Quantized model with config.

Fine-Tuning Workflow (FR-002):

Input: Quantized model, dataset. Functionality: Apply LoRA/QLoRA; trainers with PEFT library. Output: Adapted model adapters.

Benchmarking Suite (FR-003):

Input: Models, eval dataset, hardware spec. Functionality: Measure accuracy (e.g., BLEU), latency (ms), memory (MB), power (if sim). Output: Comparative reports, graphs.

Deployment Optimization (FR-004):

Input: Model, target (CPU/GPU/Android). Functionality: Convert to ONNX/TFLite/CoreML; optimize ops. Output: Deployable binary.

Model Comparison (FR-005):

Input: Multiple models, constraints (e.g., max 1GB RAM). Functionality: Rank by Pareto front (accuracy vs size). Output: Recommended model.

Interfaces and Integrations

UI: Web app for uploading, visualizing benchmarks. API: CLI commands like quantize â€“model gpt2 â€“tech int8. Data Flow: Load model -> Quantize -> Fine-tune -> Benchmark -> Deploy -> Monitor.

Error Handling and Validation

Validation: Auto-check accuracy drop; rollback if >threshold. Errors: Handle incompatible hardware with warnings.

NFRD (Non-Functional Requirements Document) Performance Requirements

Process Time: Quantize 7B model <30min on V100 GPU. Inference: <50ms/token on mobile.

Scalability and Availability

Handle models up to 70B params. Cloud deployable with auto-scaling.

Security and Privacy

Secure model uploads; no data retention. Compliance: MIT license for open components.

Reliability and Maintainability

Fault Tolerance: Resume interrupted fine-tuning. Code: 90% coverage, modular plugins for new techs.

Usability and Accessibility

Intuitive GUI with tutorials. Support dark mode, screen readers.

Environmental Constraints

Run on CPU-only for low-end users.

AD (Architecture Diagram) text+â€”â€”â€”â€”â€”â€“+ | User Interface/CLI | (Streamlit: Upload, Config, Viz Dashboard) +â€”â€”â€”â€”â€”â€“+ | v +â€”â€”â€”â€”â€”â€“+ | Workflow Orchestrator | (Airflow/Dagster: Pipeline Management) +â€”â€”â€”â€”â€”â€“+ / | | v v v v +â€“+ +â€“+ +â€“+ +â€“+ | Quant| | Fine| | Bench| | Deploy| | Pipe | | Tune | | Suite| | Opt | +â€“+ +â€“+ +â€“+ +â€“+ | | v v +â€”â€”â€”â€”â€”â€“+ +â€”â€”â€”â€”â€”â€“+ | Model Registry | | Monitoring Agent | | (HF Hub) | | (Prometheus) | +â€”â€”â€”â€”â€”â€“+ +â€”â€”â€”â€”â€”â€“+ HLD (High Level Design)

Components:

Orchestrator: Use Hugging Face Accelerate for distributed. Quant: Torch.quantization, bitsandbytes. Benchmark: Torch Profiler, hardware sims. Deployment: ONNX Runtime.

Design Patterns:

Factory for quantization types. Observer for monitoring.

Data Management:

Datasets: Alpaca for fine-tune, GLUE for eval.

High-Level Flow:

Config input -> Run pipeline stages sequentially or parallel. Store artifacts in registry.

LLD (Low Level Design)

Quantization LLD:

Static INT8: model = torch.quantization.quantize(model, qconfig_spec, inplace=False) Calibration: Run forward passes on 1000 samples.

Fine-Tuning LLD:

LoRA Config: from peft import LoraConfig; config = LoraConfig(r=16, lora_alpha=32) Trainer: from transformers import Trainer; trainer.train()

Benchmark LLD:

Accuracy: from evaluate import load; acc = load(â€œaccuracyâ€).compute(preds, refs) Latency: with torch.profiler.profile(): model(input); print(profile.key_averages())

Comparison LLD:

Pareto: Use scipy.optimize for multi-objective ranking.

Pseudocode textclass QuantFinePlatform: def **init**(self): self.hf_hub = HFHub()

```
def quantize(self, model_name, tech='int8', calib_data):
    model = self.hf_hub.load_model(model_name)
    if tech == 'int8':
        q_model = torch.quantization.quantize_dynamic(model, {nn.Linear: torch.qint8})
    elif tech == 'int4':
        q_model = bitsandbytes.quantize(model, 4)
    q_model.calibrate(calib_data)
    return q_model

def fine_tune(self, q_model, dataset, method='qlora'):
    config = LoraConfig(...) if method == 'lora' else QLoRAConfig(...)
    peft_model = get_peft_model(q_model, config)
    trainer = Trainer(peft_model, train_dataset=dataset, eval_dataset=val)
    trainer.train()
    return peft_model

def benchmark(self, models, eval_data, hardware='cpu'):
    results = []
    for m in models:
        acc = evaluate_model(m, eval_data)
        lat, mem = profile_inference(m, hardware)
        results.append({'acc': acc, 'lat': lat, 'mem': mem})
    return results

def deploy(self, model, target='mobile'):
    if target == 'mobile':
        converted = convert_to_tflite(model)
    return converted

def compare(self, benchmarks, constraints):
    filtered = [b for b in benchmarks if b['mem'] < constraints['max_mem']]
    ranked = sort_by_pareto(filtered, keys=['acc', '-lat'])
    return ranked[0]
```