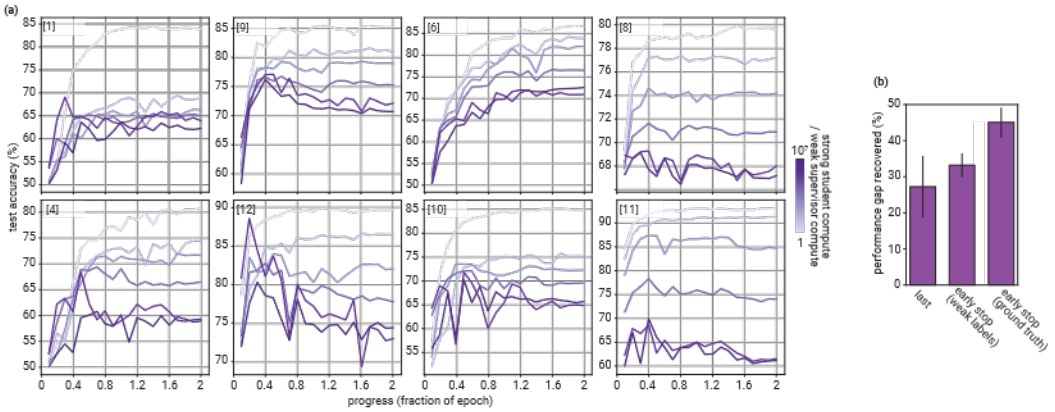
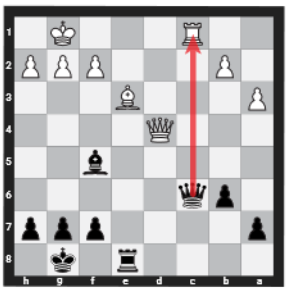


Figure 13: **Overfitting during training, for NLP datasets. Strong models overfit to the weak labels. (a)** Ground truth test accuracy of strong students over the course of training for a subset of our NLP task. Hues indicate the gap between weak supervisor and strong student model compute. Inset numbers indicate dataset id (compare Figure 12). (**b**) Median best, early-stopped according to weak label agreement, and final performance gap recovered (PGR) aggregated across all supervisor-student pairs and all NLP tasks. Error bars indicate standard error of the mean (s.e.m.).



Prompt: "1. d4 1... Nf6 2. Nf3 2... d5 3. e3 3... e6 4. Bd3 4... c5 5. c3 5... Be7 6. Nbd2 6... O-O 7. O-O 7... Nc6 8. Re1 8... Bd7 9. e4 9... dxe4 10. Nxe4 10... cxd4 11. Nxf6+ 11... Bxf6 12. cxd4 12... Nb4 13. Be4 13... Qb6 14. a3 14... Nc6 15. d5 15... exd5 16. Bxd5 16... Bf5 17. Bxc6 17... Qxc6 18. Nd4 18... Bxd4 19. Qxd4 19... Rfe8 20. Rxe8+ 20... Rxe8 21. Be3 21... b6 22. Rc1 22..."

Label: " Qxc1+"

(a) Elo-695 puzzle

Prompt: "1. e4 1... e5 2. Nc3 2... Nf6 3. Nf3 3... Nc6 4. Bb5 4... Bc5 5. Bxc6 5... dxc6 6. d3 6... Bg4 7. h3 7... Bxf3 8. Qxf3 8... O-O 9. g4 9... Bb4 10. Bd2 10... Nd7 11. h4 11... Be7 12. g5 12... Nc5 13. O-O-O 13... Qd7 14. h5 14... Qd8 15. Qg3 15... Ne6 16. Rdg1 16... b5 17. Qxe5 17... a5 18. f4 18... Re8 19. Qf5 19... b4 20. Na4 20... Nd4 21. Qg4 21... c5 22. f5 22... Ra6 23. f6 23... Bd6 24. fxg7 24... Kxg7 25. Rg2 25... Qc8 26. h6+ 26... Kg8 27. Qh5 27... Qd7 28. Rf1 28... Re6 29. Rgf2 29... Rg6 30. c3 30... bxc3 31. Nxc3 31... a4 32. Nd5 32... Qb5 33. Nf6+ 33... Kh8 34. Qh3 34... Rb6 35. Be3 35... Ne6 36. Nxh7 36... Qxd3 37. Rd1 37... Qc4+ 38. Kb1 38... Qxe4+ 39. Ka1 39... Be5 40. Nf6 40... Qc4 41. Nd5 41... Rb7 42."

Label: " Qf5"

(b) Elo-2253 puzzle

Figure 14: **Chess puzzles: example datapoints.** Two representative examples of an easy **(a)** and a hard **(b)** chess puzzle with corresponding prompts and target label formats.

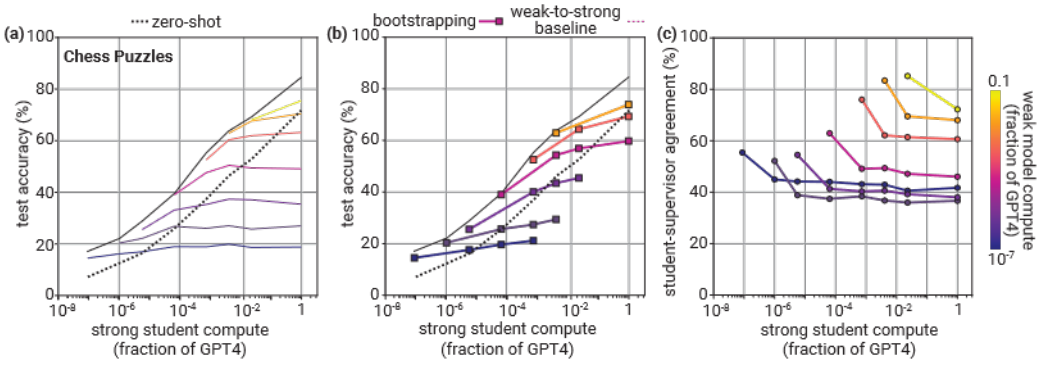


Figure 15: **Additional results on chess.** Test accuracy of **(a)** baseline and **(b)** bootstrapping (see section 4.3.1) compared to a zero-shot baseline. Zero-shot performance improves with model size, and students supervised with much weaker supervisors sometimes underperform compared to the corresponding zero-shot model. **(c)** Supervisor-student agreement on the chess puzzle data. Similar to Figure 8, agreement decreases as the student becomes larger. Hue of line indicates compute of weak supervisor.

Zero-shot results. In Figure 15(a, b), we compare the naive baseline and bootstrapping (see section 4.3.1) generalization to a zero-shot baseline on the chess puzzle data. Especially since the models were pretrained on chess games, zero-shot evaluation provides a strong baseline. In particular, strong students trained with much weaker supervisors underperform the zero-shot baseline for the same model size in some cases.

Supervisor-student agreement results. In Figure 15(c), we report the supervisor-student agreement on the chess puzzles. Similar to the NLP tasks (see Section 5.1.3), the agreement on chess also decreases as the student models get larger.

A.3 ChatGPT Reward Modeling

Data preprocessing. Each datapoint presents a dialog d between a user and an assistant, with a last message coming from the user; for each dialog, there are multiple candidate completions $(c_1, c_2, \ldots, c_m)$, i.e. responses from the assistant. We also have access to pairwise comparisons of completions, where the labeler specifies the preferred completion within a given pair of completions. To sum up, the datapoints can be viewed as (d, c_1, c_2, y), where the label y is 1 if the labeler preferred completion c_2 and 0 otherwise. We use a mixture of multiple datasets used to train the reward models for ChatGPT.

Models. To adapt the language models to the reward modeling setting, we replace the unembedding layer of the model with a linear head with a single output, which is the logit for a given completion. The weights for this head are initialized to the unembedding weights of an arbitrary token in the original embedding layer. Similar to past work (Stiennon et al., 2020; Ouyang et al., 2022), we run two forward passes for each comparison, and the model prediction is given by $\sigma(\mathcal{M}_w(d, c_2) - \mathcal{M}_w(d, c_1))$, where σ is the sigmoid function and $\mathcal{M}_w(d, c)$ is the logit for completion c predicted by the model.

Training hyperparameters. We train for 1 epoch with a batch size of 220. We do not apply early-stopping.

Weak labels. We train the weak models on half of the available comparison data, and then make predictions on the other half. The weak label y_w for a comparison (d, c_1, c_2) is given by $y_w = \sigma(\mathcal{M}_w(d, c_2) - \mathcal{M}_w(d, c_1))$, where σ is the sigmoid function and $\mathcal{M}_w(d, c)$ is the logit for completion c predicted by the weak model.

Supervisor-student agreement results. In Figure 16, we report the supervisor-student agreement on the RM task. Similar to the NLP tasks in Figure 8 and chess puzzles in Figure 15(c), the agreement decreases as the student gets larger.

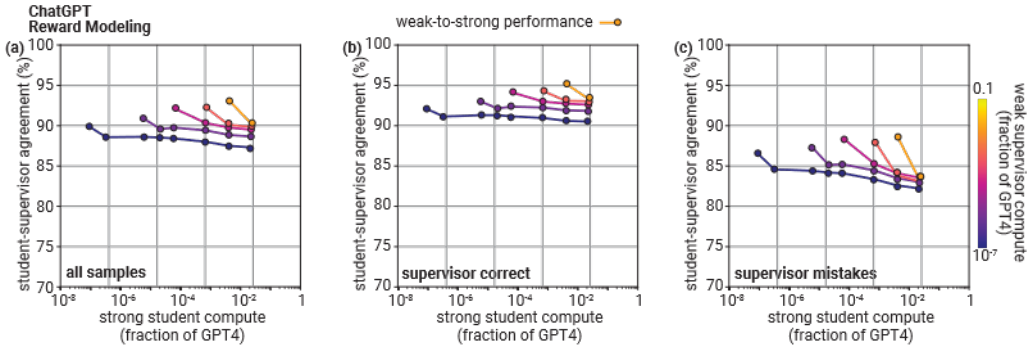


Figure 16: **Supervisor-student agreement decreases for stronger students on RMs.** Please refer to caption of Figure 8 for detailed explanation of the plot. We reproduce the supervisor-student agreement experiment on the reward modeling data, and observe similar trends to the NLP tasks.

Generative finetuning. In Figure 17, we show that the PGR improvements from the generative finetuning on RM data (Section 5.2.2) and from early-stopping on ground truth test accuracy (Section 5.1.1) stack together, leading to results competitive with the NLP and chess settings. In Figure 18, we report the results of an experiment similar to Figure 10, but where the weak models are also pretrained with an additional generative finetuning step on the RM data.

A.4 Auxiliary Confidence Loss

Here, we provide a detailed description of the method we use in Section 4.3.2.

We use the following loss function:

$$L_{\text{conf}}(f) = (1-\alpha)\cdot \text{CE}(f(x), f_w(x)) + \alpha \cdot \text{CE}(f(x), \hat{f}_t(x)) \tag{1}$$

where $\text{CE}(\cdot,\cdot)$ is the cross-entropy loss between the predictive distributions on a given input x, $f_w(x) \in [0,1]$ represents the weak label predictive distribution, $f(x) \in [0,1]$ is the strong model predictive distribution, α is a weight and t is a threshold. The predictions $\hat{f}_t(x)$ correspond to hardened strong model predictions using a threshold t, i.e. $\hat{f}_t(x) = I[f(x) > t] \in \{0,1\}$ where I is the indicator function. We set the threshold t adaptively, so that $f(x) > t$ holds for exactly half of examples in the batch[7]. We set $\alpha_{\max} = 0.75$ for the largest student models and to 0.5 otherwise and linearly warm-up α from 0 to $\alpha_{\max}$ over the first 20% of training.

Our balancing mechanism incorporates a prior over the distribution of labels into training and is only practically feasible in the low-n classification setting. For most weak-strong pairs and datasets, it had a small or neutral effect on weak-to-strong generalization; however, in a few settings it made a significant improvement.

We note that the loss in Equation 1 can be rewritten as a self-bootstrapping loss:

$$L_{\text{conf}}(f) = \text{CE}(f(x), (1-\alpha)\cdot f_w(x) + \alpha \cdot \hat{f}_t(x)), \tag{2}$$

i.e. the cross-entropy target is a mixture of the weak model predictions and the (thresholded) predictions of the strong student itself. This loss is related to the bootstrapping methods in Reed et al. (2014) and Arazo et al. (2019) for addressing label noise. It is also similar to self-training (Lee et al., 2013) and conditional entropy minimization (Grandvalet & Bengio, 2004), which have led to state-of-the-art results in semi-supervised learning (Xie et al., 2020) and domain adaptation (Shu et al., 2018). Chen et al. (2020b) and Wei et al. (2020) show that self-training can mitigate the bias of the supervisor model.

In Appendix B we also describe other methods we considered; for most of these methods, we got negative early results.

[7]The choice of exactly half reflects the prior over classes, and should be computed explicitly from weak model predictions in non-balanced or non-binary settings.

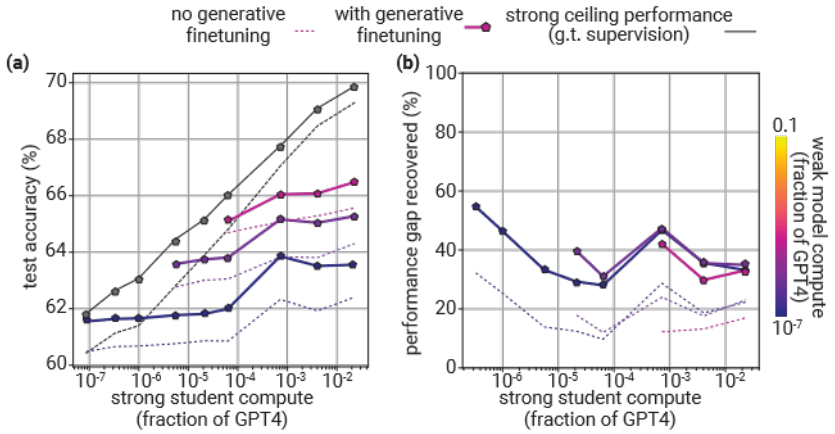


Figure 17: **The benefits of improved task-specific tuning and ground truth early stopping stack, resulting in even higher PGR.** Like Figure 10 but with ground truth early stopping based on test accuracy.

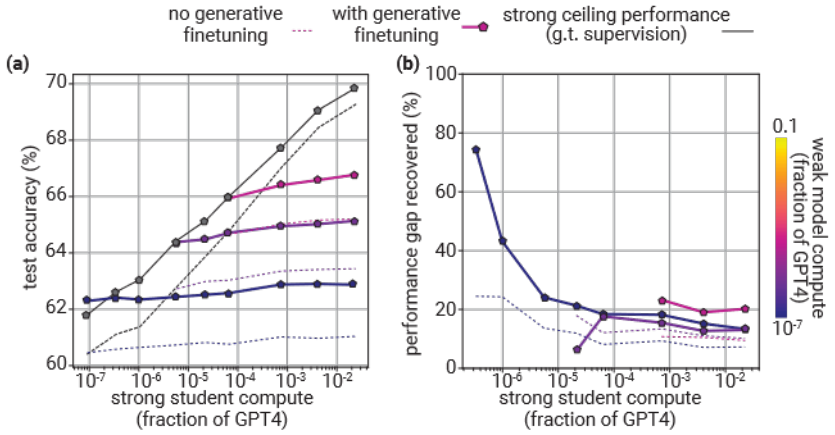


Figure 18: **PGR improves when *both* supervisors and students have an extra generative finetuning step.** Like Figure 10 but where "with generative finetuning" indicates that both supervisors and students have an extra generative finetuning step. In other words, for this experiment all base models have an extra generative finetuning step following pretraining.

B Additional results on methods

We did preliminary experiments on a variety of methods for improving the strong model performance in our weak-to-strong generalization setting. We found many of them not useful for improving over the naive finetuning baseline, and others yielding limited improvements on a subset of settings but not consistently over all datasets and model sizes. We summarize the algorithms, the motivations, and the takeaways below. Note that we did not optimally tune each of the methods, so it is possible that with better tuning they may still perform well.

Confidence thresholding. To filter out incorrect weak labels, we used a simple cut-off method that selected only the top 5% to 20% examples from each class where the weak supervisor is most confident to train the strong model. We found that our weak labels are typically well-calibrated, but confidence thresholding only helps when the weak labels are very bad (e.g. 60% accuracy) and stops being useful when the weak labels reach around 70% to 80% accuracy. We observed these results both in NLP and in the chess puzzle settings. See Appendix C for more discussion of related experiments.

Confidence losses. To encourage strong model to make confident predictions (Grandvalet & Bengio, 2004), we added an auxiliary loss that encourages the model predicted class probabil-

Table 2: **Custom prompts used in the zero-shot and few-shot experiments**. We design a simple custom prompt for each of the tasks in the table below. In the few-shot setting, we also append labeled (with ground truth or weak labels) examples to the prompt.

Dataset	Custom Prompt
CosmosQA	*There is some context, followed by a question about the context, and an answer. Respond with 1 if the answer is correct, and 0 otherwise.*
Deontology	*There is a short scenario, consisting of a situation or request, followed by an excuse. Respond with 1 if the excuse is valid, and with 0 otherwise.*
Justice	*There is a statement. Respond with 1 if it is correct or just and with 0 otherwise.*
QuAIL	*There is a news context, followed by a question about the context, and an answer to the question. Respond with 1 if the answer is correct, and with 0 otherwise.*
SciQ	*There is a science knowledge question, followed by an answer. Respond with 1 if the answer is correct, and with 0 otherwise.*
Social IQa	*There is some context, followed by a social question, followed by an answer. Respond with 1 if the answer is correct, and 0 otherwise.*
Virtue	*There is a short scenario, followed by a judgement of the person involved. Respond with 1 if the judgement is correct, otherwise respond with 0.*

ity p to be far away from 0.5. We tried both the l_2 loss $-(p-0.5)^2$ and the entropy loss $p\log p + (1-p)\log(1-p)$. We found these losses to be helpful in preliminary experiments in the linear probing setting, but they generally performed less well than the confidence auxiliary loss in Equation 1 in the finetuning setting. We have also observed negative results with the confidence losses when the training data is highly class-imbalanced or when we do not use the rebalancing procedure described in Section 4.3.

Product confidence loss. We also tried a confidence-like loss which sets the cross entropy targets to be proportional to the product of the probabilities that the weak and strong models assign, renormalized across classes and without propagating gradients through the targets. In preliminary experiments, this loss consistently gave positive results over the baseline on two NLP tasks, but performed poorly compared to our main confidence loss. Variants like geometric mean instead of product gave no boost. Compared to the confidence loss, it could be useful as it has no inter-batch dependence and could potentially be adapted for generative tasks.

LP-FT. We used the LP-FT technique proposed in Kumar et al. (2022) which first trains a linear probe on frozen strong model representations and then finetunes all layers, to avoid destroying the pretrained representation. We were unable to get improvements compared to the finetuning baseline.

Weight regularization. To regularize the strong model weights to avoid imitating the weak labels[8], we tried a variety of regularization techniques for strong model training, including stronger weight decay (Krogh & Hertz, 1991) and dropout (Srivastava et al., 2014). We did not find significant improvement.

LoRA. As another regularization technique, we also considered low-rank adaptation (LoRA) (Hu et al., 2022), i.e. only making a low-rank update to the parameters of each layer of the model during finetuning. We did not find any improvement, even when sweeping the LoRA rank.

Data augmentation. Inspired by the success of consistency algorithms in self-supervised training (Chen et al., 2020a; Caron et al., 2021), we used the strong student models to rephrase the inputs in each sample, and added an auxiliary loss enforcing the strong model predictions to be consistent between original and rephrased samples. We did not find any improvement on a selected subset of NLP datasets.

[8]However, as we discuss in Section 5.1.3, in our setup the strong model tends to be bad at imitating the weak labels. Therefore, regularization could be more important in settings where the strong model can fit the weak labels well.

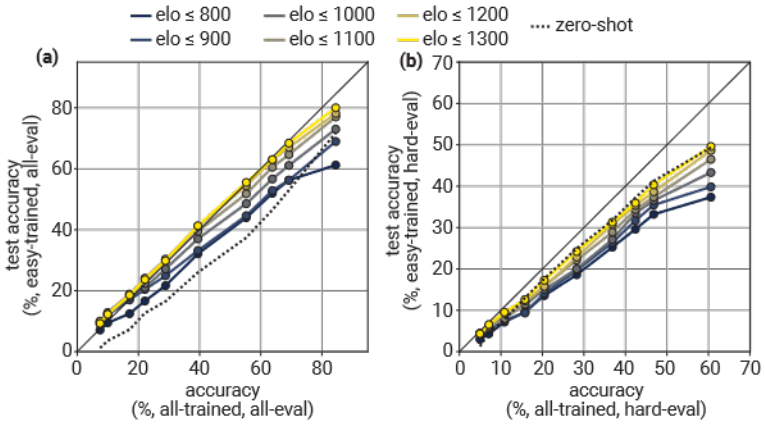


Figure 19: **Easy-to-hard generalization on chess puzzles.** We finetune models on chess puzzles with Elo $\leq t$, varying the threshold t, and evaluate the finetuned models on **(a)**: all test puzzles, and **(b)**: hard test puzzles with Elo ≥ 2000. Across the board, we see strong performance, even when training only on very easy puzzles (Elo ≤ 800). For reference, we also include the zero-shot performance of the model. Finetuning on easy puzzles, we improve upon the performance on average on the test set, but we do not improve on hard puzzles, compared to the zero-shot model.

Adding label noise, special losses for noisy labels. We experimented with the generalized cross-entropy loss proposed in Zhang & Sabuncu (2018) that is more robust to label noise, but did not find improvement over cross-entropy. We also tried adding random noise to weak labels, and found that the strong models were able to simulate the weak labels less well, especially early in training, but it did not ultimately result in improved performance.

Few-shot prompting. As an alternative to fine-tuning, we can use the in-context learning ability of the strong student models. For each task, we append a custom prompt shown in Table 2. For a detailed description of the results, see Section 5.2.1.

Weight averaging. Prior work (Izmailov et al., 2018; Cha et al., 2021; Wortsman et al., 2022b;a) suggested that various forms of weight averaging can substantially improve performance, especially in distribution shift settings. In our setup, we experimented with applying exponential moving averaging to the parameters of the model during training, but did not observe improvements relative to the baseline.

C EASY-TO-HARD GENERALIZATION

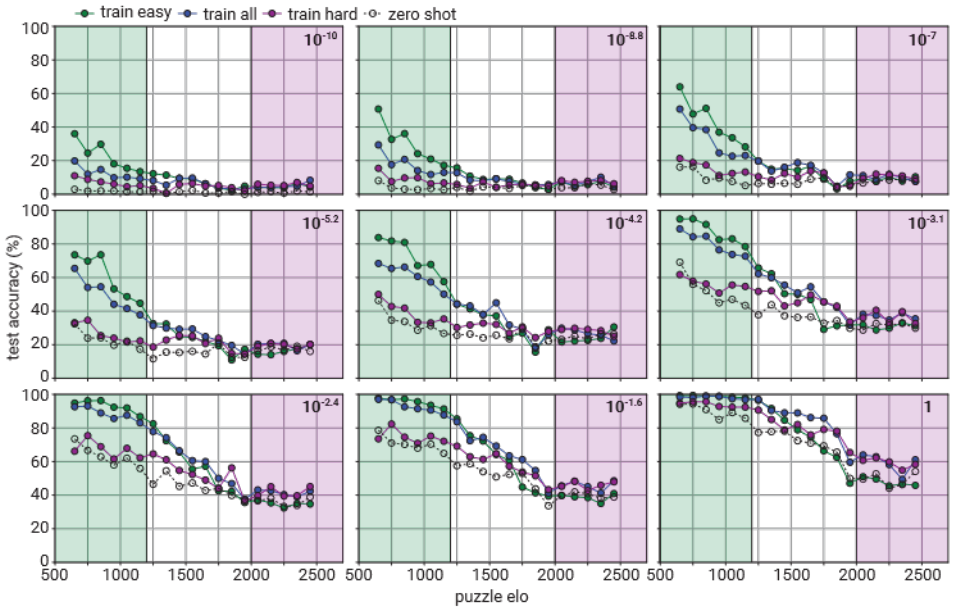
In Section 5.1.3 and Appendix E, we discuss that one reason weak-to-strong generalization may be difficult is if the weak labels have systematic errors that the strong model can learn to emulate. One natural type of systematic weak label error is to do poorly on hard examples and well on easy examples.

In this section, we focus on studying what we call **easy-to-hard generalization**, where we train only on easy examples using ground truth supervision, and assess generalization to harder examples.

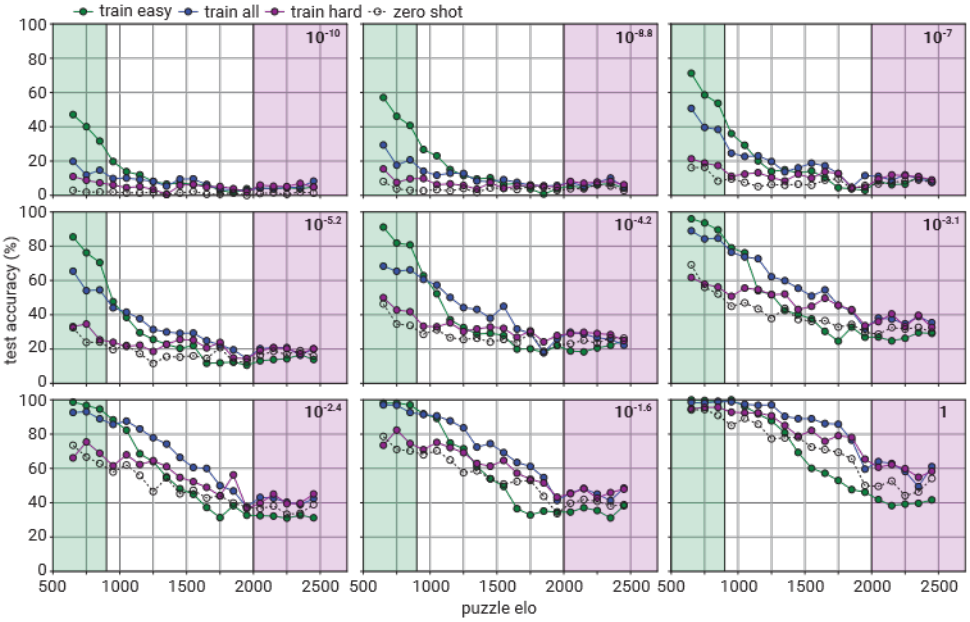
C.1 CHESS PUZZLES

Each chess puzzle comes with a natural difficulty label: an Elo score, which describes its difficulty according to humans. On the `https://lichess.org` website, people try to solve puzzles, which can be viewed as a game between a puzzle and a human player. The Elo scores are then assigned to both human players and chess puzzles following the standard Elo algorithm.

We consider the easy-to-hard generalization problem, where the difficulty is defined according to the puzzle Elo rating. We note that the puzzle Elo describes the difficulty of the entire puzzle move sequence, while we are only training the model to predict the first move in the sequence



(a) Easy cutoff: Elo $\leq$ 1200



(b) Easy cutoff: Elo $\leq$ 900

Figure 20: **Easy-to-hard generalization on chess puzzles.** We present detailed performance of models finetuned on different subsets of chess puzzles across model sizes and test puzzle difficulty levels. For each model size, we compare models trained only on easy puzzles, hard puzzles, or all puzzles. We also include the zero-shot model performance. We provide results for the easy puzzle Elo cutoffs of **(a)**: 1200 and **(b)**: 900. All finetuned models are trained on $50k$ random datapoints from the corresponding distribution. The size of the model is shown in the upper-right corner of each panel, in terms of fraction of GPT-4 compute.

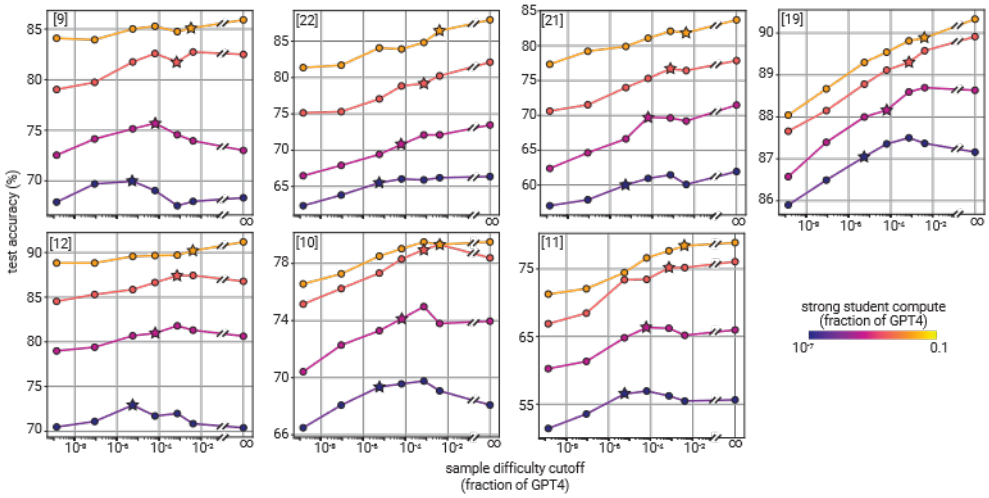


Figure 21: **Effect of varying training data difficulty on test set accuracy.** Test accuracy as a function of sample difficulty cutoff on a subset of our NLP tasks. The leftmost point on the horizontal axis corresponds to only using datapoints that models of all sizes that we consider get right when trained on other data sampled from the same task, and the rightmost point (denoted with ∞) corresponds to training on all datapoints; the point with value x on the horizontal axis corresponds to only using the datapoints that models with x or higher compute (fraction of GPT-4) consistently get right. Inset numbers indicate task id (compare Figure 12). Hue indicates compute of weak supervision. Stars indicate points where weak supervisor size corresponds to sample difficulty cutoff.

(see Appendix A.2). Consequently, the puzzle Elo is a high-quality but still imperfect measure of difficulty of the problem for humans. It is also important to note, that puzzle Elo may not be a good measure of difficulty for the models: easy puzzles for humans can be hard for the models and vice versa.

We then split the dataset into subsets according to the puzzle Elo. We consider the hard set to be puzzles with difficulty above Elo 2000. For the easy set, we consider cuttoffs in $\{800, 900, 1000, 1100, 1200, 1300\}$, and use puzzles with difficulty below the cutoff. We also consider the unrestricted set of *all* puzzles. We sample $50k$ puzzles from each of these sets randomly, and finetune the model on them[9].

We report the results in Figure 19, where we also provide the performance of a zero-shot baseline for reference. We plot the accuracy of the models trained on the easy subsets of puzzles against the performance of the same model trained on all puzzles. We find that the models generally perform well on average on the test set in panel (a), and outperform the zero-shot baseline. Interestingly, when evaluated on hard examples only, in panel (b), the models perform similarly to the zero-shot baseline, or slightly worse.

When trained on easy puzzles, the models shift towards performing well on the easy puzzles, and underperform on the hard puzzles. In Figure 20, we can see that generally the models improve upon the zero-shot baseline outside of their training difficulty range, often up to Elo of 1500 or higher, but underperform on the hardest examples.

C.2 NLP TASKS: DIFFICULTY THRESHOLDING

NLP tasks do not come with a natural source of difficulty labels, but we can create such labels by looking at performance as a function of model size.

[9]For easy puzzles with 800-Elo cutoff, we only use $25k$ puzzles, because there are not $50k$ puzzles available in this difficulty range.

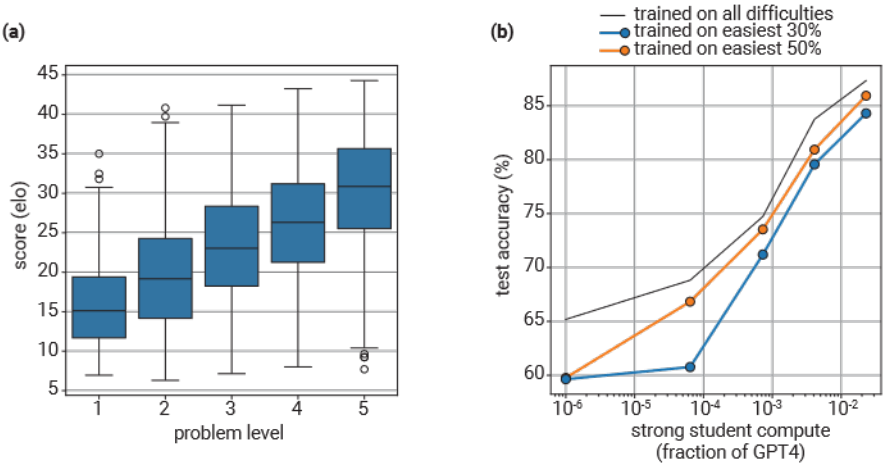


Figure 22: **Filtering training samples by GPT-4 generated Elo scores results in very good easy-to-hard generalization.** **(a)** GPT-4 generated Elo scores for different, human-defined, problem difficulties (1 - easiest, 5 - hardest) on the MATH dataset. **(b)** Average test accuracy as a function of strong student compute on a subset of our NLP tasks. Student is trained on ground truth labels on samples of all difficulties (black), only the 30% easiest tasks (orange), or only the 50% easiest tasks (blue).

We define *difficulty* of a datapoint based on the smallest model size that consistently predicts the label on this datapoint correctly, when trained on ground truth. For example, suppose we have 4 ground truth models W_1, W_2, W_3, W_4 that use compute $C_1 < C_2 < C_3 < C_4$ respectively. Suppose models W_1, W_3, W_4 predict the example correctly when it is in a held-out set, while W_2 predicts it incorrectly. Then we will assign a difficulty of C_3 to the example.

Then given a difficulty cutoff D, we filter the training set to examples with difficulty $\leq D$. We subsample the filtered set so that the number of training examples is equal to the number of examples at the lowest difficulty level. We train a model on the subsampled training set using ground truth labels, and measure its accuracy on a held out test set (with no subsampling).

The subsampling ensures that we use the same training set size for each difficulty cutoff. Using ground truth labels ensures that the label accuracy is the same (100%) for each cutoff. We also use the same test set for each cutoff. This setup lets us vary only training data difficulty, and measure its impact on the trained model's accuracy.

We plot results in Figure 21. The y-axis is accuracy on the test set, while the x-axis is the difficulty cutoff. Increasing the difficulty cutoff generally leads to an increase in accuracy. This result suggests that solving easy-to-hard generalization is non-trivial even if there are no weak label errors.

For smaller models (darker lines), the accuracy initially increases, but starts to decrease beyond a point. The drop generally happens when the difficulty cutoff exceeds the capacity of the model itself, i.e. when the examples are too difficult for the model to fit. However, large models trained on easy examples often perform well.

C.3 GPT-4 PREDICTED DIFFICULTY

Ultimately, we care about strong models generalizing from human supervision. From this perspective, it is important to understand whether we can achieve easy-to-hard generalization, where the difficulty is measured according to humans, rather than capacity-constrained models. In Appendix C.1, we explored this question in chess, but we would want to extend this analysis to the NLP tasks.

Most natural datasets do not come with information about problem difficulty. As a rough estimate, we automatically generated difficulty labels using GPT-4. More concretely, we used GPT-4 to rank pairs of examples in each dataset, asking "which question is easier, Question A or Question B?" We then calculated the Elo scores for each example via a finite number of random comparisons.

Table 3: **Weak-to-strong generalization on ImageNet**. We train linear probes on the representations extracted by DINO models with weak supervision from an AlexNet model. The strong students substantially outperform their weak supervisor.

Model	Top-1 Accuracy (%)	PGR (%)
AlexNet (weak supervisor)	56.6	-
Dino ResNet50	63.7	-
Dino ViT-B/8	74.9	-
AlexNet $\rightarrow$ DINO ResNet50	60.7	57.8
AlexNet $\rightarrow$ DINO ViT-B/8	64.2	41.5

To evaluate the quality of GPT-4 Elo score as a measure of difficulty, we performed correlation analysis against human annotations for datasets with human difficulty levels such as MATH (Hendrycks et al., 2021) and chess, as well as against weak model confidence. We found that the three measures align better for reasoning tasks such as MATH, as we show in Figure 22(a), but not much for some natural language tasks. When looking at the samples, we found that GPT-4 Elo scores tend to be higher for longer questions, but those questions may actually be easy for smaller models since they provide more context.

Using GPT-4 Elo score as a proxy for human difficulty, we used different cutoffs on scores to separate easy and hard examples, trained the strong models on the easy examples only (with ground truth labels), and evaluated on the hard examples. Preliminary results are shown in Figure 22(b).

In general, we found that using GPT-4 Elo as measure of hardness makes generalization slopes steeper than our main setup of weak-to-strong generalization. One possible confounder for interpretation is that our Elo measurements could be noisy, causing generalization to be better.

Note that this setup is a classic covariate shift problem, whereas our main setup focuses more on concept shift and noisy labels. It is unclear which setup would be more relevant, and we think it is important to study easy-to-hard generalization more thoroughly in future work.

D Other weak-to-strong settings

D.1 Self-supervised vision models

We additionally demonstrate weak-to-strong generalization in a simple image classification experiment. We use a pretrained AlexNet model (Krizhevsky et al., 2012) as a weak supervisor, and use it to generate weak labels on the ImageNet (Russakovsky et al., 2015) validation set. As a strong student, we use linear probing on frozen representations extracted by DINO models (Caron et al., 2021) based on ResNet-50 (He et al., 2016) and ViT-B/8 (Dosovitskiy et al., 2020) architectures. The DINO models are pretrained in an unsupervised way and did not observe direct supervision for ImageNet classification or any other classification task during pretraining, so this experiment does not have the pretraining leakage disanalogy discussed in Section 6.1.

We use $40k$ datapoints from the validation set to train the linear probes, and evaluate performance on the remaining $10k$ datapoints. For training the linear probes, we use a batch size of 128, Adam optimizer (Kingma & Ba, 2014) and a learning rate of 10^{-3}. We run 20 epochs of training for ResNet-50 and 5 epochs for ViT-B/8.

We report the results in Table 3. Similarly to our main experiments in Section 4, the student can substantially outperform the supervisor, achieving PGR on the order of 50%. This experiment shows that our results are not limited to the natural language setting, and generalize to other domains. It also shows that strong students can generalize from weak supervision on tasks where they only had indirect pretraining, i.e. where the knowledge of the task is latent.