Distilling CodeT5 for Efficient On-Device Test-Assertion Generation

Author: Andrei Vlad Nicula Responsible professor: Mitchell Olsthoorn Supervisor: Annibale Panichella

1. INTRODUCTION

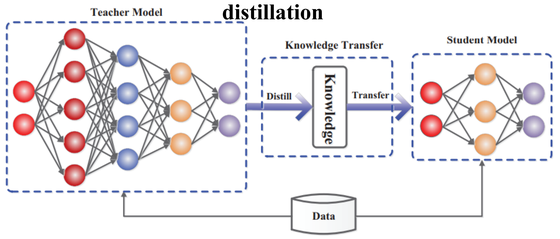
Background

- **Software testing** is essential for ensuring correctness, reliability, and preventing regressions in complex codebases.
- Crafting precise **test assertions** (e.g. *assertEquals, assertTrue, assertNotNull)* is challenging, labor-intensive, and prone to human error.
- Classical **Search-Based Software Testing (SBST)** tools automate test-input generation and scaffolding but treat programs as black boxes, generating only simple value-based checks that miss deeper semantic meaning.
- **Large Language Models (LLM)** fine-tuned on code like **CodeT5** excel at synthesizing semantically rich assertions but are impractical for on-device use due to high memory, latency, and computational requirements.

Distillation

- **Knowledge distillation** trains a compact ***student*** to mimic a larger ***teacher*** by learning from the teacher's output (probability distributions over possible tokens/outputs) alongside true labels.
- Students achieve high task performance with drastically reduced model size and inference latency, enabling on-device deployment and stronger privacy guarantees by not relying on an external providers (e.g. *OpenAI*).

The generic teacher-student framework for knowledge distillation



Research Question

To what extent can we distill CodeT5 into a compact student that retains reasonable performance of the teacher's assertion-generation accuracy while reducing RAM footprint and inference latency on CPU hardware?

2. METHODS

Dataset

A subset of 10K data points of the **Methods2Test** benchmark dataset containing: the *class under test*, the *focal method* (method for which the assertions are built), the *test method,* and the originals assertions written *Java* and *JUnit* code, augmented with the teacher model's *logits* (raw, unnormalized scores produced by the last layer of a neural network before these scores are transformed into probabilities) and predicted assertions.

Teacher

The teacher is **CodeT5-base -** an encoder–decoder Transformer specialized for code understanding and generation**.** It is pretrained on large-scale source code from multiple languages, giving it deep semantic knowledge for both *code* and *Natural Language*. The model is further **fined-tuned** on the assertion pairs in the dataset.

Student

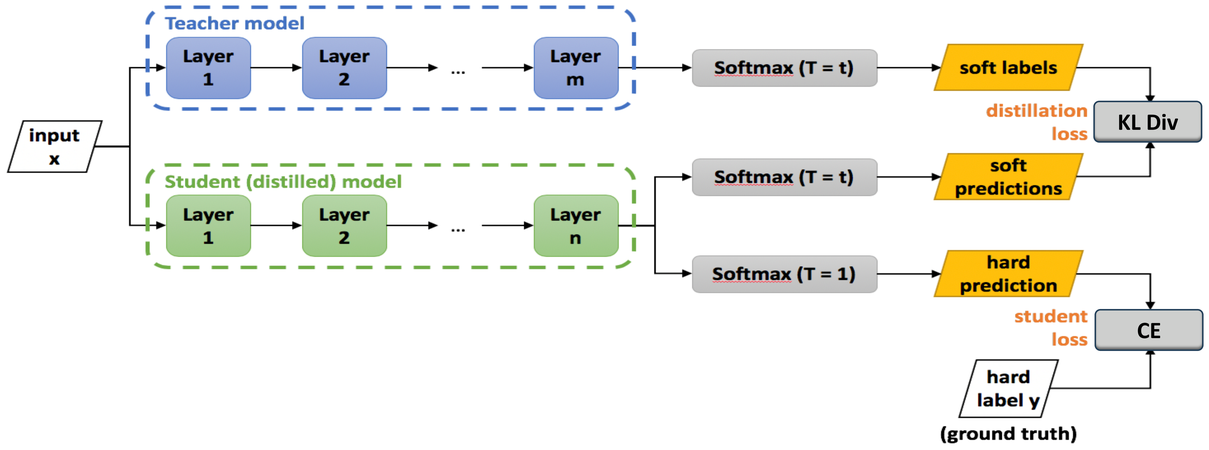
We derive several compact students from the original CodeT5 architecture by systemically reducing **hidden layer counts, feed-forward layer size, number of attention heads:**

- **CodeT5-small:** a smaller variant of the CodeT5-base that is pretrained in the same manner as the teacher
- **Custom Mini-Student**: smaller architecture with randomly initialized weights.

Pipeline

1. **Input Preparation:** Provide the student with the test method without the assertions plus surrounding class and focal-method code as context for better generation.
2. **Soft-Target Distillation:** Compute the **Kullback–Leibler** (KL) divergence (a way to measure how 2 probability distributions differ) between teacher and student output after **Softmax** (converts logits to probability distributions) function has been applied.
3. **Hard-Label Training:** Compute **cross-entropy** (CE) loss on the ground-truth assertion tokens, which measure how closely student's predicted probability distribution aligns the actual labels.
4. **Loss Balancing and Backpropagation:** Combine soft and hard losses via a weighted sum and update student parameters through backpropagation.

Illustration of the pipeline setup using a temperature T



3. RESULTS

Experiments

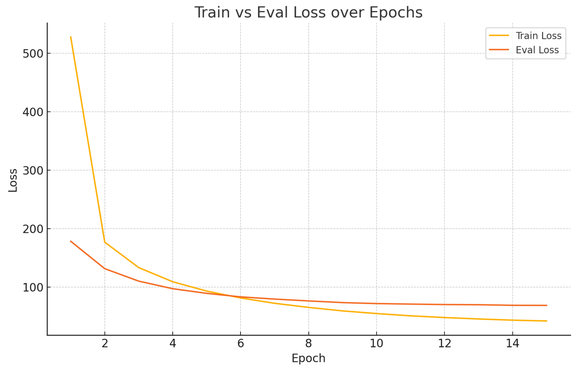
- **Distillation Loss:** tracking the convergence of the distillation over time while training.
- **Assertion Quality:** exact-match accuracy, precision, recall, similarity compared to the teacher and to the ground truth.
- **Pre-training vs custom model:** comparing the capably of the pre-trained CodeT5-small vs an even smaller Custom architecture that is randomly initialized
- **Inference Speed and Memory Footprint:** measure latency on CPU (single-threaded) and GPU across different devices and the memory footprint reduction compared to the teacher.

Findings

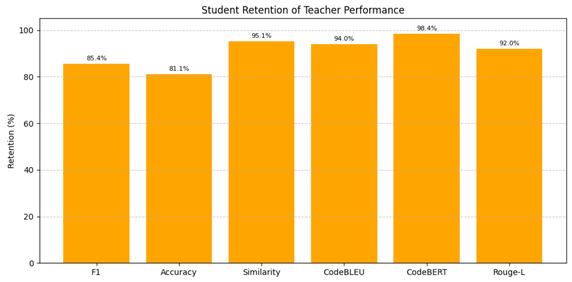
Memory Footprint:

- **CodeT5-base:** 900MB
- **CodeT5-small:** 230MB (75% reduction) – **Best student**
- **Custom Mini:** 60MB (90% reduction)

Loss Curves: For the **CodeT5-small** loss drops rapidly over 15 epochs, while validation loss steadily declines with no signs of overfitting.



Retention Chart: CodeT5-small preserves roughly about 80% of the teacher's accuracy on ground-truth metrics and above 90% of the similarity.



Custom Mini-Student: Trained for **62 epochs** (not converging): **accuracy** and **F1** are **3.6x lower** and **similarity** is **2x lower** when compared to the CodeT5-small.

Inference Speed Across devices: Presenting Mean, Standard Deviation and 95% confidence interval for each setup. The time is reported per test-method.

Table 1: Inference Latency Statistics by Configuration

Configuration	Mean (s)	SD (s)	95% CI (s)
GPU A100	0.6943	0.9270	±0.0575
GPU RTX 3070	1.7204	2.3926	±0.1483
CPU i7-1165G7	1.9086	2.1475	±0.1331
CPU 6800H	2.6557	3.2182	±0.1995

4.CONCLUSION

Discussion

Results indicate that our best CodeT5 student deliver **near-teacher assertion quality (≥ 80 %** retained accuracy on exact matches) with substantially lower resource demands **(<250 MB** memory) within 15 epochs, maintaining strong semantic fidelity across diverse tests, suitable for on-device and CI integration. While non-pretrained students require much more time (**>100** epochs) to fully converge.

Future Work

- **Context Window:** Explore configurations with larger context windows as well as techniques that allow efficient retrieval of related methods and classes to generate assertions in complex scenarios.
- **Scope:** Explore other languages and testing frameworks.
- **Training Set:** Extend the training set to the full size of the Methods2Test dataset.
- **Student Configuration Search:** Systematically search the possible configurations for the student model that optimize latency and accuracy the most (e.g. using genetic algorithms, grid search, etc..)
- **Human Study:** Implement a Human Study with Human Feedback Reinforcement Learning to improve training and to asses the actual accuracy of the assertion in real life scenarios.
- **Improve training infrastructure:** Better GPU setups will allow training Custom models for longer until converging.

5.ACKNOWLEDGEMENTS

Software Engineering Research Group at TU Delft especially professor Annibale Panichella and professor Mitchell Olsthoorn.

6.CONTACT

anicula@tudelft.nl