Training Pipeline Engineering: Building a Domain-Specific Medical Language Model

Aethon Health -- Automating Radiology Report Generation for Community Hospitals

---

Section 1: Industry Context and Business Problem

The Company

Aethon Health is a Series A healthcare AI startup (\$18M raised) building AI-assisted radiology workflows for community hospitals -- facilities with 50-300 beds that lack the staffing of major academic medical centers. Their flagship product, **RadAssist**, helps radiologists draft preliminary reports from medical imaging studies (X-rays, CT scans, MRIs).

The current RadAssist pipeline uses a retrieval-augmented approach: given structured findings from a computer vision module, it retrieves similar historical reports and adapts them. This works for routine cases but fails on complex, multi-finding studies where no close historical match exists.

The Problem

Aethon's Chief Medical Officer has flagged a critical gap: **35% of radiology studies involve multi-system findings** (e.g., a chest CT showing both pulmonary nodules and cardiac enlargement), and RadAssist's retrieval approach produces incoherent reports for these cases. Radiologists end up rewriting these reports from scratch, eliminating any time savings.

The engineering team has proposed training a **domain-specific language model** on 2.3 million de-identified radiology reports to generate coherent multi-finding reports directly. The model must:

- Generate reports using proper radiology terminology and structure
- Handle domain-specific vocabulary (e.g., "pneumoperitoneum", "hepatosplenomegaly", "ground-glass opacities")
- Produce reports in under 3 seconds on a single A100 GPU (inference latency constraint)
- Train on 4 A100 GPUs within a \$12,000 compute budget

Constraints

Constraint	Value	Rationale
Training budget	\$12,000 (4x A100 for ~72 hours)	Series A runway limits
Inference latency	< 3 seconds per report	Radiologist workflow integration
Model size	350M-750M parameters	Must fit in single A100 (40GB) at inference
Training data	2.3M reports (~1.8B tokens)	De-identified hospital partner data
Vocabulary coverage	> 99.5% of domain terms	Medical safety requirement
Training stability	Zero divergence events	Cannot afford restarts on limited budget

Business Impact

If successful, RadAssist v2 would: - Reduce radiologist report time from 12 minutes to 4 minutes for complex cases - Expand addressable market from 800 to 2,200 community hospitals - Generate an estimated \$8.4M in additional ARR within 18 months

Why This Is a Training Pipeline Problem

The Transformer architecture (GPT-style decoder) is well-understood. The challenge is entirely in the **training pipeline**:

1. **Tokenization:** Medical vocabulary contains thousands of rare compound terms that standard BPE tokenizers fragment into meaningless pieces, wasting model capacity.
2. **Data loading:** Reports vary from 50 to 2,000 tokens. Naive padding wastes 40% of compute on padding tokens.
3. **Optimization:** Domain-specific training from scratch (not fine-tuning) requires careful learning rate scheduling to avoid divergence, especially with limited compute budget.
4. **Gradient stability:** Medical text contains extreme vocabulary distribution skew (some terms appear millions of times, others fewer than 100 times), causing gradient spikes.

Section 2: Technical Problem Formulation

From First Principles

We are training an autoregressive language model P_θ that maximizes the likelihood of radiology reports:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\log P_\theta(x_t^{(i)} \mid x_1^{(i)}, \ldots, x_{t-1}^{(i)})$$

where N is the number of reports, T_i is the length of report i, and $x_t^{(i)}$ is the token at position t.

The Tokenization Challenge

Standard GPT-2 BPE tokenization handles medical text poorly:

Term	GPT-2 Tokens	Count
pneumoperitoneum	["pne", "um", "oper", "iton", "eum"]	5
hepatosplenomegaly	["hep", "atos", "pl", "en", "om", "eg", "aly"]	7
cardiomegaly	["card", "iom", "eg", "aly"]	4
ground-glass opacities	["ground", "-", "glass", " opac", "ities"]	5

This fragmentation means: (a) sequences become 30-40% longer than necessary, increasing attention compute quadratically; (b) the model must learn that "pne" + "um" + "oper" + "iton" + "eum" = a medical condition, wasting model capacity.

Our approach: Train a domain-specific BPE tokenizer on the medical corpus with a vocabulary of 32,768 tokens. This preserves common medical terms as single tokens while maintaining general English coverage.

Metric: Domain Vocabulary Coverage

We define domain vocabulary coverage as:

$$\text{Coverage}(V) = \frac{|\{w \in D : w \text{ is a single token in } V\}|}{|D|}$$

where D is a curated list of 4,200 essential radiology terms. Our target is Coverage > 99.5%.

The Data Loading Challenge

Radiology reports have highly variable lengths:

- Chest X-ray reports: ~80 tokens (short)
- Abdominal CT reports: ~350 tokens (medium)
- Complex multi-system reports: ~1,200 tokens (long)

With a fixed context length of 512 tokens: - Short reports waste 80% of each sequence on padding - Long reports must be truncated or split

Our approach: Dynamic batching with sequence packing -- concatenate multiple short reports into a single 512-token sequence with separator tokens, and split long reports across multiple sequences.

The packing efficiency is:

$$\text{Efficiency} = \frac{\text{Total real tokens in batch}}{\text{Total positions in batch}} = \frac{\sum_i T_i}{B \times L}$$

where B is batch size and L is context length. Without packing, efficiency is ~58%. With packing, we target >92%.

The Optimization Challenge

Training a 500M parameter model from scratch on 1.8B tokens with a budget of 72 GPU-hours requires:

$$\text{Tokens per second} = \frac{1.8 \times 10^9 \text{ tokens} \times 3 \text{ epochs}}{72 \times 3600 \text{ seconds}} \approx 20,800 \text{ tokens/sec/GPU}$$

This is achievable with mixed-precision training (bfloat16) and an effective batch size of ~512 sequences (via gradient accumulation).

Baseline and Target Metrics

Metric	Baseline (Retrieval)	Target (LM)
Report coherence (physician rating 1-5)	3.1	> 4.2
Multi-finding accuracy	62%	> 88%
Validation perplexity	N/A	< 12.0
Inference latency	1.8s	< 3.0s
Training compute	N/A	< \$12,000
Domain vocabulary coverage	84%	> 99.5%

Loss Function

We use standard cross-entropy loss with a modification: **masked loss** that ignores padding tokens and separator tokens in packed sequences.

$$\mathcal{L}_{\text{masked}} = -\frac{1}{\sum_t m_t} \sum_{t=1}^{T} m_t \cdot \log P_\theta(x_t \mid x_{<t})$$

where $m_t \in \{0, 1\}$ is the mask (1 for real tokens, 0 for padding/separators).

Section 3: Implementation Notebook Structure

3.1 Environment Setup and Data Loading

```
# Install dependencies
!pip install tiktoken torch matplotlib numpy

# Load and inspect the medical report dataset
# (Using synthetic data that mimics radiology report structure)

import torch
import numpy as np
```

```
# Simulate the report length distribution
np.random.seed(42)
report_lengths = np.concatenate([
    np.random.normal(80, 20, 5000),     # Short (chest X-ray)
    np.random.normal(350, 80, 3000),    # Medium (CT)
    np.random.normal(1200, 200, 1000),  # Long (complex)
]).astype(int)
report_lengths = np.clip(report_lengths, 20, 2000)

print(f"Total reports: {len(report_lengths):,}")
print(f"Mean length: {report_lengths.mean():.0f} tokens")
print(f"Median length: {np.median(report_lengths):.0f} tokens")
print(f"Min/Max: {report_lengths.min()}/{report_lengths.max()}")
```

3.2 Domain-Specific BPE Tokenizer

```
# TODO: Train a BPE tokenizer on the medical corpus
# - Start with byte-level vocabulary
# - Run BPE merges prioritizing medical terms
# - Target vocabulary size: 32,768
# - Measure domain vocabulary coverage

# Key question: How does vocabulary size affect
# the tradeoff between sequence length and embedding memory?

def train_medical_bpe(corpus, vocab_size=32768):
    """
    Train a domain-specific BPE tokenizer.
    YOUR CODE HERE
    """
    pass
```

3.3 Sequence Packing Implementation

```
# TODO: Implement dynamic sequence packing
# - Pack multiple short reports into single 512-token sequences
# - Use a separator token between reports
# - Generate attention masks that prevent cross-report attention
# - Measure packing efficiency

class PackedDataset(torch.utils.data.Dataset):
    """
    Pack variable-length reports into fixed-size sequences.
    YOUR CODE HERE
    """
    pass
```

3.4 Efficient DataLoader with Packing

```
# TODO: Build a DataLoader that:
# - Sorts reports by length for efficient packing
# - Creates packed sequences with separator tokens
# - Generates attention masks
# - Shuffles at the pack level (not report level)
# - Measures and reports packing efficiency per batch
```

3.5 Model Architecture

```
# TODO: Implement a GPT-style decoder-only Transformer
# - 500M parameters (24 layers, 16 heads, d_model=1024)
# - Rotary positional embeddings (RoPE)
# - Pre-norm (LayerNorm before attention and FFN)
# - Weight tying (embedding = output projection)

class MedicalLM(torch.nn.Module):
    """
    GPT-style language model for medical report generation.
```

```
    YOUR CODE HERE
    """
    pass
```

3.6 Training Loop with Warmup + Cosine Decay

```
# TODO: Implement the complete training loop
# - AdamW optimizer (lr=3e-4, beta1=0.9, beta2=0.95, wd=0.1)
# - Linear warmup for 2000 steps
# - Cosine decay to lr_min=1e-5
# - Gradient clipping at max_norm=1.0
# - Mixed precision (bfloat16) for 2x speed
# - Gradient accumulation to simulate batch size 512

def train_epoch(model, dataloader, optimizer, scheduler, scaler, device,
                accumulation_steps=8, max_grad_norm=1.0):
    """
    One epoch of training with all stability mechanisms.
    YOUR CODE HERE
    """
    pass
```

3.7 Evaluation and Perplexity

```
# TODO: Compute validation perplexity
# - Use masked loss (ignore padding and separator tokens)
# - Track per-report-type perplexity (short, medium, long)
# - Compare against GPT-2 baseline tokenizer

@torch.no_grad()
def evaluate(model, val_loader, device):
    """
    Compute validation perplexity with masked loss.
    YOUR CODE HERE
    """
    pass
```

3.8 Training Stability Analysis

```
# TODO: Monitor and visualize training stability
# - Track gradient norm distribution over training
# - Identify gradient spike events
# - Compare training with and without gradient clipping
# - Analyze loss landscape smoothness via learning rate sensitivity

def stability_analysis(training_metrics):
    """
    Analyze and visualize training stability.
    YOUR CODE HERE
    """
    pass
```

3.9 Report Generation and Qualitative Evaluation

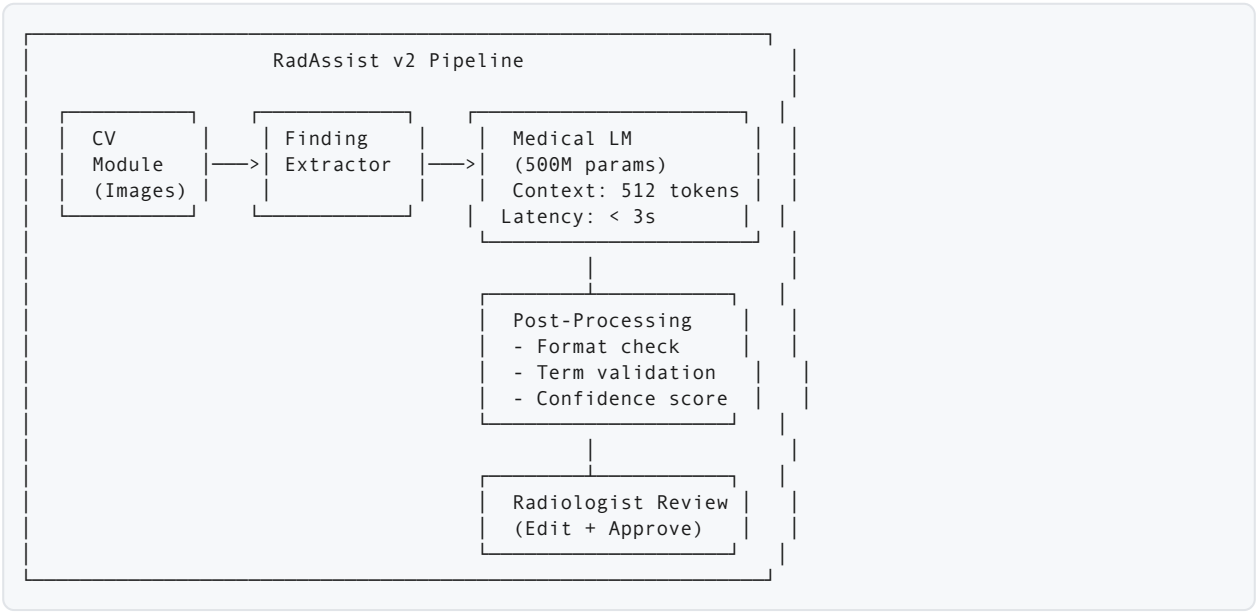
```
# TODO: Generate sample reports and evaluate quality
# - Generate reports from structured findings
# - Compare domain tokenizer vs standard tokenizer outputs
# - Measure inference latency
# - Qualitative assessment of medical terminology usage

def generate_report(model, tokenizer, findings, max_length=512,
                    temperature=0.7, top_p=0.9):
    """
    Generate a radiology report from structured findings.
    YOUR CODE HERE
```

```
    """
    pass
```

Section 4: Production and System Design Extension

Deployment Architecture



Scaling Considerations

Data pipeline scaling: As Aethon onboards more hospital partners, the training corpus will grow from 2.3M to 10M+ reports. The sequence packing approach scales linearly -- more reports simply produce more packed sequences. The BPE tokenizer should be retrained periodically (quarterly) to capture new medical terminology.

Model scaling: The 500M parameter model can be scaled to 1-2B parameters if compute budget increases. The training pipeline (tokenizer, packer, optimizer configuration) remains identical -- only the model architecture config changes. This is a key advantage of building a robust pipeline.

Inference optimization: - **KV cache:** Store key-value pairs from previous tokens to avoid recomputation. Reduces inference from $O(T^2)$ to $O(T)$ per new token. - **Quantization:** INT8 quantization reduces model memory by 4x with < 1% perplexity degradation, enabling deployment on smaller GPUs. - **Speculative decoding:** Use a small 50M draft model to propose tokens, verified by the 500M model in parallel. Achieves 2-3x speedup.

Monitoring in Production

Metric	Threshold	Action
Inference latency (p99)	> 3.5s	Scale GPU instances

Metric	Threshold	Action
Report rejection rate	> 15%	Retrain or fine-tune
Unknown token rate	> 0.5%	Retrain tokenizer
Gradient norm (training)	> 10x mean	Investigate data quality
Validation perplexity	> 15.0	Early stop, investigate

Cost Analysis

Phase	Cost	Duration
Tokenizer training	\$200 (1x A100, 2 hours)	2 hours
Model training (3 epochs)	\$11,000 (4x A100, 68 hours)	68 hours
Evaluation	\$300 (1x A100, 4 hours)	4 hours
Total	**\$11,500**	**~74 hours**

The \$500 buffer under the \$12,000 budget accounts for hyperparameter tuning runs (2-3 short experiments on smaller data subsets to validate learning rate and batch size choices before the full training run).

Ethical Considerations

- All training data is de-identified per HIPAA Safe Harbor guidelines
- The model generates **draft** reports only -- a licensed radiologist must review and approve every report
- Confidence scores flag reports where the model is uncertain, routing them for immediate human review
- Regular bias audits ensure the model performs equitably across patient demographics and hospital types