Predicting Compiler Speedup via Fine-Tuned CodeRankEmbed + Autoencoder

Javier Sin Pelayo (Author)
Senior Computer Science Student
javier.sinpelayo@uri.edu

Marco Álvarez (Tutor)
Department of Computer Science and Statistics
Associate Professor
malvarez@uri.edu

Christian Esteves (Co-Tutor)
Department of Computer Science and Statistics
Lecturer (temp)
cesteves@uri.edu

University of Rhode Island

CSC 561

International Engineering Program

Outline

1. Motivation & Challenges
2. Dataset & Task
3. Model Architecture
4. Training & Loss
5. Experiments & Results
6. Conclusions & Future Work

Motivation & Problem

- Compilers apply many loop transformations (unrolling, tiling, distribution) → huge search space
- Equivalent code variants can perform very differently
- Manual / brute-force search is too slow and expensive

Key Challenges & Solutions

- **Semantic embeddings:** fine-tune CodeRankEmbed via LoRA
- **Imbalanced data:** 80% slowdowns, 20% speedups → focal loss
- **Efficiency:** autoencoder regularizer + MLP head

Dataset & Task

- ~2,100 unique C loop nests → ~70 k samples
- 56-dim multi-hot vector of applied flags
- Label: slowdown (≤1×) vs. speedup (>1×)
- Stratified 80/10/10 split by loop group

Model Pipeline (Part 1)

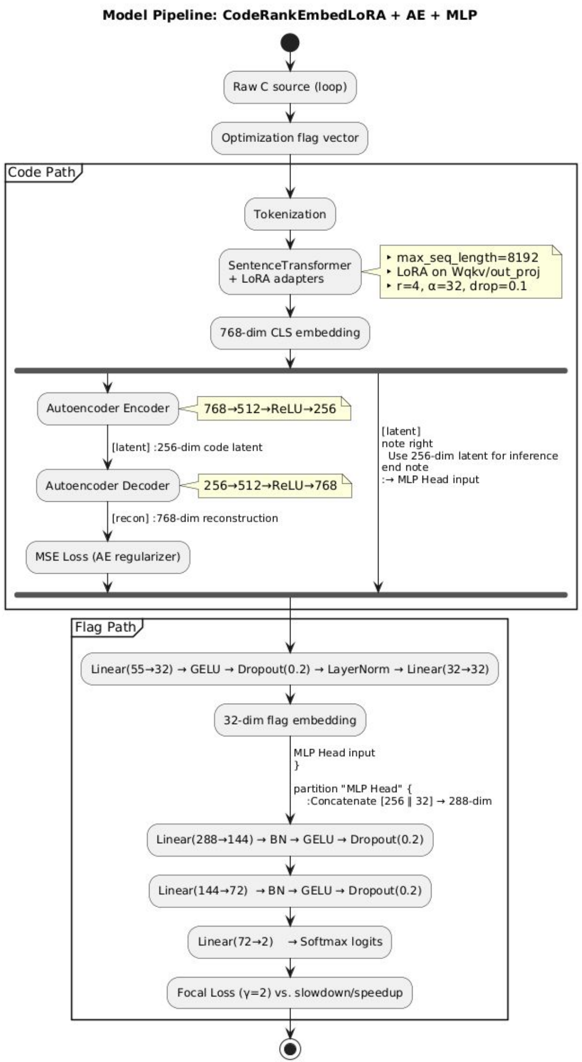
- Code Encoder
  - nomic-ai/CodeRankEmbed (BERT-style, CLS → 768)
  - LoRA on Wqkv & out_proj (r=4, α=32, drop=0.1)
- Autoencoder
  - Encoder: 768 → 512 → BN → ReLU → 256
  - Decoder: 256 → 512 → BN → ReLU → 768
  - MSE reconstruction loss

Model Pipeline (Part 2)

- Flag Projection
  - 56 → 32 linear → GELU → Dropout(0.2) → LayerNorm → 32 → 32 linear
- Classification Head
  - [256 // 32] → 288 D → MLP: 288→144→72→2 → softmax

Modeling Pipeline (Diagram)

1. C source → CodeRankEmbed + LoRA → → 768-d token → 256-d latent via AE
2. Flag vector → 32-d projection
3. Concatenate → MLP → 2-way softmax



Training & Loss

$$FL(p_t) = -(1 - pt)^{\gamma} log(p_t)$$

- 'p_t' is the model's predicted probability for the true class (after softmax)
- $\gamma \geq 0$ is the focusing parameter (in my case, $\gamma = 2$)

- Loss: two-class focal loss (γ=2, no α class weights)
- Optimizer: AdamW (lr=1e-3, wd=1e-3)
- Scheduler: ReduceLROnPlateau (factor=0.5, patience=3)
- Batch size: 128, Epochs: 20

Experimental Setup

- Filter loop-groups with +3 transforms → 1,586 groups
- Max token length ≤250
- No oversampling; relied on focal loss
- Hyperparam grid over
  - LR {1e-3,1e-4,1e-5}, WD {1e-3,1e-4,1e-5}
  - Opt {SGD,AdamW}, Sched {none,plateau}
  - Dropout {0.1…0.5}
  - LoRA $r \in \{2,4,8\}$, $\alpha \in \{16,32,64\}$, drop $\in \{0.1,0.2\}$

Results (Averages over 5 runs)

- **5 runs averages (± stddev)**
  - **Val Acc**: 0.8846 ± 0.0444
  - **Speedup** F1: 0.7160 ± 0.0326
    - Precision: 0.7360 ± 0.0287
    - Recall: 0.6960 ± 0.0393
  - Slowdown F1: 0.9260 ± 0.0350
    - Precision: 0.9180 ± 0.0402
    - Recall: 0.9320 ± 0.0325

Comparisons

- vs non-LoRA baseline (*Christian Esteves's approach*) :
  - + 9 pts on SU-Recall
  - + 3 pts on SU-Precision
  - + 7 pts on SU-F1
- vs original paper (*Learning to Make Compiler Optimizations More Effective*):
  - + 15 pts on SU-Recall
  - + 7 pts on SU-Precision
  - + 11 pts on SU-F1

Conclusions & Future Work

- Takeaways
  - LoRA-tuned CodeRankEmbed + AE + focal loss excels on skewed data
  - Achieved ~88.5 % val accuracy, speedup-F1≈0.72
- Next Steps
  - Multi-class buckets (add “Neutral” + finer speedup ranges)
  - Test other LLM backbones (CodeT5+, DeepSeek)

References

[1] nomic-ai. (2023). CodeRankEmbed: A Transformer-Based Code Embedding.

Retrieved May 2, 2025, from https://huggingface.co/nomic-ai/CodeRankEmbed

[2] Hugging Face. (2023). PEFT: Parameter-Efficient Fine-Tuning Library.

Retrieved May 2, 2025, from https://github.com/huggingface/peft

[3] LORE: A Loop Repository for the Evaluation of Compilers

Chen, Z., Gong, Z., Szaday, J. J., Wong, D. C., Padua, D., Nicolau, A., Veidenbaum, A. V., Watkinson, N., Sura, Z., Maleki, S., Torrellas, J., & DeJong, G. (2017). LORE: A loop repository for the evaluation of compilers. In Proceedings of the IEEE International Symposium on Workload Characterization (IISWC) (pp. 219–228).

[4] Learning to Make Compiler Optimizations More Effective

Mammadli, R., Selakovic, M., Wolf, F., & Pradel, M. (2021). Learning to make compiler optimizations more effective. In Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming (pp. 9–20).

Acknowledgments

- **Dr. Marco Álvarez**
  – For inviting me into this project, providing mentorship, and offering continuous guidance and resources throughout my research journey.

- **Christian Esteves**
  – For sharing his deep expertise in compiler optimization, quickly bringing me up to speed on the problem context, and patiently answering my technical questions.

Thanks for staying in the loop!