Alex Turner, Neale Ratzlaff, and Prasad Tadepalli. Avoiding side effects in complex environments. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21406–21415. Curran Associates, Inc., 2020.

Kailas Vodrahalli, Tobias Gerstenberg, and James Zou. Uncalibrated models can improve human-ai collaboration. *ArXiv*, abs/2202.05983, 2022.

Carroll L Wainwright and Peter Eckersley. Safelife 1.0: Exploring side effects in complex environments. *arXiv preprint arXiv:1912.01217*, 2019.

Hai Wang and David McAllester. On-the-fly information retrieval augmentation for language models. *arXiv preprint arXiv:2007.01528*, 2020.

Richard Webby and Marcus O'Connor. Judgemental and statistical time series forecasting: a review of the literature. *International Journal of forecasting*, 12(1):91–118, 1996.

Shengjia Zhao, Tengyu Ma, and Stefano Ermon. Individual calibration with randomized forecasting. In *International Conference on Machine Learning*, pages 11387–11397. PMLR, 2020.

A Additional Experimental Details and Results

A.1 Autocast Experiments

Calibration Results. In Figure 8, we show the adaptive binning calibration curve for crowd forecasts on all resolved true/false questions by plotting the fraction of positives against the model's predicted probability for the positive class.

Additionally, we can compare the calibration of our static and temporal models to crowd performance on the resolved test set. Treating true/false questions as two-class classification problems and combining them with multiple-choice questions, we calculate adaptive binning calibration error with a bin size of 50 samples. The largest FiD Static model incurs a 40% calibration error while the human crowd incurs a much smaller 8% calibration error. By leveraging crowd predictions in our FiD Temporal models, we reduce the calibration error to 17%, showing potential for improvements.

Model and Training Loss. The FiD Temporal model uses three separate linear heads after its hidden state outputs to answer each type of questions (true/false, multiple-choice, and numerical). In particular, the multiple-choice head has 12 outputs which is the maximum number of choices in the training set. Additionally, the original input embeddings are replaced with a linear layer to map from the FiD Static's hidden states to the GPT-2's hidden states. Finally, to make training more stable, we average the loss over the sequence of predictions for each question to weigh the questions evenly. Moreover, the losses of the three types of questions are normalized by their respective baseline loss (uniformly random predictions) before summing together so that their losses are on the same scale.

Retrieval from CC-NEWS. Given a question, for each day the question is active, we retrieve the top 10 relevant news articles from the daily articles. In our FiD-Temporal experiments, we only use the top 1 from every day. Then, we aggregate all these articles from different dates and rank them according to the retrieval score. The top 10 articles are used for the FiD-Static model. We follow the Terms of Use for the Common Crawl website. The dataset is fully reproducible with the script to download and filter CC-NEWS on GitHub.

A.2 Confidence Intervals

Interval Construction. In the reference implementation of the get_confidence_intervals function in Figure 7, we construct our intervals by first producing a point estimate for each question and iteratively adding on non-negative, non-symmetric deltas on both sides, so that the intervals become nested and wider for higher confidence levels.

Training Loss for Baseline. First, because the labels span a large numerical range, we normalize them by taking the log. Then, we construct a loss with three components shown in Figure 7: (1) $\mathcal{L}_p$: MSE loss between the predicted point estimate and the ground-truth target, (2) $\mathcal{L}_b$: MSE loss between the boundaries of the predicted confidence intervals and the ground-truth target for boundaries that are on the wrong side of the target, (3) $\mathcal{L}_i$: a penalty on the length of the predicted intervals to encourage finer predictions. Based on whether the ratio of true labels contained in the predicted intervals is higher than the target confidence level, we either activate the boundary loss $\mathcal{L}_b$ or the interval length loss $\mathcal{L}_i$ for that particular confidence level output. Lastly, the three components are weighted by coefficients $1, 1, 0.01$ chosen with a simple search using the validation set.

	Resolved	Unresolved	Total
Train	2815 4411	1375 1974	4190 6385
Test	907 1292	1610 2305	2517 3597
Total	3722 5703	2985 4279	6707 9982

Table 4: The number of forecasting questions in Autocast. In total, there are nearly 10,000 questions. Gray text indicates the number of questions after augmenting true/false questions with their negations, a procedure we use to balance the dataset.

Adaptive RMS Metric. An important task for numerical forecasting is outputting calibrated uncertainty estimates. However, a unique challenge in this setting is that answers can vary across many orders of magnitude. To evaluate the calibration of confidence intervals across a large dynamic range of output values, we design a specialized local calibration metric (Zhao et al., 2020; Kull et al.,

```
Is = [0.5, 0.55, ..., 0.95]
num_intervals = len(Is)

def low_containment_mask(lowers, uppers, labels, Is):
    # lowers, uppers: Predicted lower and upper bounds of intervals
    # Is: Target confidence levels
    # Returns: A list of boolean values indicating which confidence level
    #          has containment ratio below the target level within batch
    contained = (lowers <= labels) * (labels <= uppers)
    ratio_contained = contained.mean(dim=0)
    return ratio_contained < Is

def get_confidence_intervals(logits):
    # logits: Model output with (2 * num_intervals + 1) neurons
    deltas, point_estimates = softplus(logits[:, :-1]), logits[:, -1:]
    lower_deltas = deltas[:, :num_intervals]
    higher_deltas = deltas[:, num_intervals:]
    interval_lengths = lower_deltas + higher_deltas
    # custom cumsum with gradients accumulated once on each delta
    lower_deltas = utils.cumsum(lower_deltas)
    higher_deltas = utils.cumsum(higher_deltas)
    lowers = point_estimates - lower_deltas
    uppers = point_estimates + higher_deltas

    return lowers, uppers, point_estimates, interval_lengths

out = get_confidence_intervals(logits)
lowers, uppers, point_estimates, interval_lengths = out

$\mathcal{L}_p$ = MSE(point_estimates, labels)
l_mask = lowers > labels
u_mask = uppers < labels
$\mathcal{L}_b$ = MSE(lowers, labels) * l_mask + MSE(uppers, labels) * u_mask
$\mathcal{L}_i$ = MSE(interval_lengths, 0)
# normalize loss by the label magnitude, adjusting for small labels
$\mathcal{L}_p$ /= (1 + abs(labels))
$\mathcal{L}_b$ /= (1 + abs(labels))
$\mathcal{L}_i$ /= (1 + abs(labels))
# activate loss for particular confidence levels based on ci_mask
ci_mask = low_containment_mask(lowers, uppers, labels, Is)
$\mathcal{L}_b$ = $\mathcal{L}_b$.mean(dim=0) * ci_mask
$\mathcal{L}_i$ = $\mathcal{L}_i$.mean(dim=0) * (1 - ci_mask)

$\alpha$, $\beta$, $\gamma$ = 1, 1, 0.01 # hyperparameters
loss = $\alpha$ * $\mathcal{L}_p$.mean() + $\beta$ * $\mathcal{L}_b$.mean() + $\gamma$ * $\mathcal{L}_i$.mean()
```

Figure 7: A reference implementation of the baseline training loss for outputting calibrated confidence intervals. For the confidence levels where too few true labels fall inside the predicted intervals, we encourage the model to adjust its boundaries through boundary loss $\mathcal{L}_b$. Conversely, we encourage the model to shrink the predicted intervals if too many fall inside the predicted intervals.

2019a), shown in Algorithm 1. First, test examples are sorted by their target value and split into bins with a fixed number of examples each (adaptive binning). Then, we calculate calibration error across all bins using a Euclidean norm (Hendrycks et al., 2019). Finally, we average this local calibration error across all confidence levels, giving the final metric. For brevity, we refer to this overall metric as RMS Calibration Error. A low value for this error metric indicates that models are calibrated across their entire dynamic range of output values.