Distilling Knowledge for Assertion Generation: Alpha-Temperature Tuning in Smaller Language Models

Kristian Hristov[1]

Supervisor(s): Mitchell Olsthoorn[1]**, Annibale Panichella**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Name of the student: Kristian Hristov
Final project course: CSE3000 Research Project
Thesis committee: Mitchell Olsthoorn, Annibale Panichella, Petr Kellnhofer

June 22, 2025

Abstract

Testing of software is crucial to the quality of the final product - manual test assertion creation has become a significant bottleneck in the development process, which delays release. Having shown promise in generating assertions automatically, Large language models (LLMs) have showed promise in generating assertions automatically. This is due to their fluency in both natural languages and code, as well as the fact that they produce tests a lot faster than a developer would. However, LLMs must reckon with deployment issues that come with the high computation time and latency of large models, or the limited functionality of their smaller, locally-executable counterparts. Knowledge distillation, a technique that aims to "transfer knowledge" from a teacher model to a student one, can thus enable the potential of smaller and faster models. This drives the research to explore the effectiveness of knowledge distillation in developing a smaller and efficient model for assertion generation. With CodeT5 as the teacher model, the student model learns from the teacher. The student is iteratively trained in epochs, validated on unseen data.The metrics used to evaluate include assertion accuracy, similarity to teacher model output and ground truth, model size, inference time, with the goal to quantify the trade-offs and determine the feasibility of distilled models for practical assertion generation. We presented and analyzed the results we achieved. The capability the student showed was around 1/3 of that of the teacher, which suggest a potential for creating efficient, yet reliable assertion generation tools.

1 Introduction

Software testing is a critical process for ensuring software quality by evaluating and verifying that a product or application behaves as intended. Its goal is to identify defects, ensure requirements are met, and improve overall reliability before release. This is done with the help of "test cases" - different scenarios in which a given functionality of the software is tested. However, crafting effective test cases, and in particular the manual creation of accurate test assertions, represents a significant bottleneck [15]. Developers dedicate considerable time to manually defining and reviewing these assertions, which could cause delays in releasing a software [8]. While techniques like Search-Based Software Testing (SBST), which automate the generation of test data and test assertions, offer some advancements [7] such as reduced manual effort in the initial generation of tests, and other approaches like dynamic invariant detection [2] exist, they often still rely on manual verification against confirmed correct behaviors or runtime observations, demanding substantial developer involvement [15].

Large Language Models (LLMs) present exciting possibilities for automating and improving aspects of software testing, including test case and assertion generation [10]. Their combined fluency in natural language and code [1, 14] allows them to produce effective and more readable tests [3]. Nevertheless, deploying powerful LLMs like GPT-4 introduces challenges: high computational cost, latency issues, and a dependence on persistent internet connectivity hinder their use in resource-constrained environments. This underscores a major practical limitation of current powerful LLMs as they cannot be easily or effectively deployed everywhere. Smaller, local models often lack the coding sophistication to create appropriate assertions for complex code structures.

Knowledge distillation [13] emerges as a potential solution. By transferring the knowledge of a larger, more capable "teacher" LLM to a smaller "student" model, we could develop a more efficient, localized model capable of assertion generation with acceptable quality, mitigating the drawbacks of larger LLMs. Knowledge distillation is particularly promising as it allows for the transfer of nuanced code understanding and assertion logic, learned by a large model, into a more resource-efficient student model. Although LLMs have been explored for testing and knowledge distillation [3, 10, 13, 14], a practical, small-scale model optimized for efficient and accurate assertion generation for complex code structures is yet to be perfected.

This work aims to bridge this gap by evaluating knowledge distillation for creating a small, efficient assertion generation model. We will distill a large LLM's assertion generation capabilities into a smaller model, assess its performance characteristics against the teacher and a baseline, and evaluate its effectiveness using a dedicated dataset of code. Success in this area could significantly widen the availability of advanced assertion generation capabilities, making them accessible in diverse development environments.

Our distilled student model showed promise in reproducing teacher assertion with results for accuracy, precision, recall and f1-score (all these metrics will be explained later in the paper) ranging from 0.356 to 0.364 and a similarity score of 0.816, while being significantly smaller (around 4 times) than its teacher. The student also showed promise when we tested it against the ground-truth - reproducing around 30% of the factual assertions. While exact replication of the teacher's outputs and the ground-truth was challenging, these results suggest a significant potential for creating efficient, yet reliable assertion generation tools.

2 Background

This section outlines the foundational concepts crucial for understanding the subsequent research: automated test assertion generation, the role and limitations of Large Language Models in this domain, and the principles of knowledge distillation.

Assertion-based Testing. In software testing, an assertion test is a boolean expression in a test case that verifies if a specific condition holds true at a certain point during the program execution. Typically, assertions signal a defect whenever they fail. Manual generation of complete and correct assertions is the most significant bottleneck in software development, consuming much of developers' time and often delaying release.

Search-Based Software Testing. Search-Based Software Testing (SBST) applies meta-heuristic search techniques to generate test data. Thus, SBST automates some parts of test case generation, but the assertions generated from it are observed runtime behavior. This observation means an assertion can be a manifestation of the currently misbehaving behavior of the program rather than the reflectively intended correct behavior; therefore, such assertions need to undergo deliberate manual inspection and improvement.

Dynamic Invariant Detection. Dynamic invariant detection seeks to determine likely invariants of a program from the execution traces. However, such tools generate very many invariants, many of which may be trivial, coincidental, or not immediately useful as test assertions-and thus require the developer to expend heavy effort in curation. The main trade-off both SBST and dynamic invariant detection frequently offer is between the level of automation achieved and the quality, relevance, and manual inspection costs incurred in verifying one asserted statement. **Large Language Models for Assertion-based Testing.** There are substantial trade-offs when it comes to the usefulness of LLMs in real-world software development processes. Large-scale, state-of-the-art LLMs (like GPT-4) perform well but come with high computational costs, which cause generation latency and frequently call for cloud-based APIs, which raises questions about data privacy and continuous connectivity. The nuanced understanding and sophisticated generation capabilities of their larger counterparts are typically absent from smaller, locally executable LLMs, which are faster and more resource-efficient. They frequently struggle with complex code structures or generate generic, less insightful assertions. This leads to a choice between reduced cost/latency and compromised capability or high capability with high cost/latency.

Knowledge distillation. Knowledge distillation is technique that offers an interesting approach to model compression. The core idea behind it is to use the knowledge of a large model, the teacher, to train a smaller one, called the student model. The student model learns both from the teacher one (soft target) as well as from a ground-truth data (hard target). By learning from both the ground-truth labels and the teacher, the student model can often learn more effectively and generalize better than if trained solely on the hard labels. In order to do that, a loss function - a function that "shows the difference" between the student's output and the reference one, is utilized. The final loss for training the student is a weighted sum of the loss on soft targets (mimicking the teacher) and the loss on hard targets (fitting the ground truth). A hyperparameter (alpha) balances the contribution of these two loss components. After that, the weights of the model are updated in a effort to minimize this loss and the process occurs again.

In the context of this, study large language models have showed promise. However, there are some barriers to practical adoption. Knowledge distillation provides an attractive means to extract a large teacher model's nuanced code understanding and assertion-generation rationale into a compact, nimble student model. This could enable the use of helpful, LLM-based assertion generation tools in computationally resource-poor settings, or for applications requiring lower latency.

3 Methodology

With the focus on employing knowledge distillation techniques in building a smaller and efficient language model used for assertion generation, this chapter explains the step-by-step process followed for the study. The CodeT5 [12] "teacher" model is trained and refined, then a specific dataset is used to train a "student" model to mimic the behavior of the teacher model. The evaluation measures cover both performance and resource utilization.

Teacher model. The teacher model for this research is CodeT5, a state-of-the-art pre-trained encoder-decoder model recognized for its strong performance across a variety of code intelligence tasks. CodeT5 relies on the attention mechanism, allowing it to focus on the most relevant parts of the input when generating code[11]. A version of CodeT5, more specifically, the Salesforce/codet5-base [1], available on Hugging Face, was used. This pretrained teacher model serves as the primary source of knowledge for the student model and the benchmark against which the student's assertion quality is measured.

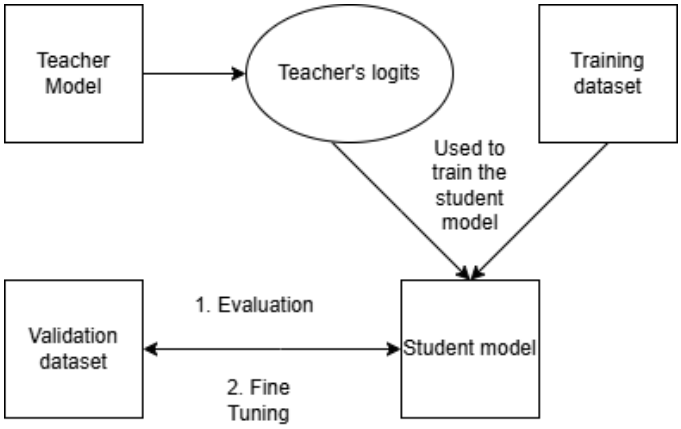

Figure 1: High-overview of the pipeline

Dataset. The dataset [5] consists of Java methods under test (MUTs). For the purpose of training, each MUT is paired with a corresponding test suite where the original assertions have been masked. The complete, ground-truth assertions are retained for evaluation purposes. This structure allows the student model to learn the task of filling in these masked assertions. Each modeling step utilizes a comprehensive training and validation datasets. Importantly, the validation set is guaranteed to contain data that has not been seen by the model during training, preserving an objective evaluation of generalization performance.

Student model. The student model for this research is the Salesforce/codet5-small [2], again available on Hugging Face. We chose the student model to be smaller than CodeT5-base in architecture size in order to properly evaluate the effectiveness of the knowledge distillation.

Knowledge distillation. The most important part of this study is knowledge distillation. The student model is trained on both the ground-truth data and the outputs of the teacher model.

Training. The student model, Salesforce/codet5-small is trained on around 9000 examples, part of the dataset (described in section 3). Training proceeds for a default of 3 epochs (a common, small number for initial fine-tuning experiments to observe learning without excessive time) with a default training batch size of 2.

The optimization process uses the AdamW optimizer [9], configured with a learning rate of 5e-5 (a standard, effective starting learning rate for fine-tuning Transformer models like T5) and weight decay of 0.01 (a common regularization value for AdamW to help prevent

[1] https://huggingface.co/Salesforce/codet5-base
[2] https://huggingface.co/Salesforce/codet5-small

overfitting). A learning rate scheduler is employed, featuring a linear warm-up over the first 10% (helps stabilize training in the initial phase by gradually increasing the learning rate) of the total training steps, followed by a linear decay towards zero by the end of the final epoch. Gradients are accumulated (defaulting to 1 – meaning no accumulation unless a larger effective batch size is needed than fits in memory per step) before an optimizer step. Gradients are clipped to a maximum norm of 1.0 (a standard value to prevent exploding gradients and stabilize training).
At the beginning of each epoch, the student model is set to training mode. The training loop then iterates. Each forward and backward pass for a batch proceeds as follows:

- Data Preparation: A batch of objects is loaded. This includes input_ids, attention_mask, hard_labels, and potentially teacher_logits.
- Student Forward Pass: We pass the input_ids and attention_mask through to the student model to obtain student_logits.
- Loss Computation: We compute a combined distillation loss, using the hard loss and the soft loss:
  - Hard Loss: This is the standard cross-entropy loss which is derived by measuring the student model's assertions with the dataset. This term steers the student model towards the generation of factually correct assertions.
  - Soft Loss: This term motivates the student to follow the output probability distributions (logits) of the fine-tuned CodeT5 teacher model. Calculation is done with the use of Kullback-Leibler (KL) divergence. The distributions are tempered more with a temperature (T) parameter, allowing the transfer of subtle "dark knowledge" from the teacher, such as the relative probabilities of different tokens. Temperature scaling [4], helps to calibrate the output probabilities, leading to more reliable uncertainty estimates.

  The overall loss function is a weighted sum: Total Loss = alpha * SoftLoss + (1 - alpha) * HardLoss. The hyperparameter alpha controls the relative importance of mimicking the teacher versus fitting the hard labels.
- Backpropagation: We backpropagate the loss.
- Optimizer Step: After gradients are accumulated, they are clipped. Then we update the weights, and adjust the learning rate. Gradients are then set to zero.

During the epoch, training progress is logged every 50 steps. At the end of every epoch, we validate the student model on the validation dataset, which consists of around 1000 examples the student has not seen before. Then if this is the best model, we have found so far, we save it to the specified directory. Finally, after all epochs, the final model state is saved regardless of validation performance relative to earlier epochs.

After every training session followed by an evaluation on the validation set, model parameters are updated. This includes:

- Alpha: The value of alpha, which balances the soft and hard losses, is adjusted to control the learning from the teacher and the actual data.
- Temperature: This parameter is adjusted to control how much detail of knowledge is passed down by the teacher's logits.

This iterative refinement process aims to maximize the student model's performance on the validation metrics.

Hardware setup. The training and evaluation steps took place on Google Colab, utilizing an NVIDIA A100 GPU.

4 Study Design

This section aims to explain how we evaluated our student model in order to pursue answers to the research questions: (1) How does the distilled model compare to a larger teacher model and a baseline, ground-truth, in terms of assertion accuracy, similarity, model size? (2) How changes of a certain parameters (temperature, alpha) influence the student's performance? To investigate these problems, we designed the following evaluation framework.

Evaluation setup. The core of our evaluation relies on assessing the student model (Salesforce/codet5-small, pre-trained and then fine-tuned via distillation) on a dedicated validation dataset. This dataset, like the training data, consists of Java Methods Under Test (MUTs) where the original assertions are masked, along with the corresponding ground-truth assertions and the pre-computed outputs (predicted assertions and logits) from the teacher model (Salesforce/codet5-base). The approximately 1000 examples in the validation dataset are those the student model was not exposed to during its training phase, hence they are used for an objective evaluation of how well the student model generalizes. For each entry in the validation set, the following process occurs:

- Input preparation: The student model receives the masked tests along with focal content (the class under test) as input. Then we tokenize the input, using the student model's tokenizer.
- Assertion generation: The student model generates assertions for the given input.
- Output decoding: We decode the token IDs that the student generated back into textual assertions using the student tokenizer.
- Normalization: Before comparison, we normalized the assertions. We did this in order for the comparison to be as fair as possible. Examples of normalization include collapsing multiple whitespaces to one and using the same casing.
- Metric computation: The generated assertions are then compared against the ground-truth assertions and the teacher model's assertions. We will describe the evaluation metrics in the following paragraph.

Evaluation metrics. To comprehensively evaluate the performance of the distilled student model and compare it to the teacher model (CodeT5-base), the following metrics are utilized:

- Precision: This shows the ratio between the number of the exact matches - generated assertions which match completely the reference ones, and the number of the total generated assertions. Higher precision means higher quality of the generated tests.

- Recall: This indicates the ratio between the number of the exact matches to the total number of the reference assertions. It indicates the proportion of reference assertions successfully reproduced by the model.
- F1-Score: The harmonic mean between precision and recall, which aims to provide a balanced measure of the test's overall accuracy in terms of both correctness and completeness of the generated assertion set.
- Accuracy: This is the ratio between the exact matches and the bigger number between the total reference assertions and the generated ones. We penalize the student for creating too many assertions relative to the reference ones.
- Similarity: Measures the character-level similarity. For each normalized generated assertion, its highest similarity ratio against any normalized reference assertion is found, and these best-match scores are then averaged. This provides a granular, character-by-character measure of similarity.
- Model Size: The physical size of the student model is compared against that of the teacher.
- Inference Time: The average time taken by the student model to generate an assertion for a given input is recorded and contrasted with the teacher model's inference time.
- Loss: A combined overall loss between the hard and the soft ones. A lower value suggests that the student is more successful in generating factual assertions and mimicking the the teacher model.

These metrics try to directly answer our research questions that concern the trade-offs between accuracy of assertions, inference speed, and resources consumed by distilled models as compared to these larger teacher models. These evaluation results will lend credence to the question of whether distilled LLMs are a viable option for efficient generation of assertions in disparate development environments.

5 Results

Value	Inference Time (ms/sample)	Loss
0.1	343.3	0.424
0.5	381.9	0.493
1.0	427.7	0.662
1.5	463.1	1.063
2.0	513.1	1.762
3.0	576.9	4.657
4.0	611.7	11.083

Table 1: Inference Time and Loss for the Temperature

In this section we will present the outcomes of our experiments. We will show results which aim to shed some light on the research questions.

Student model versus the teacher one. In this part, we will mainly consider the results which were obtained with temperature = 1.0 and alpha = 0.5. We chose them because the student model produced the best results with them, when compared to the teacher one. We will explain in a later section why this may be the case. The other values for these parameters produced a bit worse results.

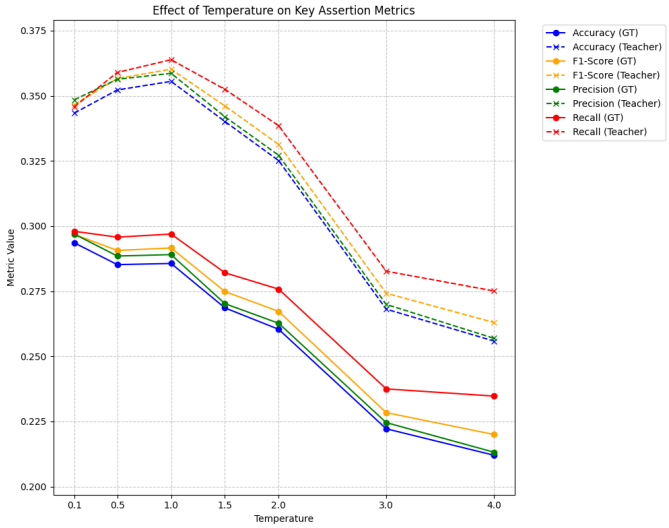

Figure 2: Accuracy, Precision, Recall and F1-Score Metrics for the Temperature

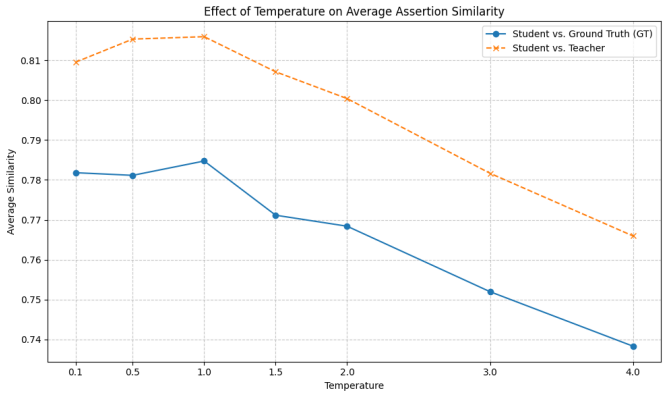

Figure 3: Similarity Metric for the Temperature

Value	Inference Time (ms/sample)	Loss
0.1	365.7	0.791
0.2	395.7	0.754
0.5	427.7	0.662
0.6	424.3	0.626
0.7	466.9	0.596
0.9	522.6	0.537

Table 2: Inference Time and Loss for the Alpha

- Precision: The highest assertion precision we achieved is 0.359 when running the student model against the teacher one.
- Recall: The proportion of the reference assertions successfully reproduced by the student model peaked at 0.364.

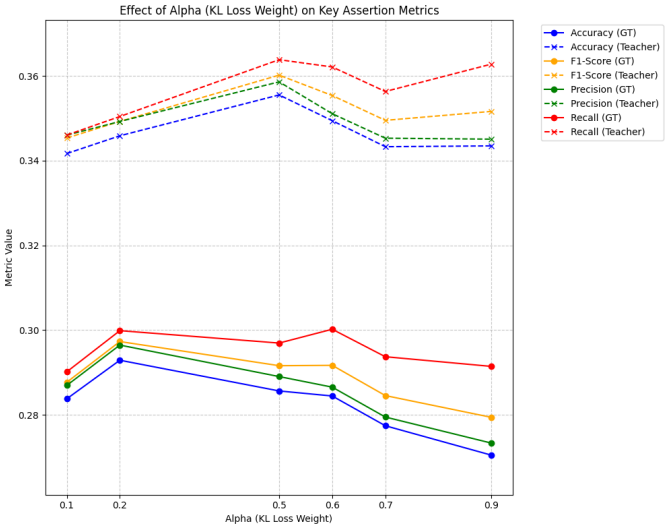

Figure 4: Accuracy, Precision, Recall and F1-Score Metrics for the Alpha

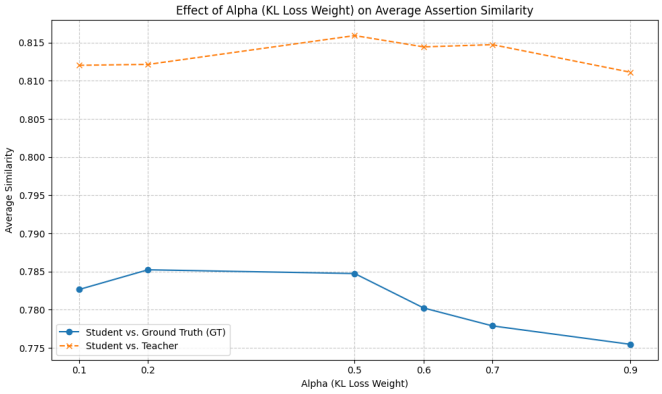

Figure 5: Similarity Metric for the Alpha

- F1-Score: We recorded the harmonic mean between the best results of the precision and the recall metrics to be 0.360.
- Accuracy: The highest accuracy we managed to achieve when testing the student model against the teacher one is 0.356.
- Similarity: The best similarity our experiments produced is 0.816.
- Model size: The chosen student model, namely Salesforce/codet5-small, is approximately 1/4 of the size of the teacher one - Salesforce/codet5-base.
- Inference time: The average time the model needed to generate an assertion for a given input is 427.7 miliseconds (ms).

Student model versus the ground-truth. Here, we will talk about the results which we obtained when comparing the student model against the ground-truth.

- Precision: The precision peaked at 0.297 when running the student, with temperature = 0.1 and alpha = 0.5, against the ground-truth.
- Recall: The highest recall we recorded is 0.3. We achieved this with temperature = 1.0 and alpha = 0.6.
- F1-Score: When we used temperature of 1.0 and alpha of 0.2, we got the best value for the F1-Score - 0.297.
- Accuracy: The highest accuracy we achieved is 0.294 with parameters - temperature = 0.1 and alpha = 0.5.
- Similarity: Temperature of 1.0 and alpha of 0.2 led to the best similarit we achieved - 0.785.

Effect of the temperature. The tests for the temperature were run with alpha = 0.5, meaning that the soft and hard losses had equal impact. We chose this value in order to see what impact the temperature has when comparing to both the teacher and the ground-truth. We will go through the results for each evaluation metric:

- Precision: As we can see from figure 2, the assertion precision, when comparing to the teacher, was the highest at temperature (T) = 1. On the other hand, when comparing to the ground-truth the model performed the best when T = 0.1. However, for both comparisons, the bigger the temperature was (that is true after T = 1 for the comparison with the teacher), the worse the performance got.
- Recall: Here we observe (figure 2) almost the same results as with the Precision. The only difference is that in this case, when comparing to the ground-truth, T = 0.1 and T = 1.0 performed practically the same.
- F1-Score: Considering that this is the harmonic mean of the Precision and the Recall, it is no surprise that we achieved similar results to the two aforementioned metrics as we can see from figure 2.
- Accuracy: As figure 2 shows us, our model peaked at T = 1.0, when comparing to the teacher, whereas, when we consider the ground-truth, the lower the temperature, the better the results we got.
- Similarity: Here the results (figure 3) when comparing against both the ground-truth and the teacher, follow similar pattern - they both peaked at T = 1.0, for T bigger than 1.0 they got increasingly worse and the results were close between T = 0.1 and T = 1.0.
- Inference Time: As we can see from table 1, the higher the temperature, the more the time for generating an assertion for a given input increased.
- Loss: With the increase of the temperature, the overall loss also increased. This can be seen from table 1.

Effect of the alpha parameter. The tests for the alpha were run with temperature = 1.0 because we found it performed the best as we will describe in the Discussion section. We will go through the results for each evaluation metric:

- Precision: We can see from figure 4, that the student performed the best, compared to the teacher, when alpha was 0.5. However, this is not the case for the comparison with ground-truth, where the student performed the best at alpha = 0.2.

- Recall: The best results we got for the student versus the teacher were at alpha = 0.5 and alpha = 0.9. The student practically performed equally well with these values. The graph for the student versus the ground-truth peaked at alpha = 0.2 and alpha = 0.6, although it did almost as well at alpha = 0.5. We can observe this at figure 4.
- F1-Score: As figure 4 shows, the result for the comparison with the teacher gradually improved up until alpha = 0.5 where it peaked. Once again, the value of alpha for which the comparison with the ground-truth peaked was 0.2. However, both alpha = 0.5 and alpha = 0.6 were close second.
- Accuracy: From figure 4 we can see that the correlation from the previous metrics is still intact - the student performed the best at alpha = 0.5, when we test it against the teacher, whereas, when put against the ground truth, the peak was at alpha = 0.2.
- Similarity: When looking at the results for the student versus the teacher for that metric, presented at figure 5, we can see that all values for alpha which we experimented with gave quite close outcomes, but once again alpha = 0.5 performed better than the others. The comparison with the ground-truth shows that the best results were achieved at alpha = 0.2 and alpha = 0.5.
- Inference time: As we can see from table 2, the higher the alpha, the more the time for generating an assertion for a given input increased, with the exception of alpha = 0.6 which led to a slightly shorter time than alpha = 0.5.
- Loss: As table 2 shows, With the increase of the alpha, the overall loss decreased.

6 Discussion of the results

In this section, we will interpret the outcomes presented in the Results section. We will go through the results for every research question.

Discussion about the comparison between the student and the teacher model. We will focus on how the student performed compared to the teacher.

The precision, recall, f1-score and the accuracy metrics collectively indicate the level of fidelity of the student model in mimicking the teacher one. The scores which are between 0.356 and 0.364 suggest that, while the student is not perfectly mimicking the teacher, it manages to reproduce a significant, slightly higher than 1/3, of the teacher's assertions correctly. The similarity offers an interesting perspective. The high score of 0.816, suggest that the student has a significant knowledge of the teacher's style and behavior. These results are particularly valuable as they imply the student can produce assertions that are teacher-like in quality and form, even if not identical on every occasion.

The achieved results become particularly interesting when we consider that the student model size is 1/4 of the teacher's one. This matters because our model can be used in resource-constrained environments while offering a solid performance. The inference time, coupled with the smaller model size, enhances the student model's utility in scenarios requiring rapid assertion generation or where computational resources are limited.

The distilled student model offers a compelling trade-off when compared to the larger teacher model. It achieves a substantial reduction in model size and offers fast inference. This efficiency comes at the cost of perfect, exact replication of the teacher's output. However, the high similarity indicates that the student has learned a big part of the teacher's nuances. When we consider the fact that we used a compressed teacher's logits, which inarguably hindered the results, the outcome suggest that there is potential in knowledge distillation and requires further research.

Discussion about the comparison between the student model and the ground-truth. We will analyze the student model's performance when directly evaluated against the ground-truth assertions.

The precision peaked at 0.297 and the accuracy did so at 0.294 with temperature = 0.1 and alpha = 0.5 for both these best scores. This suggest that for an optimal performance of the student, when tests on the ground-truth assertions, a low temperature is beneficial. This may be the case because a lower T does not allow much exploration - it makes the student model focus on high-probability tokens. A balanced alpha indicates that benefited equally form learning both from the student model and the ground-truth.

The best F1-Score and the parameters, T = 1.0 and alpha = 0.2, we achieved it with, reinforces the observation that the current model's ability to both generate correct and cover a good portion of the ground-truth is around the 30% mark. An alpha of 0.2, which prioritizes learning from the ground-truth (hard loss), seems to align with our intuition that a metric that directly measures correctness against factual data performs the best.

The temperature of 1.0 and alpha of 0.6 for the highest recall we achieved, 0.3, suggest that to capture a broader range of correct assertions, some influence from the teacher's distribution and a slightly less conservative generation strategy might be helpful.

The similarity score of 0.785 implies that, even if the student's assertion are not an exact match, they are at least often lexically and structurally close. The combination of T = 1.0 and a low alpha (0.2) suggests that while focusing on ground-truth, the teacher's unsoftened distribution still helps in shaping these structurally similar outputs.

The performance of the student on these metrics suggest that, even with its small size and the handicap of compressed logits, it is still able to reproduce a little below 30% of the factual assertions. Even though, we obtained the results for the metrics with different parameters, certain ones - temperature of 0.1 and 1.0, alpha of 0.2 and 0.5, consistently performed well and can make for good starting points in further researches.

Discussion about the effect of the Temperature. We will focus on how the temperature influenced the mimicking of the teacher, aligning with the ground-truth and efficiency and training dynamics.

We observe that a temperature of 1.0 consistently emerged as the best option when it comes to replicating the teacher's outputs. A possible reason for this might be that T = 1.0 likely corresponds to the teacher's unsoftened output probability distribution. This distribution might offer the richest "dark knowledge" of the teacher's distinctive features, which the student aims to learn. Values lower than 1.0 may make the teacher's signal too sparse, which can lead

to losing beneficial for the student nuances. On the other hand, temperatures higher than 1 might lead to "over-smoothing" of the teacher's features, making it hard for the student model to learn from them.

The lowest temperature we researched, namely 0.1, was the best performer on almost all of the metrics, although 1.0 was a close second. Lower temperatures makes the student to focus on the predictions the teacher is the most confident about. If these high-confident predictions turn out to be correct (aligned with the ground-truth), this might help the student generate assertions that are more accurate, when compared to the factual truth. In contrast, high temperatures might lead to more ambiguous information relative to the ground-truth.

When it comes to the inference time, it increased with higher temperatures. A possible cause might be that the model, which is trained with softer distributions when the temperature is higher, may explore more token possibilities during generation.

Reported training loss increases significantly with higher temperatures. This is primarily an artifact of the distillation loss function scaling the soft loss component by T to the power of 2, which is a standard technique to maintain the relative impact of the soft loss when distributions are very flat.

We can see that there is not such a thing as an optimal temperature - this largely depends on the distillation objective. However, considering that our main goal is to learn from a teacher model, T = 1.0 emerges as a promising choice.

Discussion about the effect of the Alpha. We will focus on how the alpha influenced the mimicking of the teacher, aligning with the ground-truth and efficiency and training dynamics.

As we can see from the graphs, alpha = 0.5 generally provides the best results for the mimicking of the teacher. This might be because a focus purely on the teacher (very high alpha) might lead to the student replicating the teacher's errors. The inclusion of the ground-truth balances the learning and helps the student to gain a refined version of the teacher's behavior, thus leading to a better overall mimicry of the valuable aspects.

An alpha of 0.2 gives the best performances when it comes to the aligning with the ground-truth. However, an alpha of 0.5 is a close "competitor". Directly optimizing for performance against the ground-truth maximizes the outcome in this department. The teacher's logits, though, still can provide useful information, potentially helping the student generalize better than solely learning from hard labels.

The inference time tends to increase with higher values for alpha, with one exception. A possible reason for this might be that a stronger focus on the teacher, leads to the student learning more complex behavior, hence needing more time.

Overall training loss tends to decrease as alpha increases. This is likely because the KL divergence (soft loss) often yields numerically smaller values than cross-entropy (hard loss) against sparse labels, especially if the student can effectively match the teacher's smoother distribution.

Again the value we choose for alpha depends on the context we are in. Lower values lead to better factual accuracy against the ground-truth, whereas a more balanced alpha is better for capturing the teacher's overall behavior.

7 Threats to Validity

Several factors could influence the interpretation and generalizability of our findings:

- Teacher Logit Compression and Information Retainability: A primary consideration is the use of compressed teacher logits. The original teacher model (Salesforce/codet5-base) produces logits in 32-bit floating-point format. Due to their substantial size, and the necessity of transferring this data (to local machines for dataset preparation or to environments like Google Colab for training), these logits were compressed to 8-bit representations using the LZ4 algorithm. This 32-bit to 8-bit compression inevitably involves a loss of precision. Consequently, the student model learns from a teacher signal whose maximum possible retain is not at its absolute highest compared to using uncompressed, original 32-bit logits. The observed distillation effectiveness is therefore conditional on the quality of these decompressed 8-bit logits.
- Dataset Specificity: Our experiments utilize a dataset of Java methods where assertions are masked. The effectiveness of the distilled student model might differ for other programming languages (e.g., Python, C++) or datasets with assertions generated or masked differently.

Addressing these threats involves further experimentation, using different datasets and utilizing the uncompressed teacher logits. Our current findings should be considered with the aforementioned limitations in mind.

8 Responsible Research

The study is committed to upholding ethics, transparency, and reproducibility. We feel that any advancement of the field of automated software testing, particularly assertion generation, should be straight. That is to say that new tools and techniques must be accessible and their implications understood.

Ethical considerations. The ultimate goal to this work is exploring the feasibility of creating smaller, more efficient LLMs for assertion generation by means of knowledge distillation. Aware of the potential positives, one could consider:

- Accessibility: Offering advanced assertion generation capabilities to developers constrained by environment or simply preferring local tools.
- Efficiency: Higher computational cost and latencies are in the way of development working cycles-long. By being efficient, it could allow for neater handling of those issues.

To the contrary, we also consider the following potential ethical considerations:

- Over-reliance: In the end, a developer might start to overly rely on automated assertion generation without review. We insist that such distilled models should be used as assistive tools, complementing developer knowledge and critical judgment in test assertion generation. Human oversight is still necessary.
- Quality of Distilled Knowledge: The performance of the student model is inherently linked to the quality of the

teacher model and the data. A distilled model retains useful knowledge, so it must continuously be evaluated for usefulness and limitations.

Reproducibility and Transparency. This study is based on publicly available resources and intends to make as many core parts as possible publicly accessible:

- Dataset: The dataset used for training and evaluation, which consists of Java methods with masked assertions and teacher model logits, is publicly available at Zenodo [5]. Being based on publicly available code sources for the dataset makes the data origins transparent.
- Models: Both the teacher model (Salesforce/codet5-base) and the student model (Salesforce/codet5-small) are open-source models to which one can freely gain access on platforms such as Hugging Face, hence affording other researchers the opportunity to use the same underlying architectures.
- Methodology and Code: The distillation process, combined loss function, training procedure, and all evaluation metrics are fully described in Section 3 (Methodology) and Section 4 (Study Design). The core Python script used for conducting the experiments is available at Zenodo [6] including default hyperparameter settings, to enable replication of our study design.

By adhering to these principles, we hope to contribute positively to the research community, fostering an environment where advancements in LLMs for software engineering are developed and deployed ethically and transparently.

9 Conclusions and Future Work

The main objective of this work was to verify the possibility of knowledge distillation into a smaller, and thus, more resource-friendly, assertion generation model. It was possible to successfully distill knowledge from the larger CodeT5-base, acting as the teacher, into the CodeT5-small as the student, thus designing a trade-off between performance and resource burden for deployment. The study further explored the role of hyperparameters, temperature and alpha, in enabling the student to reproduce the teacher and the ground-truth assertions. Our results confirmed that knowledge distillation can be considered a way of developing smaller models for test assertions. Distillation-wise, it hugely shrinks by a factor of 4 with the student model, while inference is very fast. While an exact duplication of the teacher´s and the ground-truth assertions was almost impossible, the student model showed great similarity in output, indicating that a great deal of stylistic and structural knowledge got transferred. The student model was able to correctly reproduce around one-third of the teacher assertions and around 30% of the factual assertions.

Optimal hyperparameter settings were found to be context-dependent: a temperature of 1.0 and alpha of 0.5 yielded the best mimicry of the teacher model, while lower temperatures (e.g., 0.1) and a lower alpha (0.2) generally improved alignment with ground-truth assertions. This highlights the trade-off between reproducing a (potentially imperfect) teacher and directly optimizing for ground-truth accuracy. Despite the inherent information loss from using compressed 8-bit teacher logits, the results affirm the potential of distillation in this domain.

Being limited to compressed (8-bit) teacher logits might have capped the student's highest achievable performance in this study. The original 32-bit logits would probably have supplied more information to teach. Another point is that our findings are based on one dataset of Java methods with masked assertions, whether it generalizes to other languages, other kinds of code structures, or other assertion generation tasks requires further investigation. Subsequent studies should ideally experiment with uncompressed (32-bit) teacher logits to set a higher performance ceiling above which the student model can learn. It would also be interesting to explore the effects of different student model architectures than CodeT5-small. Moreover, extending the evaluation to other datasets involving other programming languages and test generation scenarios would increase the generality of these findings. Finally, human evaluation of generated assertions would add qualitative insights into their practical utility beyond automated metrics.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Wojciech Zaremba, et al. 2021. Evaluating Large Language Models Trained on Code.

[2] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69, 1-3 (2007), 35–45.

[3] Gregory Gay. 2023. Improving the Readability of Generated Tests Using GPT-4 and ChatGPT Code Interpreter. In *International Symposium on Search-Based Software Engineering*. Springer Nature Switzerland, Cham.

[4] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. *arXiv preprint arXiv:1703.06874* (2017). arXiv:1703.06874 [stat.ML]

[5] Kristian Hristov. 2025. *Distillation dataset.* doi:10.5281/zenodo.15716445

[6] Kristian Hristov. 2025. *Replication package v1.0.* doi:10.5281/zenodo.15716434

[7] Gunel Jahangirova and Valerio Terragni. 2023. SBFT tool competition 2023 – Java test case generation track. In *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. IEEE.

[8] Capers Jones. 2011. The costs of inadequate software testing! (2011).

[9] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[10] Max Schäfer, Peiyi Wu, Stefan Noller, Nils Göde, Jannis Hörst, Stefan Küpper, Lars Lüdicke, Tobias Rose, Michele Tufano, Christian Kästner, and Alexey Svyatkovskiy. 2023. An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering* (2023).

[11] Vaswani. 2017. Attention is All You Need.

[12] Yifeng Wang, Wenhu Wang, Shijie Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[13] Chuanpeng Yang, Yao Zhu, Wang Lu, Yidong Wang, Qian Chen, Chenlong Gao, Bingjie Yan, and Yiqiang Chen. 2024. Survey on Knowledge Distillation for Large Language Models: Methods, Evaluation, and Application. *ACM Transactions on Intelligent Systems and Technology* (2024).

[14] Quanjun Zhang, Weifeng Sun, Chunrong Fang, Bowen Yu, Hongyan Li, Meng Yan, Jianyi Zhou, and Zhenyu Chen. 2024. Exploring Automated Assertion Generation via Large Language Models. *ACM Transactions on Software Engineering and Methodology* (2024).

[15] Yucheng Zhang and Ali Mesbah. 2015. Assertions are strongly correlated with test suite effectiveness. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*.