Closing The Gap

Java Test Assertion Generation via Knowledge Distillation with Trident Loss

Jeroen Chu
m.d.chu@student.tudelft.nl

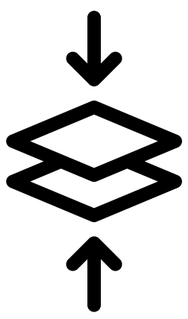
Supervisors:
Annibale Panichella, Mitchell Olsthoorn

TU Delft

1. Problem

Main Goal

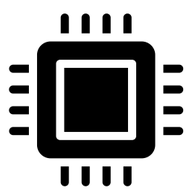
This work aims to overcome the high computational demands of Large Language Models (LLMs) by developing a distilled model for Java assertion generation, making assertion generation accessible to developers in resource-constrained environments.



Motivation

While large language models (LLMs) show impressive capabilities in generating test assertions, their practical deployment in real-world software development is hindered by several critical barriers:

- Computational Cost:
State-of-the-art models are computationally expensive and memory-intensive, limiting their use in common developer scenarios like local IDEs or CI/CD pipelines where rapid feedback is essential.



- Privacy Risks: Reliance on cloud-based APIs for large models introduces significant privacy concerns, as developers must send proprietary or sensitive code to third-party servers.

- Accessibility Barriers: The dependency on powerful commercial models creates a financial barrier to entry, excluding developers and researchers who lack the resources to access them, which in turn limits broader community innovation.

Research Questions (RQ)

- **RQ1**: How does our proposed Trident loss compare to traditional knowledge distillation methods?

- **RQ2**: How do the individual components of the Trident loss contribute to the final code quality?

- **RQ3**: How much does knowledge distillation reduce the computational footprint (model size and memory) of the assertion generation model?

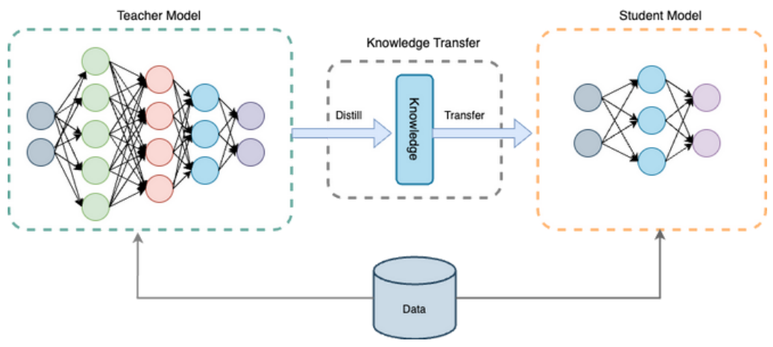
2. Context

Existing solutions

Automated test assertion generation has been a long-standing goal in software testing research. Early approaches often relied on dynamic analysis or inferring program invariants, but these proved difficult to integrate into real-world developer workflows.
More recently, large language models like AthenaTest have been trained to generate tests automatically. However, these models often exhibit a limited depth of code understanding, generating syntactically plausible code that fails to capture the essential logic of a valid assertion.

Knowledge Distillation

To address the high computational cost of these large models, Knowledge Distillation (KD)—a technique that trains a smaller "student" model to mimic a larger "teacher"—offers a path to efficiency. But traditional distillation methods are not optimized for the strict syntactic and semantic demands of code generation. They typically penalize a model for generating a valid assertion that is not an exact match, even though they are logically identical.



Towards smarter loss functions

Applying traditional KD to code is challenging because the method penalizes valid, semantically equivalent code that differs syntactically. For instance, the following are logically equivalent:

```
- assertFalse(list.isEmpty())
- assertTrue(list.size() > 0)
```

However, a basic loss function would treat it as an error. This highlights the need for a more nuanced approach that moves beyond simple token-matching to understand the underlying logic of the code. To solve this, a multi-component loss function is suited—one that can balance different learning objectives to capture syntactic correctness, semantic meaning, and agreement with the teacher model simultaneously.

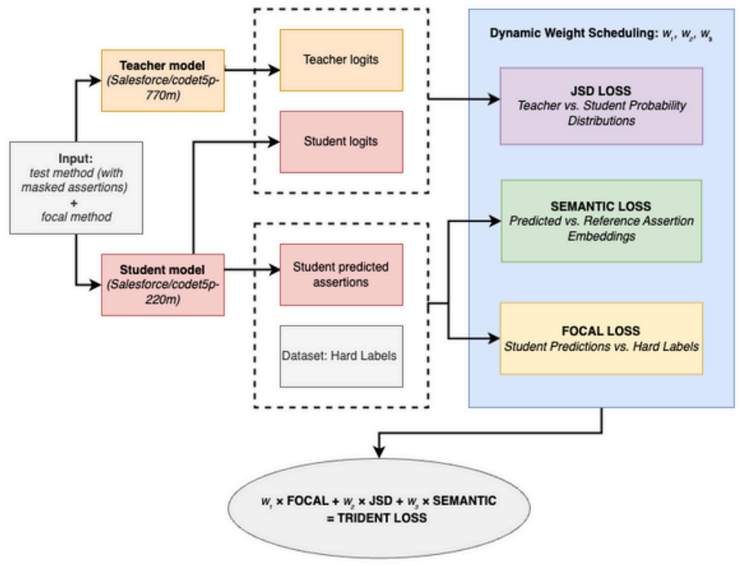
3. Solution

Methodology

We train our student model (CodeT5+ 220M) using knowledge from a larger teacher (CodeT5+ 770M). Our method is centered on the **Trident loss**, which dynamically adjusts the importance of three components as training progresses:
- **Focal Loss** for concentrating on difficult tokens.
- **Jensen-Shannon Divergence** for stable knowledge transfer.
- **Semantic Similarity Loss** for rewarding logically correct assertions.

This dynamic weighting allows the model to first learn the teacher's broad knowledge before focusing on the fine-grained semantic details of code.

KD with Trident loss overview:



Results

- Trident loss outperforms traditional KD methods
- KD with Trident loss successfully **reduced** the model size and required GPU memory by **71.4%**.
- The resulting student model **retains 90%** of the teacher's Code Quality Score, enabling deployment in resource-constrained environments.
- A key insight is that the static loss configuration was more effective, suggesting pre-trained models benefit from a consistent optimization pressure.