Statement of Purpose (SoP)
DSL501: Machine Learning Project

Team Name: **Singularity**

September 7, 2025

1. Project Details

- **Project Title:** Deep Learning-Based Smart Contract Vulnerability Detection using Optimized-CodeBERT
- **Code Repo Link :** To be made
- **If Own Idea:** [No]

2. Problem Statement

- **Context and motivation:** Smart contracts power decentralized applications but are immutable once deployed, making vulnerabilities costly and irreversible. Past exploits such as the DAO and BEC token attacks caused millions in losses, highlighting the need for reliable vulnerability detection. Static tools like Slither and Mythril rely on predefined rules, often yielding false positives/negatives.
- **Importance in ML domain:** Detecting vulnerabilities can be modeled as a natural language processing task. Deep learning models, especially pre-trained code models like CodeBERT, can learn semantic and syntactic patterns of vulnerabilities, offering improved adaptability and accuracy.
- **Scope of work:** This project implements and extends the *Lightning Cat* framework using Optimized-CodeBERT, LSTM, and CNN models. It focuses on four key vulnerabilities—Reentrancy, Timestamp Dependency, Unhandled Exceptions, and `tx.origin` misuse— evaluated on the SolidiFI-benchmark dataset, with performance compared against traditional static tools.

3. Methodology

- **Proposed ML models/architectures:** Three deep learning models will be implemented and compared—Optimized-CodeBERT (fine-tuned transformer), Optimized-LSTM (sequential model for temporal code patterns), and Optimized-CNN (convolutional model for structural features).
- **Key techniques, algorithms, or frameworks:** Vulnerability-focused function code will be extracted and preprocessed using the CodeBERT tokenizer, with sequences truncated/padded to 510 tokens. CodeBERT embeddings will provide semantic representations. Models will be trained in PyTorch with AdamW optimization, grid search hyperparameter tuning, and dropout regularization. Evaluation will be conducted on the SolidiFI-benchmark dataset using Accuracy, Precision, Recall, and F1-score.
- **How the approach differs from existing methods:** Unlike static tools (e.g., Slither, Mythril) that rely on predefined rules, this approach uses deep learning to automatically learn vulnerability patterns. It also improves upon prior ML methods by leveraging CodeBERT embeddings and focusing on critical vulnerable functions, reducing noise and handling long contract texts effectively.

- **Components to be built:** The implementation will include data preprocessing scripts (contract cleaning, function extraction, tokenization), training pipelines for each model, hyperparameter tuning modules, and evaluation/benchmarking scripts to compare deep learning models against static analyzers.

4. Dataset Details

- **Name and source:** The project will use the *SolidiFI-benchmark* dataset for evaluation, alongside training data from three main sources: (a) Slither Audited Smart Contracts Dataset, (b) SmartBugs-Wild repository, and (c) 1,000 expert-audited contracts.
- **Size and structure:** The combined training dataset contains approximately 31,909 vulnerable code snippets covering four major vulnerability types: Reentrancy, Timestamp Dependency, Unhandled Exceptions, and `tx.origin` misuse. The SolidiFI-benchmark test set includes 9,369 injected vulnerabilities, ensuring standardized comparison with other tools.
- **Preprocessing:** Contracts are parsed to extract only vulnerable functions, which are then tokenized using the CodeBERT tokenizer. Sequences exceeding 510 tokens are truncated, and shorter ones are zero-padded. Comments and irrelevant characters are removed to reduce noise and improve convergence.
- **Why suitable:** These datasets are widely used benchmarks in smart contract analysis, cover realistic vulnerabilities, and allow fair comparison with traditional tools like Slither, Mythril, and SmartCheck. Using function-level vulnerable code ensures better feature extraction and improved model performance.

5. Required Resources

- **Hardware:** GPU-enabled machine (e.g., NVIDIA Tesla V100), multi-core CPU, 32 GB RAM, ample storage.
- **Software:** Python 3.x, PyTorch, Scikit-learn, Gensim (Word2Vec/FastText), Solidity compiler, Slither.
- **Other tools:** GitHub for version control, JSON/text parsers, preprocessing scripts from ScrawlD.

6. Novelty of Approach

Highlight the unique contributions of your work:

- **Explainable Vulnerability Detection:** Extend the Lightning Cat framework by incorporating attention-based visualization. During inference, CodeBERT's self-attention weights will be extracted and mapped back to Solidity tokens, highlighting the lines of code most responsible for a vulnerability classification.
- **Benefit:** This approach transforms the system from a black-box detector into a developer-friendly assistant. Instead of providing only a binary prediction, the model will also show *why* a function is vulnerable, thereby aiding in debugging and increasing trust in ML-driven vulnerability detection.
- **Novelty:** While prior work (e.g., Lightning Cat) demonstrated the effectiveness of CodeBERT for vulnerability detection, it did not address interpretability. This project introduces an explainability layer that bridges machine learning outputs with human-auditable insights, making the framework more practical for real-world smart contract auditing.

7. Team Composition and Individual Contributions

- **Member 1:** *Vasu Garg*, Roll No.: *12342330*, Contribution: *Complete Project*.

8. Expected Outcomes

List the deliverables and goals you plan to achieve by the end of the project:

- Achieve high performance on the SolidiFI-benchmark dataset with Accuracy >95%, Precision >95%, Recall >92%, and F1-score around 93% for the four target vulnerabilities (Reentrancy, Timestamp Dependency, Unhandled Exceptions, and `tx.origin` misuse).
- Develop and release trained deep learning models (Optimized-CodeBERT, Optimized-LSTM, and Optimized-CNN) for smart contract vulnerability detection.
- Provide explainable outputs using attention-based visualization to highlight vulnerable code segments, improving interpretability and developer trust.
- Build a complete pipeline including data preprocessing, model training, hyperparameter tuning, and benchmarking against static analyzers such as Slither and Mythril.
- Deliverables will include:
  - A reproducible codebase with documentation.
  - Trained models and evaluation reports.
  - A comprehensive project report comparing results with prior tools and methods.

9. References

1. Tang, X.; Du, Y.; Lai, A.; Zhang, Z.; Shi, L. Deep learning-based solution for smart contract vulnerabilities detection. *Scientific Reports*, 13, 20106 (2023). `https://www.nature.com/articles/s41598-023-47219-0`

2. Deng, W.; Wei, H.; Huang, T.; Cao, C.; Peng, Y.; Hu, X. Smart Contract Vulnerability Detection Based on Deep Learning and Multimodal Decision Fusion. *Sensors* 2023, 23(16), 7246. `https://www.mdpi.com/1424-8220/23/16/7246`

3. Wu, J.; Dai, H.N.; Wang, H.; Wang, W.; Imran, M. Machine Learning for Blockchain Data Analysis: Progress and Opportunities. *ACM Computing Surveys*, 2024. `https://dl.acm.org/doi/abs/10.1145/3728474`

4. Fu, Q.; Feng, C.; Xu, C.; Chen, X.; Zhuang, W. SoliAudit: Smart Contract Vulnerability Assessment Based on Machine Learning and Fuzz Testing. *IEEE Access*, 2020. `https://ieeexplore.ieee.org/abstract/document/8939256`