ChatGPT

Fine-Tuned LLM/SLM Use Cases in TrackML-Backend

Adaptive Experiment Advisor (Hyperparameters & Architecture)

- **Feature:** An intelligent advisor that reads past experiment metadata (configs, architecture, results) and suggests next experiments. It can propose hyperparameter adjustments, architectural tweaks, or data augmentation strategies tailored to the project goals. For example, given a recent model that plateaued, it might recommend lowering the learning rate schedule or adding regularization.
- **Why LLM/SLM:** A fine-tuned model can **interpret rich experiment context** and articulate suggestions in natural language. By training on internal experiment logs and outcomes, the model learns which configurations tended to succeed. Research shows that LLMs can even match or outperform traditional hyperparameter tuning when given dataset/model descriptions [1]. Fine-tuning on domain data (i.e. past runs) makes the suggestions more accurate than a generic model.
- **Integration:** The advisor ingests TrackML's database of model configs, metrics, and notes. It forms prompts like "A CNN with LR=0.01 hit 85% accuracy on dataset X; what should we try next?" and returns prioritized experiment plans. It leverages TrackML version history to see what's already been tried, ensuring suggestions are novel.
- **Example Interaction:**

  > **User:** "Our last run (ResNet-50, LR=0.001) stalled at 75% accuracy. Any ideas?"
  > **Assistant:** "This suggests underfitting. Try increasing model capacity (add more layers or units) or reduce regularization. For example, you could raise learning rate to 0.002 and add a second dense layer. In past runs, similar tweaks improved accuracy by ~5%. Also consider augmenting the data by horizontal flips."

Failure Mode Diagnoser (Root-Cause Troubleshooter)

- **Feature:** A diagnostics tool that analyzes failed experiments and logs to **identify root causes**. It classifies failures (e.g. overfitting, gradient explosion, data issues) and suggests fixes. For instance, it parses error messages (e.g. CUDA OOM, NaN losses) and compares metric patterns against known failure signatures.
- **Why LLM/SLM:** Logs are essentially text; NLP can automatically categorize and interpret them. Modern LLMs excel at extracting meaning from error logs and correlating patterns. As one source notes, "logs can be seen as textual data" and NLP can group them to highlight issues [2]. A specialized LLM can be fine-tuned on the team's historic failures to better recognize project-specific errors.
- **Integration:** After a run fails, TrackML passes its logs and metrics to the model. The LLM reviews stack traces, loss curves, and any error comments. It then outputs a human-readable diagnosis and action list. This tightly uses TrackML's error and log metadata. It can also suggest whether retrying with more hardware or code fixes are needed.

- **Example Output:** "**Issue Detected:** Training loss became NaN at epoch 3, likely due to exploding gradients. **Likely Cause:** Learning rate is too high for this model. **Recommendation:** Decrease the learning rate (e.g., from 0.01 to 0.002) or use gradient clipping. Past runs with similar symptoms were fixed by using `Adam` instead of `SGD` or by adding BatchNorm layers."

Automated Experiment Documentation

- **Feature:** A **report generator** that turns raw experiment data into polished documentation. Given a completed run, it writes a summary describing the model, dataset, hyperparameters, results, and key observations. It can format reports in Markdown or PDF, creating experiment logs, summaries for wiki pages, or README updates automatically.
- **Why LLM/SLM:** LLMs are highly adept at summarization and writing technical descriptions. In similar contexts, LLMs have been used to auto-generate documentation from schemas (e.g. Databricks used LLMs to write table and column docs [3]). Fine-tuning on the team's style (e.g. commit messages, reports) ensures the language matches internal standards. A smaller specialized model (SLM) could generate docs more cheaply once fine-tuned for this one task [4].
- **Integration:** When an experiment completes, TrackML feeds all metadata (config files, version diffs, metrics) to the LLM. The model produces a narrative: explaining what was tried and what the outcomes were. This summary is attached to the experiment entry or emailed to stakeholders. It can draw on comments or notes fields stored in TrackML.
- **Example Output:** "**Run #17 Summary:** Trained a 3-layer CNN on Dataset A (data version 2025-06-10) with learning rate 0.005 and batch size 32. Achieved 89.5% accuracy (vs. previous 85%). Observed validation accuracy improved steadily, indicating no overfitting. Key changes from prior run: added BatchNorm and increased epochs from 20→30. **Conclusion:** Architecture change yielded +4.5% gain. Next step: try LR=0.01 or data augmentation to push above 90%."

Conversational Project Assistant (ChatOps Interface)

- **Feature:** A chatbot interface (Slack/Teams/CLI) for asking natural-language questions about the TrackML projects. Users can query experiment histories, compare models, retrieve metric trends, or get recommendations. The bot understands project-specific terms (dataset names, model IDs) and responds conversationally.
- **Why LLM/SLM:** A fine-tuned LLM can map user queries to TrackML data retrieval. Unlike a rigid search, it can handle ambiguous or incomplete queries. For example, LLMs have been shown to convert plain-language requests into precise actions (even in log query contexts [5]). Training on internal documentation and metadata ensures the bot uses the correct vocabulary (e.g. "Experiment 42", "Prod-Model v3"). Using an SLM can make the system lightweight and fast for simple queries, while still grounded in domain data [6].
- **Integration:** The bot leverages TrackML's API. When a question comes in, it either queries TrackML directly (retrieval) or uses RAG (prompting with relevant data). Its training includes project docs and labels so it knows, for example, that "latest accuracy" refers to a specific metric field. Access control ensures it only reveals allowed info.
- **Example Interaction:**
- **User:** "What was the best validation loss in project X this month?"
- **Bot:** "The lowest validation loss was 0.254 in Run #345 (ResNet-18, LR=0.002) on 2025-07-02 [7]."
- **User:** "How do Run 30 and Run 31 compare?"

- **Bot:** “Run 30 (ResNet-34, dropout=0.3) had test accuracy 91.2%, Run 31 (ResNet-34, dropout=0.5) had 89.7%. Both used same data; the higher dropout in Run 31 likely caused the lower accuracy.”

Anomaly and Insight Notifier

- **Feature:** A monitoring assistant that scans incoming experiment logs and metrics for **anomalies or notable patterns**, and generates alerts with explanations. For instance, it might notice a sudden drop in accuracy or an unexpected validation spike, and flag it with context (e.g. “accuracy dropped after epoch 5, a new data augmentation may be at fault”).
- **Why LLM/SLM:** LLMs can contextualize numeric anomalies and provide narrative insights. Traditional threshold alerts often spam teams with false positives, but an LLM can filter by looking at historical context and natural-language clues. As noted, ML can “automatically identify issues, even if there’s a huge number of logs” [8] . Fine-tuning on the project’s metric patterns (e.g. what is “normal” fluctuation) allows the model to focus only on significant outliers.
- **Integration:** The system ingests streaming training metrics from TrackML. When a metric deviates (e.g. sudden loss increase, flatlining accuracy), the LLM reviews recent runs and outputs a rationale. Alerts are posted to a monitoring dashboard or chat. Over time it learns what anomalies were actionable (reducing false alarms).
- **Example Output:** “**Alert:** Validation accuracy plateaued at 70% (baseline was ~85%). **Analysis:** This is unusual – in similar runs, accuracy continued rising. Check if the learning rate changed or if training data was inadvertently pruned. In the last commit, data normalization was altered; that may be the cause.”

Experiment Variation Recommender

- **Feature:** A creative assistant that **generates new experiment ideas** by varying existing ones. Given a baseline experiment, it can suggest permutations of hyperparameters or pipeline changes to explore. For example, it might propose “Run the same model with a larger batch size and learning rate decay” or “Add Gaussian noise augmentation and monitor its effect”.
- **Why LLM/SLM:** Fine-tuned on the team’s past experiments, the LLM learns which variations were fruitful. It can then recombine parameters in novel ways and even rank them by expected impact. This goes beyond rigid grid-search: the model can intelligently propose non-obvious tweaks, drawing from textual knowledge embedded in comments or literature. Small specialized models can handle this efficiently once trained on internal experiment logs and notes.
- **Integration:** After a run completes, users can click “Suggest variations”. The LLM reviews the run’s config and results, then outputs a list of 2–5 new experiments. It uses TrackML’s history to avoid duplicating attempts. This feature ties into the experiment planning phase of the lifecycle.
- **Example Output:** “Based on Run #200 (LR=0.01, dropout=0.3, no augmentation): Try one run with dropout increased to 0.5 and introduce random crops. Try another with a cosine LR schedule (start=0.01 → end=0.0001) and include horizontal flips. Historically, these changes improved performance by ~2-3% on similar models.”

Code and Config Snippet Generator

- **Feature:** An assistant that generates code or configuration templates for common tasks. For instance, a user could request “PyTorch training loop for classification with my dataloader” or

"Kubernetes config for deploying model X", and the system produces a ready-to-use snippet, customized to the project's style and libraries.
- **Why LLM/SLM:** LLMs are powerful at code generation from natural descriptions [9]. By fine-tuning on the team's actual codebase or config files, the model learns the project's idioms and best practices. A smaller LLM (SLM) can be used here as it needs to focus on domain-specific APIs and boilerplate, reducing compute while providing domain accuracy [6].
- **Integration:** The user interface lets engineers describe what they need (either via a prompt box or chat). The LLM pulls in context (e.g. the model class names, dataset fields from TrackML metadata) to tailor the snippet. It can insert the correct import statements and parameters from the project's environment.
- **Example Interaction:**

> **User:** "Write a PyTorch training loop for `ModelA` on DataLoader `train_loader` with Adam optimizer."
> **Assistant:** (Outputs code block)

```
model = ModelA().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        inputs, labels = batch
        optimizer.zero_grad()
        outputs = model(inputs.to(device))
        loss = criterion(outputs, labels.to(device))
        loss.backward()
        optimizer.step()
```

Model Lifecycle & Update Advisor

- **Feature:** A lifecycle manager that monitors deployed models and suggests when updates or retraining are needed. It uses model metadata (deployment date, version history, performance drift) to proactively plan maintenance. For example, if incoming data distribution shifts, it might warn that "Model v1.4 (deployed 3 months ago) is likely stale given new data stats". It can also draft version-release notes automatically.
- **Why LLM/SLM:** Managing lifecycle involves synthesizing diverse information (logs, code changes, metrics) into human-readable advice. An LLM can do this by learning patterns of drift and obsolescence. It can also translate version diff logs into summaries for stakeholders. A tailored SLM can run on smaller scale infrastructure (even on-device) to quickly evaluate if a model's metrics warrant retraining.
- **Integration:** The feature watches performance metrics from production (also tracked in TrackML) and the model registry. When certain thresholds or time limits are crossed, it triggers the LLM. The assistant then outputs a recommendation (e.g. "Schedule model retraining", "Rollback to previous stable version"). It logs comments in TrackML and notifies the MLOps team.
- **Example Output:** "**Lifecycle Note:** Model *Classifier-v1.2* (trained on data up to 2024-12) shows a 7% drop in accuracy on recent incoming data. Its training dataset is now outdated. **Suggested Action:**

Retrain with the latest data pipeline; consider adding synthetic data augmentation (as we did for v1.0). Also, prepare a changelog highlighting these updates for the model registry."

Team Knowledge & Collaboration Bot

- **Feature:** A collaborative AI assistant that helps the team share knowledge. It can summarize team discussions (from comments, pull requests, or meeting notes), tag experiments with relevant keywords, and maintain an FAQ. For example, it might notice repeated questions ("What does loss_spike mean?") and create a glossary entry. It can also write or update documentation pages based on common themes in the TrackML logs and comments.
- **Why LLM/SLM:** Fine-tuned on the project's communication (issue trackers, chat logs) the model can learn team lingo and recurring issues. It can distill lengthy discussions into concise notes. Since this is an internal and specialized task, a compact SLM suffices and is cost-effective. Domain finetuning ensures that when it auto-tags an experiment or paraphrases a discussion, it uses the correct technical terms.
- **Integration:** The bot integrates with TrackML's comments and version comments. Periodically, it generates a "Team Brief": a summary of recent experiments, merged PRs, and open issues. It can answer internal FAQs like "Who was responsible for dataset cleaning last month?" by searching tracked comments. It also suggests linking relevant experiments when a new run is created (e.g. "Link this run to baseline experiment #10 which used similar config").
- **Example Output:** "**Weekly Team Update:** This week the team ran 12 experiments. Highlights: Run #112 achieved 95% accuracy after adding MixUp (baseline was 90%). Run #115 stalled at epoch 2 due to a missing file (fixed by adjusting data path). Open item: define naming convention for model artifacts (several PRs had inconsistent tags). Next steps: plan a data review to address the *class imbalance* noted in Runs #110–112."

Sources: These use-case concepts are inspired by current trends in AI-augmented MLOps and generative AI for documentation and code. For instance, LLMs have been shown to suggest hyperparameters from model descriptions [1], convert natural language into precise log queries [5], and automatically write technical documentation [3]. Small, specialized LLMs (SLMs) can be fine-tuned quickly and cheaply for domain-specific tasks [4] [6], making them ideal for integrating deep learning assistants into an ML tracking backend.

---

[1] [7] [2312.04528] Using Large Language Models for Hyperparameter Optimization
https://arxiv.org/abs/2312.04528

[2] [8] A Machine Learning Approach to Log Analytics: How to Analyze Logs?
https://neptune.ai/blog/machine-learning-approach-to-log-analytics

[3] [4] Bespoke LLM for AI-Generated Documentation | Databricks Blog
https://www.databricks.com/blog/creating-bespoke-llm-ai-generated-documentation

[5] Using LLM for Cybersecurity Log Analysis: A Complete Guide - Tolu Michael
https://tolumichael.com/using-llm-for-cybersecurity-log-analysis-a-complete-guide/

[6] SLMs vs LLMs: What are small language models?
https://www.redhat.com/en/topics/ai/llm-vs-slm

9 [2406.00515] A Survey on Large Language Models for Code Generation
https://arxiv.org/abs/2406.00515