Distilling LLaMA

Kevin Jones
Purdue University
jone2342@purdue.edu

1 Introduction

Large language models (LLMs) like LLaMA have become incredibly powerful for a wide range of natural language tasks, but that performance comes with a steep cost in computation and memory. This makes them hard to use in settings where resources are limited or where latency is critical. To make these models more efficient without sacrificing too much accuracy, I focused on knowledge distillation: training a smaller "student" model to learn from a larger, more capable "teacher" model.

For this project, I distilled the LLaMA 3.2 1B model into a smaller version by first running forward passes through the teacher to capture both its final predictions (hard labels) and its probability distributions over all possible outputs (soft labels). I then trained the student model using a mix of standard cross-entropy loss on the hard labels and KL divergence loss on the soft labels, applying temperature scaling and an α-weighted balance between the two loss terms.

To see how well the distilled model held up, I evaluated it on several NLP benchmarks, MMLU, Hellaswag, AGIEval, ARC-Challenge, SQuAD, and OpenbookQA, using log-likelihood-based scoring. This let me measure how much accuracy was retained, how inference speed and memory usage improved, and how well the model generalized across different types of tasks.

By the end, I had a distilled version of LLaMA 3.2 1B that kept much of the original's performance while being faster and lighter, making it more practical for real-world deployment in resource-constrained environments.

2 Distillation

To distill the LLaMA 3.2 1B model, I began by removing every other transformer layer but ensuring that the last layer was kept. The original LLaMA 3.2 1B model had 1.23 billion parameters. After removing every other transformer layer, the new distilled model had 749 million parameters. All models were trained on the fineweb dataset and were quantized to bfloat16 to allow for faster training. The models were trained with knowledge distillation. I trained the models using the soft loss from the teacher model as well as the hard masked language modeling loss. The loss function was calculated as

$$\mathcal{L} = \alpha\mathcal{L}_{soft} + (1-\alpha)\mathcal{L}_{hard}$$

The $\mathcal{L}_{hard}$ loss is the cross entropy loss between the predicted logits and the ground truth label. For the $\mathcal{L}_{soft}$ I tried two different choices: cross entropy loss and KL divergence. The per token cross entropy soft loss $l^i_{soft\_CE}$ is calculated as

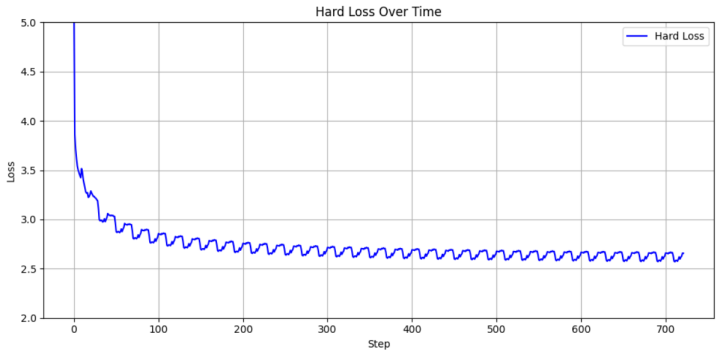
$$l^i_{soft\_CE} = -\sum_{c=1}^{C} \log \frac{\exp(x_{i,c})}{\sum_{c'}^{C} \exp(x_{i,c'})} y_{n,c}$$

The average loss across all tokens is then calculated. This represents the soft loss where $y_{i,c}$ comes from the ground truth logits from the teacher model. The per token KL divergence soft loss $l^i_{soft\_KL}$

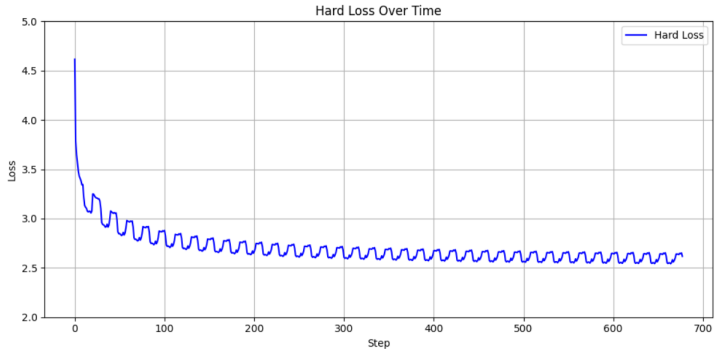
is calculated as

$$l^i_{soft\_KL} = -\sum_{c=1}^{C} \log \frac{\exp(x_{i,c})}{\sum_{c'=1}^{C} \exp(x_{i,c'})} y_{i,c} + \sum_{c=1}^{C} y_{i,c} \log y_{i,c}$$

Again the average loss across all tokens is calculated. The KL divergence loss is very closely related to the cross entropy loss but it adds the second term which accounts for the entropy of the target distribution; this term helps measure how confident or uncertain the target distribution is, and including it ensures the loss penalizes deviations more when the teacher (soft target) is confident and less when it is uncertain. I tested each loss empirically to see which one performed better. Figure 1 depicts the training loss curves for two models, one trained using cross entropy for the soft loss, and the other using KL divergence for the soft loss.



Model trained with cross entropy loss

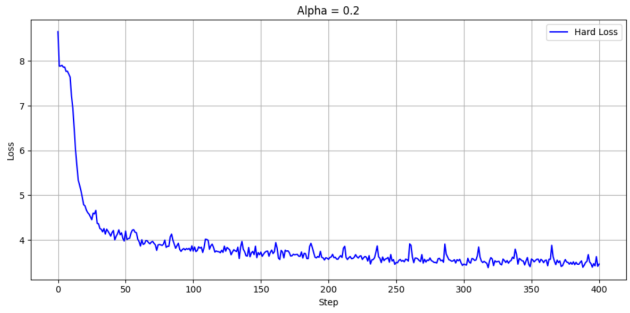


Model trained with KL divergence loss

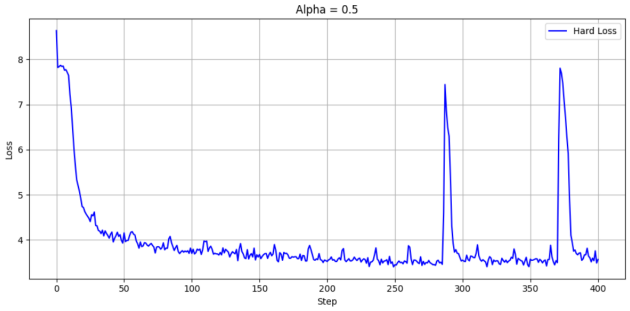
Figure 1: Loss comparison between cross entropy loss and KL divergence

Because the values of the soft loss for each loss type differ in range, the y-axis represents the hard loss of the tokens as it is the same for both models. The x-axis is the number of training steps divided by 100. Both models were trained on approximately 2.8 billion tokens from the fineweb dataset. As seen from Figure 1 the two perform very similarly, but the model trained with the KL divergence loss does achieve a marginally smaller hard loss, so for future training the KL divergence loss was used for the soft loss. One important thing to address from the above loss curves is their cyclic pattern. This is due to a small bug where the model trained on the same part of the dataset over and over again instead of training over the full dataset which resulted in the cyclic pattern above. This bug was fixed for the next iteration of models, and the above models were only used to decide which loss function to use for the soft loss.

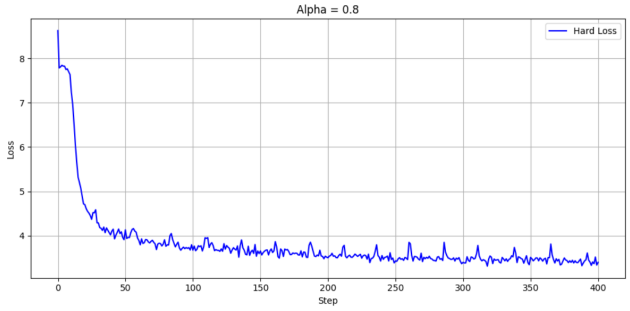
The next set of experiments performed was to choose the value for α which determines what fraction of the overall loss comes from the soft loss versus the hard loss. For this I tested three possible values $\alpha_{\text{candidate}} \in \{0.2, 0.5, 0.8\}$.



Alpha = 0.2



Alpha = 0.5



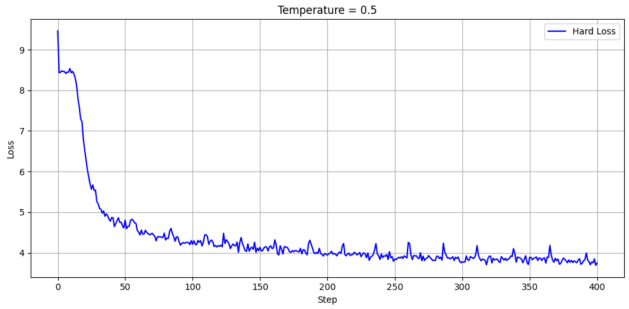
Alpha = 0.8

The figures on the left depict the training loss curves when using the KL divergence soft loss and respective alpha values. Again the hard loss is used to compare the alpha values. Each model is trained only on approximately 40 million tokens as the models do not need to be trained for an extensive period of time in order to observe their performance and choose an alpha value. I notice that the model trained with an alpha value of 0.5 diverges slightly several times. This is likely due to the fact that when I was training these models, I did not do any gradient clipping. This was implemented later when I did a large training run using the chosen hyper-parameters. When observing the figures on the left, I noticed there was not a major difference in training loss for each alpha value. I decided to choose 0.8 for the alpha value because I wanted the distilled model to make use of the soft labels from the teacher model. A higher alpha value means the soft loss contributes more to the overall loss, so during back propagation the soft loss will have a more significant impact on weight updates than the hard loss. I still train on the hard loss to allow the model to learn to attribute higher probability to the correct word.

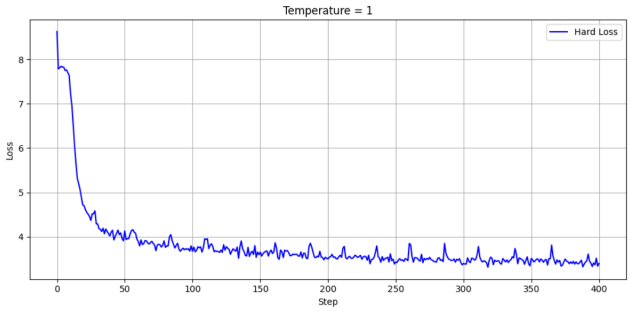
The final set of experiments I performed were to choose the temperature for the teacher model. The temperature parameter T controls the softness of the output probability distribution produced by the teacher model. Specifically, the logits $x_{i,c}$ for each class c are divided by T before applying the softmax function:

$$p_{i,c}^{(T)} = \frac{\exp\left(x_{i,c}/T\right)}{\sum_{c'=1}^{C} \exp\left(x_{i,c'}/T\right)}$$

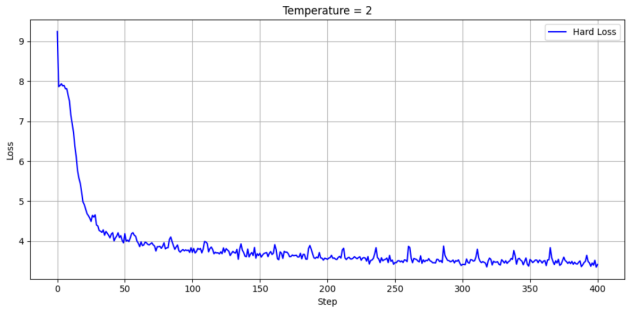
When $T > 1$, the logits are scaled down, which makes the output distribution softer, meaning the probabilities are more spread out across classes. This exposes more information about class similarities, as the teacher assigns non-negligible probability mass to secondary classes it considers plausible. In contrast, when $T < 1$, the logits are scaled up, making the softmax outputs more peaked and confident. This leads to harder targets that closely resemble one-hot vectors. For this I tested three possible values $T_{\text{candidate}} \in \{0.5, 1, 2\}$.



Temp = 0.5



Temp = 1



Temp = 2

The figures on the left depict the training loss curves when using the KL divergence soft loss and respective temperature values. An alpha value of 0.8 was used here. Again the hard loss is used to compare the alpha values. Each model is trained only on approximately 40 million tokens as the models do not need to be trained for an extensive period of time in order to observe their performance and choose a temperature value. When observing the training loss curves on the left, I noticed that when the temperature is set to 1 or 2, the models achieved a very similar loss over the training steps. However, when the temperature is set to 0.5, the loss does not get as low. This is likely due to the fact that a temperature below 1 will harden the targets from the teacher as explained above. This essentially lessons the amount the student model can learn from the teacher model. In the extreme case where T approaches 0, the logits become a one-hot vector with the token the teacher assigns the highest probability to being 1 and all others being 0. This is essentially a bad version of the masked language modeling loss because instead of training on the gold label, the model trains on what the teacher thinks is the gold label. I ended up choosing temperature to be 1 for future training to allow the model to learn on the exact teacher logits.

From my experiments, I chose to train a distilled model using the KL divergence for the soft loss, an alpha value of 0.8, and temperature equal to 1. Figure 2 shows the training and validation loss that the model was trained on. The loss is a combination of the soft and hard loss according to the equation

$$\mathcal{L}_{\text{total}} = 0.8\mathcal{L}_{\text{soft}} + 0.2\mathcal{L}_{\text{hard}}$$

I don't expect any overfitting to the fineweb dataset because it is enormous and I cannot even train over one epoch, but I do hold out a small validation set to be sure. The validation set consists of 100 samples from the fineweb dataset. The validation set is small so that I can evaluate it during training without it taking too long. By chance, the model records a slightly higher loss on the validation set I hold out and so the red curve is above the blue training curve. Nevertheless, I can be sure the model is not overfitting. The model is trained on approximately 2.2 billion tokens. The loss appears to generally converge after the training; however, if I was able to train on many billions or even trillions of tokens as the original LLaMA model was, it is likely that the loss would continue to decrease fairly substantially. One important detail to note is that I only trained on the fineweb dataset. I chose the fineweb dataset because of its size and in hopes that it would capture much general language information. However, because I only use a small portion of this dataset, I suspect that the models are somewhat limited due to the lack of diversity in the dataset because of my limited compute.

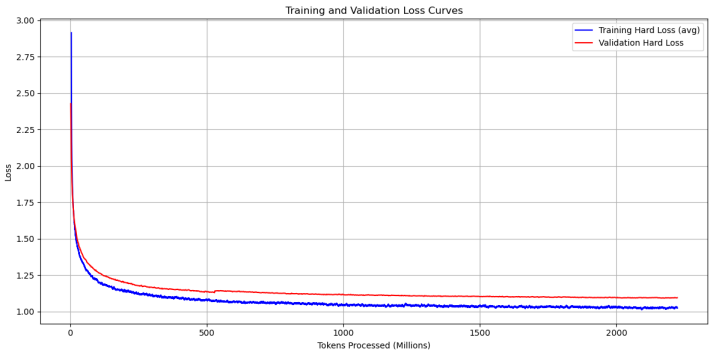


Figure 2: Training and Validation Loss Curves

3 Results

To evaluate the models, I chose an evaluation suite consisting of `MMLU`, `Hellaswag`, `AGIEval`, `Arc Challenge`, `SQuAD`, and `OpenbookQA`.

Evaluating pretrained language models presents challenges due to the mismatch between the nature of generative language modeling and the discrete-choice format of many standard NLP benchmarks. Since these models are typically trained to generate the next token, not to choose from a set of discrete options, I must convert multiple-choice tasks into a form that aligns with the model's likelihood estimation abilities.

For multiple-choice tasks such as `MMLU`, `Hellaswag`, `AGIEval`, `Arc Challenge`, and `OpenbookQA`, I reformulate the problem into a generative one by computing the log-likelihood of each candidate answer. Given a prompt x and a set of candidate completions $\{a_1, a_2, \ldots, a_k\}$:

$$\log p(a_j|x) = \sum_{t=1}^{T} \log p(a_{j,t}|x, a_{j,<t})$$

where $a_{j,t}$ is the t-th token of the j-th candidate. The candidate with the highest total log-likelihood is selected as the model's prediction:

$$\hat{a} = \arg\max_j \log p(a_j|x)$$

This enables me to use generative models for tasks that originally assume classification behavior.

For extractive question answering benchmarks like `SQuAD`, evaluation is handled differently. The model generates a free-text span in response to a question and passage, and its output is compared to a set of ground-truth answers. A prediction is considered correct if it matches any ground-truth answer exactly or has sufficient token overlap, depending on the metric used. In the setup, I use a strict "completion" accuracy, measuring whether the model generates the correct span verbatim:

$$\text{SQuAD Completion Accuracy} = \frac{\text{\# exact match predictions}}{\text{\# total questions}}$$

The metrics used are defined below:

- **macro_avg/acc_char** (used in `MMLU`): This metric averages accuracy across all tasks in the benchmark suite (macro average), where accuracy is defined at the character level to capture minor formatting differences in completions.
- **average/acc_char** (used in `AGIEval`): This is the average of the accuracies of each subtask. Accuracy is again computed using character-level matching.
- **completion** (used in `SQuAD`): A stricter metric that counts only exact completions as correct. Unlike standard F1 or EM metrics used in SQuAD, this version assumes a single correct output form and penalizes formatting variations more harshly.

Table 1 displays the accuracies achieved by each model across the chosen benchmark suite. The 'Structured' and 'Unstructured' models refer to the pruned models using the techniques described in Section 3. The (FT) refers to the finetuned model whereas the (base) refers to the base pruned model without any additional finetuning.

Model	**MMLU** macro_avg	**Hellaswag** acc_char	**AGIEval** avg acc_char	**SQuAD** completion	**ARC-Challenge** acc_char	**OpenbookQA** acc_char
`LLaMA-3.2-1B`	0.313	0.483	0.215	0.400	0.363	0.296
`Distilled`	0.247	0.294	0.228	0.205	0.177	0.138

Table 1: Evaluation metrics for base, pruned, and distilled models across multiple NLP benchmarks.

4 Analysis

The distilled model consistently underperformed the original LLaMA 3.2 1B on most benchmarks, which is expected given the compression from distillation. Since the student model is smaller, it inevitably loses some of the representational capacity of the teacher. This was most evident in benchmarks like Hellaswag, ARC-Challenge, and OpenbookQA, where accuracy dropped significantly. These tasks rely heavily on commonsense reasoning and pattern recall, suggesting that the student may not have retained enough of the teacher's knowledge in these areas.

On the other hand, I saw a surprising result on AGIEval, where the distilled model actually outperformed the teacher. I believe this is due to the nature of the distillation process, specifically, the use of soft labels with temperature scaling. These soft targets carry richer information about the teacher's uncertainty and relationships between possible answers, which can act as a form of regularization. The student may have learned a more generalized decision boundary for the types of reasoning AGIEval requires, leading to better performance than the teacher.

Performance on MMLU was low for both teacher and student, and close to random chance. I suspect this has more to do with the multiple-choice evaluation method than the model's actual reasoning ability—since both models were trained as generative LMs, they may have been producing answers outside the choice set, and restricting the evaluation to only letter choices likely penalized them heavily.

For SQuAD, which requires extracting or generating spans of text from a passage, the distilled model did worse than the teacher but still showed a reasonable degree of retention. This suggests that while distillation can preserve some higher-level comprehension skills, the reduced capacity limits the student's ability to handle more detailed retrieval-based questions.

Overall, distillation did succeed in creating a much smaller and faster model, but with trade-offs in accuracy that varied depending on the benchmark. The gains in efficiency make it promising for deployment in resource-constrained environments, especially for tasks where absolute accuracy is less critical or where soft-label training provides a generalization benefit.