M. Tech Thesis Proposal

Title of the Thesis

Automating ML/DL Paper-to-Code Translation via Multi-Agent LLM Pipelines

---

Abstract

Reproducibility in machine learning and deep learning research is hindered by the frequent absence of runnable code alongside published papers. Manually re-implementing algorithms from text, pseudocode, and diagrams is labor-intensive and error-prone, slowing innovation. This thesis proposes a **multi-agent pipeline** powered by large language models (LLMs) and vision-enabled parsing to automatically translate arbitrary ML/DL research papers into fully runnable code repositories. The system employs specialized agents for paper analysis, algorithm interpretation, API/library mapping, code integration, verification, and iterative debugging. By closing the loop from "paper → code → validated results," the framework aims to drastically reduce reproduction time, improve research efficiency, and provide a foundation for automated R&D workflows.

---

1. Introduction

The accelerating volume of ML and DL publications brings a reproducibility crisis: up to 90% of papers lack accompanying code, forcing researchers to rebuild experiments from scratch. This manual process—deciphering notation, reconstructing architectures, guessing hyperparameters—can consume weeks per paper. Our thesis investigates an **automated, agentic AI system** that ingests a paper's text, figures, and pseudocode; decomposes its methods into structured tasks; and outputs a complete, runnable code repository (including training scripts, model definitions, and evaluation harnesses).

---

2. Problem Statement

- **Low reproducibility:** Most ML/DL papers omit full implementations.
- **Manual overhead:** Researchers spend excessive time on re-implementation rather than novel development.
- **Innovation bottleneck:** Slow, error-prone reproduction impedes both academic progress and industrial R&D.

Existing tools either link to existing repositories (e.g., CatalyzeX) or handle narrow subdomains (e.g., DLPaper2Code for neural-network diagrams). A generalized, end-to-end solution is missing.

---

3. Proposed Solution

We propose a **modular multi-agent pipeline** with the following components:

a. Paper Analysis & Parsing

- Combines OCR/vision models (for diagrams and tables) with LLMs to extract model descriptions, pseudocode, and experiment settings.
- Produces a structured intermediate representation (e.g., JSON schema of modules, parameters).

b. Algorithm Interpretation Agent

- Translates mathematical notation and algorithmic steps into explicit computational workflows.
- Handles custom losses, iterative procedures, and control-flow logic extracted from the paper.

c. API/Library Mapping

- Maps each identified component to appropriate open-source frameworks (scikit-learn, PyTorch, TensorFlow, OpenAI Gym, etc.).
- Retrieves code snippets or templates from documentation when needed.

d. Code Integration & Generation

- Assembles modules into a coherent codebase: data loaders, model definitions, training/evaluation scripts, and dependency manifests (e.g., requirements.txt).
- Structures the repository for immediate execution (e.g., train.py, config files).

e. Verification Agent

- Executes experiments on benchmark datasets or synthetic inputs.
- Compares metrics (accuracy, loss curves, tables) against those reported in the paper and flags discrepancies beyond tolerance thresholds.

f. Debugging & Refinement Agent

- Diagnoses failures or metric mismatches (e.g., missing hyperparameters, implementation bugs).
- Iteratively refines code or parameters, looping back through integration and re-verification until results align.

A **top-level Planner Agent** orchestrates these stages, manages context memory, and resolves inter-agent conflicts.

---

4. System Architecture

1. **Paper Ingestion Layer**
    - Downloads paper PDF; runs OCR/vision modules on figures and tables.
    - Feeds raw text and images into the Paper Analysis Agent.
2. **NLP/Vision Engine**
    - LLM prompts for text understanding; vision models parse architecture diagrams.
    - Outputs a structured description of algorithms, model components, and experiment protocols.
3. **Agentic Workflow Layer**
    - Hosts the six specialized agents (Analysis → Interpretation → Mapping → Integration → Verification → Debugging).
    - Agents communicate via shared intermediate representations stored in a lightweight database.
4. **Execution & Feedback Loop**
    - Automated test harness runs generated code; Verification Agent logs results.
    - Debugging Agent receives logs, issues targeted refinement prompts back to Integration.
5. **Repository Deployment**
    - Once validated, the system packages the code into a Git repository with CI configuration for continuous validation.

---

5. Benefits

- **Accelerated Reproducibility:** Reduces reproduction time from weeks to hours.
- **Error Reduction:** Minimizes human coding errors by leveraging LLM expertise and structured testing.
- **Innovation Boost:** Frees researchers to focus on new ideas rather than re-implementing prior work.
- **Seamless R&D Integration:** Enables automatic Git/CI pipeline generation for continuous validation of newly published papers.

---

6. Use Case Scenario

A researcher selects a newly published transformer-based NLP paper. The system:

1. Parses the architecture diagram and pseudocode.
2. Maps transformer layers and attention mechanisms to PyTorch modules.
3. Generates train_transformer.py, data preprocessing scripts, and a requirements.txt.
4. Runs training on a sample dataset, compares perplexity and accuracy to the paper's benchmarks (±2% tolerance).
5. Detects a mismatch in learning-rate scheduling, automatically corrects the scheduler code, and re-executes until metrics align.
6. Publishes a ready-to-use GitHub repository with CI that re-runs tests on every commit.

---

7. Challenges and Limitations

- **Textual Ambiguity:** Papers often omit hyperparameters or implementation details.
- **Diagram Parsing Errors:** Complex figures may be misinterpreted by vision models.
- **Math Translation:** Converting advanced equations into code can require multiple refinement loops.
- **Verification Noise:** Random seeds and hardware variation can introduce metric fluctuations.

- **Domain Nuances:** Adapting to diverse subfields (CV, NLP, RL, statistics) may require specialized sub-agents.

---

8. Future Work

- **Extending to Other Disciplines:** Apply the pipeline to optimization, graphics, or scientific computing papers.
- **Interactive Refinement:** Incorporate a human-in-the-loop feedback interface for challenging cases.
- **CI/CD Integration:** Deepen integration with enterprise CI systems for live tracking of incoming papers.
- **Adaptive Hyperparameter Tuning:** Embed AutoML agents to automatically search hyperparameters during verification.
- **Open Benchmark Release:** Curate and publish a larger corpus of ML/DL papers with ground-truth implementations.

---

9. Conclusion

This thesis will demonstrate that a carefully orchestrated multi-agent LLM pipeline can **automatically translate** ML and DL research papers into **validated, executable code**, addressing the reproducibility crisis and catalyzing innovation. By integrating NLP, vision parsing, structured agent workflows, and automated verification loops, the proposed system stands to transform how the AI community shares, verifies, and builds upon scientific knowledge.