Similarity-Distance-Magnitude Universal Verification

Allen Schmaltz ALLEN@RE.EXPRESS
Reexpress AI, Inc., USA

Abstract

We address the neural network robustness problem by adding SIMILARITY (i.e., correctly predicted depth-matches into training)-awareness and DISTANCE-to-training-distribution-awareness to the existing output MAGNITUDE (i.e., decision-boundary)-awareness of the softmax function. The resulting SDM activation function provides strong signals of the relative epistemic (reducible) predictive uncertainty. We use this novel behavior to further address the complementary HCI problem of mapping the output to human-interpretable summary statistics over relevant partitions of a held-out calibration set. Estimates of prediction-conditional uncertainty are obtained via a parsimonious learned transform over the class-conditional empirical CDFs of the output of a final-layer SDM activation function. For decision-making and as an intrinsic model check, estimates of class-conditional accuracy are obtained by further partitioning the high-probability regions of this calibrated output into class-conditional, region-specific CDFs. The uncertainty estimates from SDM calibration are remarkably robust to test-time distribution shifts and out-of-distribution inputs; incorporate awareness of the effective sample size; provide estimates of uncertainty from the learning and data splitting processes; and are well-suited for selective classification and conditional branching for additional test-time compute based on the predictive uncertainty, as for selective LLM generation, routing, and composition over multiple models and retrieval. Finally, we construct SDM networks, LLMs with uncertainty-aware verification and interpretability-by-exemplar as intrinsic properties. We provide open-source software implementing these results.[1]

Keywords: Epistemic uncertainty quantification, Calibration, Interpretability, Out-of-distribution detection, Large language model fine-tuning

1 Introduction

Large language models (LLMs) pose a challenge for interpretable and reliable deployment given the non-identifiability of their parameters (Hwang and Ding, 1997, inter alia)[2], which can number in the billions or more. Instead of directly interpreting parameters, instance-based, metric-learner approximations and hard-attention mechanisms can be constructed with task-specific inductive biases for effective semi-supervised learning (i.e., feature detection) and introspection against the training set (Schmaltz, 2021), which can be useful for auditing predictions as a form of interpretability by example, or *exemplar*, over the representation space of the model. However, for real-world deployments, robust approaches for predictive uncertainty—and relatedly, for verifying the modeling process—are also needed, both for human decision-making and for constructing sequentially dependent LLM pipelines.

[1] `https://github.com/ReexpressAI/sdm`

[2] Informally, this means that two or more distinct sets of values for the parameters can result in identical output distributions. As a consequence, interpreting the parameters of such models is typically much more complicated than with a simple linear regression model, for example.

Known theoretical results limit the statistical quantities that can be derived over LLMs. Statistical assurances in the distribution-free setting are limited to approximately conditional quantities (Valiant, 1984; Lei and Wasserman, 2014; Foygel Barber et al., 2020, inter alia). Further, even typical approximately conditional quantities can be difficult to obtain in practice, since the minimal assumption of exchangeability with a known held-out data set is itself often violated with co-variate and label shifts, which can be difficult to foresee with existing methods. Epistemologically, the prevalence of hallucinations and highly-confident wrong answers with widely deployed LLMs suggests a technical impasse in effectively modeling the predictive uncertainty, despite significant work from Bayesian, Frequentist, and empirically motivated perspectives (Gal and Ghahramani, 2016; Angelopoulos et al., 2021; Guo et al., 2017; Lakshminarayanan et al., 2017; Ovadia et al., 2019, inter alia). A foundational piece is evidently missing from the picture.

Given these intrinsic challenges, we approach the problem of uncertainty quantification over LLMs from a new angle and ask: *Can we leverage the metric learning and dense matching capabilities of neural networks over high-dimensional inputs to at least aim to maximize, with minimal distributional assumptions, the separation of aleatoric (irreducible) uncertainty and epistemic (reducible) uncertainty, decomposing the sources of the latter in a manner that is interpretable and actionable?*

We answer this question in the affirmative with a conceptually parsimonious, LLM-driven partitioning of the data to decompose sources of epistemic uncertainty: Correctly predicted depth-matches into the training set (Similarity), the Distance to the training set, and the distance to the decision-boundary (Magnitude). We use these signals to construct a new activation function, the sdm activation, which replaces a foundational building block of contemporary AI, the softmax operation. A series of distributional transforms over an sdm activation then enable us to directly target a quantity of interest, *index-conditional calibration*, well-suited for selective classification (Chow, 1957; Geifman and El-Yaniv, 2017, inter alia), which reflects the typical need for uncertainty quantification with LLMs as part of multi-stage decision pipelines. Finally, with this new foundational behavior, we construct a new LLM architecture, the sdm network, with an intrinsic—and externally human interpretable—capability to verify its own instruction-following.

In summary, in this work:

- We introduce the sdm activation function, which encodes strong signals of epistemic uncertainty, to replace the softmax operation.
- We provide a robust estimator of index-conditional uncertainty (Def. 3) via a final-layer sdm activation over existing models.
- We propose the sdm network, a new LLM architecture and fine-tuning approach for which uncertainty-awareness and interpretability-by-exemplar are intrinsic properties.
- We empirically compare the uncertainty-awareness of our estimators to existing classes of approaches, which we demonstrate do not reliably achieve our desired uncertainty quantity in the presence of—even modest—distribution shifts. As a natural, held-out blind evaluation, we also demonstrate efficiently uncovering undetected annotation errors in the carefully curated MMLU-Pro benchmark dataset. This reflects the estimator's capacity to separate aleatoric and epistemic uncertainty.

- More broadly, the methods and results in this work provide a new understanding of the behavior of neural networks, demonstrating that there are regions of the output distribution that are low variation and high-probability that can be reliably detected. This newfound ability to control for the epistemic uncertainty in high-dimensions is a substantive departure from existing estimators, which marginalize over the distinctions across these regions, which can contribute to unexpected LLM behavior at test-time.

2 Motivation

Given the ability of LLM's to recursively cross-encode data, user instructions, and outputs, if we had a reliable means of assessing the uncertainty over an LLM's predictions that was also human interpretable (i.e., a quantifiable and verifiable assurance in their instruction-following abilities), such an LLM could serve as a *universal verifier* over existing models, which would in effect calibrate the predictive uncertainty of other models: Have, e.g., an exogenous regression or multi-label model? Simply cross-encode the data, exogenous model, and output as input to the LLM verifier and let the neural network generate the accuracy as to whether the exogenous model is correct or not. This process could be repeated, as needed, using such an LLM as a basis for building complex, compound AI systems, recursively cross-encoding the input and output, using the uncertainty over discrete predictions as the branching condition for additional test-time compute, tool calling, and human feedback — and ultimately, reliable AI-assisted decision-making. In this work, we introduce the mechanisms for constructing such a verifier, which we formalize below.

3 Preliminaries

3.1 Setting

Both LLM next-token prediction and standard classification tasks (e.g., predicting the sentiment of a movie review) are formulated similarly as predictions over discrete classes. We are given a training dataset, $\mathcal{D}_{\text{tr}} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$ of inputs, $\boldsymbol{x} \in \mathcal{X}$, paired with their corresponding ground-truth discrete labels, $y \in \mathcal{Y} = \{1, \ldots, C\}$, and a labeled calibration dataset, $\mathcal{D}_{\text{ca}}$, drawn from the same distribution as $\mathcal{D}_{\text{tr}}$. We are then given a new test instance, $\boldsymbol{x}$, from an unlabeled test set, $\mathcal{D}_{\text{te}}$, and seek to estimate the label with a prediction, $\hat{y}$, via the un-normalized log probabilities ("logits", informally) of a final linear layer: $\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{h} + \boldsymbol{b}$, where $\boldsymbol{h} = \text{network}(\boldsymbol{x}; \theta)$ is the final hidden state of a network parameterized by θ. The network can be recurrent (Hochreiter and Schmidhuber, 1997), convolutional (Dauphin et al., 2017), or self-attention-based (Devlin et al., 2019), among others. The discrete prediction is taken as $\hat{y} = \arg\max \boldsymbol{z}$; however, for learning θ, $\boldsymbol{W}$, and $\boldsymbol{b}$, and for human decision-making, we also seek an estimate of the predictive uncertainty, $p(y \,|\, \boldsymbol{x})$, which is typically obtained by normalizing $\boldsymbol{z}$ via the softmax operation described next. We will make a distinction between models, $\mathcal{M}$ (defined by θ, $\boldsymbol{W}$, and $\boldsymbol{b}$, and when applicable, the exemplar adaptor, described below), which produce the prediction, $\hat{y}$, and estimators, $\mathcal{E}$, which provide an estimate of $p(y \,|\, \boldsymbol{x})$, because different estimators can be used over the same model.

3.2 Softmax and the Cross-Entropy loss

The softmax has as its origins the work of L. Boltzmann in the 19th century (see Sharp and Matschinsky, 2015). It remains a central function in the natural and engineering sciences. It is ubiquitous in deep learning, playing an integral role as a router in self-attention mechanisms (Vaswani et al., 2017) and mixture-of-experts models (Shazeer et al., 2017); forming the basis of the cross-entropy loss used for next-token training of LLMs; and serving as the final interface between a model and the end-user, converting the un-normalized model logits to human interpretable probability distributions, at least in principle:

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{\tau \cdot z_i}}{\sum_{c=1}^{C} e^{\tau \cdot z_c}}, 1 \leq i \leq C, \tau \geq 0 \tag{1}$$

The above function induces a parameterization of the event probabilities of a categorical distribution:

$$\text{Categorical}(C = |\mathcal{Y}|, \text{softmax}(\boldsymbol{z})) \tag{2}$$

The inverse-temperature parameter, τ, controls the sharpness of the distribution. As $\tau \rightarrow 0$, the output of softmax($\boldsymbol{z}$) converges to a uniform distribution where each class has probability $\frac{1}{C}$; as $\tau \rightarrow \infty$, the output converges to a distribution in which all of the mass is assigned to a single class. In deep learning, τ is treated as a learnable, *global* hyper-parameter; *instance-wise* variation in the distance to the decision-boundary is thus determined by the relative Magnitude of $z_{\hat{y}}$. This model is learned by minimizing the cross-entropy loss between $\boldsymbol{z}$ and the index of the true labels over $\mathcal{D}_{\text{tr}}$. The *natural* logarithm of the loss is the counterpart to the base e of the softmax:

$$\mathcal{L}(\theta, \boldsymbol{W}, \boldsymbol{b}; \mathcal{D}_{\text{tr}}) = -\frac{1}{N}\sum_{n}^{N} \log_e \left(\frac{e^{\tau \cdot z_{y_n}}}{\sum_{c=1}^{C} e^{\tau \cdot z_c}} \right) \tag{3}$$

4 Methods

In this work, we revisit Eq. 1, 2, and 3 given new observations on the statistical behavior of high-dimensional objects, empirically derived from large parameter neural networks. We will seek to decouple the sources of epistemic uncertainty via a new activation function that is conceptually:

$$\text{sdm}(\boldsymbol{z})_i = \frac{\text{Similarity}^{\text{Distance} \cdot \text{Magnitude}_i}}{\sum_{c=1}^{C} \text{Similarity}^{\text{Distance} \cdot \text{Magnitude}_c}} \tag{4}$$

with a corresponding negative log likelihood loss that takes into account the change of base (§ 4.1). We will additionally introduce a transformation that rescales this value for an *instance* with exogenous information *across* $\mathcal{D}_{\text{ca}}$, effectively calibrating (Brier, 1950; Dawid, 1982) the model to produce reliable, interpretable probabilities (§ 4.2). Finally, we integrate this behavior into the LLM architecture and training, yielding an LLM with an intrinsic ability to verify its own instruction following (§ 4.3), as illustrated in Figure 1.

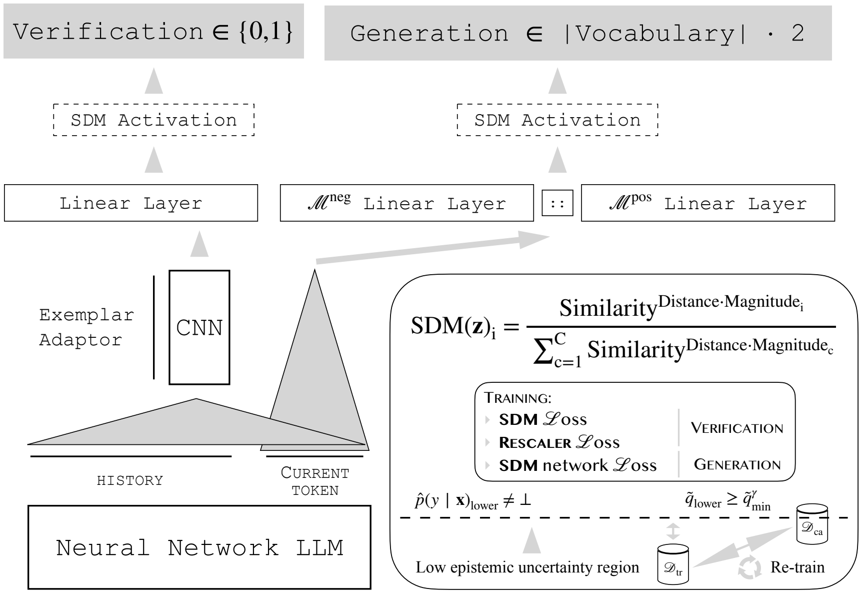
Verification ∈ {0,1}
Generation ∈ |Vocabulary| · 2
SDM Activation
SDM Activation
Linear Layer
$\mathscr{M}^{\text{neg}}$ Linear Layer
::
$\mathscr{M}^{\text{pos}}$ Linear Layer
Exemplar Adaptor
CNN
$\text{SDM}(\mathbf{z})_{\text{i}} = \frac{\text{Similarity}^{\text{Distance}\cdot\text{Magnitude}_{\text{i}}}}{\sum_{\text{c}=1}^{\text{C}} \text{Similarity}^{\text{Distance}\cdot\text{Magnitude}_{\text{c}}}}$
Training:
SDM $\mathscr{L}$oss
Rescaler $\mathscr{L}$oss
SDM network $\mathscr{L}$oss
Verification
Generation
history
Current token
$\hat{p}(y \mid \mathbf{x})_{\text{lower}} \neq \bot$
$\tilde{q}_{\text{lower}} \geq \tilde{q}^{\gamma}_{\text{min}}$
$\mathscr{D}_{\text{ca}}$
Neural Network LLM
Low epistemic uncertainty region
$\mathscr{D}_{\text{tr}}$
Re-train

Figure 1: SDM networks are uncertainty-aware via a robust estimator of index-conditional calibration (Def. 3), $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$, over output verification (i.e., binary classification of instruction-following); intrinsically introspectable via depth-matching into a training set ($\mathcal{D}_{\text{tr}}$) and correspondence to comparable points in a held-out calibration set ($\mathcal{D}_{\text{ca}}$) via $\lfloor\tilde{q}\rfloor$, which is a stable mapping and summary of the epistemic uncertainty signals of Similarity, Distance, and Magnitude; and updatable via a fine-tuning process to maximize the proportion of verifiable high-probability generations. Decoding proceeds by generating from the distribution of $\text{SDM}(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})$ up to a control token at the unit-of-analysis of the verification labels. Decoding then continues, or other branching actions are taken, based on $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$.

4.1 From Model Approximations via Exemplar Adaptors to SDM Activation Functions

Exemplar adaptors, 1-D CNN adaptors (with a final linear layer) over the frozen hidden states of a network, induce distilled, compressed representations of an underlying network's representation space conditional on its predictions. This behavior can be used to faithfully approximate a model's predictions as a mapping against a training, or support, set. This can be achieved, for example, with instance-based, metric-learning estimators, such as weighted KNNs, where the weights are learned as a transform of the exemplar adaptor's distilled representations.[3] Critically, when the approximations diverge from the predictions of the underlying model, the inputs tend to be from the subsets of the distribution over which the underlying model is itself unreliable (Schmaltz, 2021). In other words, the

[3] Such instance-based, metric-learner approximations of neural networks differ from traditional KNN rules (Cover and Hart, 1967; Devroye et al., 1996, inter alia) in two critical respects: The neural network serves as a semi-supervised learner of the distances between the dense representations that identify the instances, and there is a model prediction (in addition to the ground-truth label) for each instance in the support set. The former enables effective partitioning despite the curse of high dimensions; the latter provides an additional indicator of reliability for each instance.

approximations encode strong signals of the epistemic uncertainty, a point under-appreciated in the existing literature and which we bring to its logical conclusion in this work. Rather than constructing explicit KNN approximations, which require a separate training step and additional parameters, we instead **q**uantize the closeness of a point to the training set with a discrete estimate. Further, we transform the distance to the closest match as a quantile estimate over the distribution of distances. These quantities, combined with the output Magnitude, capture the key sources of epistemic uncertainty for an input *instance* (cf. § 4.2).

4.1.1 Exemplar Adaptor

We take as the CNN of our exemplar adaptor $g : (\boldsymbol{h}, t(\boldsymbol{z})) \in \mathbb{R}^D \mapsto \boldsymbol{h'} \in \mathbb{R}^M$, a 1-D CNN that takes as input h (if available) of the underlying network and optionally, the concatenation of the output of $t(\boldsymbol{z})$, a transform of the underlying network's output.[4] The CNN has M filters, the filter applications of which produce $\boldsymbol{h'}$, the distilled representation of the underlying network. A final linear layer, $\boldsymbol{z'} = \boldsymbol{W'}^T\boldsymbol{h'} + \boldsymbol{b'}, \boldsymbol{z'} \in \mathbb{R}^C$, then replaces the underlying network's linear layer, with the discrete prediction taken as $\hat{y} = \arg\max \boldsymbol{z'}$. This exemplar adaptor will then enable us to derive the key signals of epistemic uncertainty, Similarity, Distance, and Magnitude described next.

4.1.2 Similarity

We define the Similarity (q) of an instance to the training set as the count of consecutive nearest matches in $\mathcal{D}_{\text{tr}}$ that are correctly predicted *and* match $\hat{y}$ of the test instance. Concretely, we first sort $\mathcal{D}_{\text{tr}}$ (for which we have both model predictions and ground-truth labels) based on the L^2 distance (2-norm) from $\boldsymbol{h'}$, $\left[(\boldsymbol{x}^{tr}_{(1)}, \hat{y}^{tr}_{(1)}, y^{tr}_{(1)}), \ldots, (\boldsymbol{x}^{tr}_{(N)}, \hat{y}^{tr}_{(N)}, y^{tr}_{(N)})\right]$, such that $||\boldsymbol{h'} - \boldsymbol{h'}^{tr}_{(1)}||_2 \leq \ldots \leq ||\boldsymbol{h'} - \boldsymbol{h'}^{tr}_{(N)}||_2$, and then calculate $q \in \{0, \ldots, |\,\mathcal{D}_{\text{tr}}\,|\}$ as:

$$q = \sum_{i=1}^{|\,\mathcal{D}_{\text{tr}}\,|} \mathbf{1}_{\hat{y}=\hat{y}^{\text{tr}}_{(i)}} \cdot \mathbf{1}_{\hat{y}^{\text{tr}}_{(i)}=y^{\text{tr}}_{(i)}} \cdot \mathbf{1}_{i-1=\sum_{j=1}^{i-1} \mathbf{1}_{\hat{y}=\hat{y}^{\text{tr}}_{(j)}} \cdot \mathbf{1}_{\hat{y}^{\text{tr}}_{(j)}=y^{\text{tr}}_{(j)}}} \tag{5}$$

where the rightmost indicator function, $\mathbf{1} \in \{0, 1\}$, ensures consecutive (depth-wise) matches. By definition, q cannot exceed the count of the most prevalent class label in $\mathcal{D}_{\text{tr}}$, and since we assume a reasonable relative number of points for each class, $q \ll |\,\mathcal{D}_{\text{tr}}\,|$ is typical. For the special case of calculating q for $\boldsymbol{x} \in \mathcal{D}_{\text{tr}}$, which only occurs during learning, we exclude the self-match.

4.1.3 Distance

The L^2 distance to the nearest match in $\mathcal{D}_{\text{tr}}$ follows from above: $d_{\text{nearest}} = ||\boldsymbol{h'} - \boldsymbol{h'}^{tr}_{(1)}||_2$. However, it is difficult to work with d_{nearest} directly since its scale can vary widely depending on the input to g and the size of M. Instead, we define Distance, $d \in [0, 1]$, in terms of the class-wise empirical CDFs of d_{nearest} over $\mathcal{D}_{\text{ca}}$, as the most conservative quantile relative to

[4] For black-box LLM API's in particular, we will not have direct access to $\boldsymbol{h}$ and will instead construct a proxy of $\boldsymbol{h}$ via a transform t of the available output, which may (and typically will with current models) itself be the result of a softmax operation.

the distance to the nearest matches observed in the labeled, held-out set:

$$d = \min\left[1 - \text{eCDF}_{\text{ca}}^{y_1}(d_{\text{nearest}}), \ldots, 1 - \text{eCDF}_{\text{ca}}^{y_C}(d_{\text{nearest}})\right] \tag{6}$$

The empirical CDFs are determined by the labeled points in $\mathcal{D}_{\text{ca}}$ for which $q > 0$, where, as indicated by the superscripts, the stratification of points is by the true labels, y. For example, $\text{eCDF}_{\text{ca}}^{y_1}(d_{\text{nearest}})$ is the empirical CDF of d_{nearest} values in $\mathcal{D}_{\text{ca}}$ for which $y = 1$, a notation convention we will use throughout. (Points with $q = 0$ are effectively out-of-distribution points and treated as such in downstream decision-making, so they are excluded to avoid biasing the estimates.) At test-time, we do not see y; instead, the minimum is calculated over the quantiles of each of the class-conditional eCDFs, regardless of $\hat{y}$. As with q, for the special case of calculating d for $\boldsymbol{x} \in \mathcal{D}_{\text{tr}}$, we replace $\text{eCDF}_{\text{ca}}^{y_c}$ with the analogous $\text{eCDF}_{\text{tr}}^{y_c}$, the class-wise empirical CDFs of d_{nearest} over $\mathcal{D}_{\text{tr}}$ excluding self-matches.

4.1.4 Magnitude

We take as the Magnitude, or distance to the decision boundary, $z'_{\hat{y}}$, as in the standard softmax case but via $\boldsymbol{z}'$ from the linear layer of the exemplar adaptor.

4.1.5 sdm Activation: Formulation

We use the above quantities to define the sdm activation function:

$$\text{sdm}(\boldsymbol{z}')_i = \frac{(2+q)^{d \cdot z'_i}}{\sum_{c=1}^{C} (2+q)^{d \cdot z'_c}}, 1 \leq i \leq C \tag{7}$$

The output distribution becomes sharper with higher values of q, d, and z'. Also note that when d_{nearest} exceeds the largest distance observed in the labeled data, $d = 0$ and the output distribution is uniform, reflecting a maximally high (i.e., out-of-distribution) epistemic uncertainty estimate. The standard softmax with $\tau = 1$ is recovered by setting $q = e - 2, d = 1$. As with the softmax operation, $\arg\max \text{sdm}(\boldsymbol{z}') = \arg\max \boldsymbol{z}'$.

4.1.6 sdm Activation: Loss and Training

A loss analogous to Eq. 3 then follows with the applicable change of base. We use this loss to train the weights of the exemplar adaptor, which includes the parameters of the linear layer ($\boldsymbol{W}'$ and $\boldsymbol{b}'$), as well as the convolution weights and biases, which we collectively represent with $\boldsymbol{G}$. The weights of the underlying network remain fixed. (We return to training θ, $\boldsymbol{W}$, and $\boldsymbol{b}$ of an underlying LLM in § 4.3.)

$$\mathcal{L}(\boldsymbol{G}, \boldsymbol{W}', \boldsymbol{b}'; \mathcal{D}_{\text{tr}}) = -\frac{1}{N} \sum_{n}^{N} \log_{(2+q)} \left(\frac{(2+q)^{d \cdot z'_{y_n}}}{\sum_{c=1}^{C} (2+q)^{d \cdot z'_c}} \right) \tag{8}$$

Pseudo-code for training the sdm activation layer and sdm estimator (described in § 4.2, next) appears in Alg. 1. The first epoch is initialized with a standard softmax (i.e., setting $q = e - 2, d = 1$). Training then proceeds by re-estimating q and d for each $\boldsymbol{x} \in \mathcal{D}_{\text{tr}}$ after each epoch. We take as the stopping criteria for one learning round as the epoch with the highest average balanced (across classes) median q values over $\mathcal{D}_{\text{ca}}$. We choose the final model $\mathcal{M}_* \in \mathbb{M}$ over J iterations of random shuffles and splits of $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$ and parameter initializations as that with the globally highest average balanced (across classes) median q values over $\mathcal{D}_{\text{ca}}$. For learning, we assume $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$ are balanced across all class labels, $c \in \mathcal{Y}$.

Algorithm 1 sdm Activation Layer and sdm Estimator Training

```
Input: $\mathcal{D}_{\text{tr}}$, $\mathcal{D}_{\text{ca}}$, $\alpha'$, network, max epochs, rescaler max epochs, rescaler stopping condition
Assumption: $\mathcal{D}_{\text{tr}}$, $\mathcal{D}_{\text{ca}}$ are balanced across all class labels, $c \in \mathcal{Y}$
procedure SDM-ITERATIVE-TRAIN($\mathcal{D}_{\text{tr}}$, $\mathcal{D}_{\text{ca}}$, $\alpha'$, network, max epochs)
    $\mathcal{M}_* \leftarrow \emptyset$    ▷ Globally best model
    $\mathcal{D}_{\text{tr}*} \leftarrow \emptyset$, $\mathcal{D}_{\text{ca}*} \leftarrow \emptyset$    ▷ Data splits of best model
    $\mathcal{E} \leftarrow \emptyset$    ▷ SDM estimator (i.e., $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$)
    $\text{metric}_* \leftarrow 0$    ▷ Determines final best model
    stats $\leftarrow \{\ \}$    ▷ Summary statistics to calculate $\tilde{q}^{\gamma}_{\text{min}}, \text{m}^{\hat{y}}_{\lfloor \tilde{q} \rfloor}$ (§ 4.2.4)
    for $j \in 1, \ldots, J$ do    ▷ The learning process is repeated $J$ times
        $\mathcal{M}_{j*} = \emptyset$    ▷ Best model for a single learning round
        $\text{metric}_{\text{j}} \leftarrow 0$
        $\mathcal{D}_{\text{tr}}$, $\mathcal{D}_{\text{ca}}$ $\leftarrow$ Random shuffle and even split of $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$
        $\mathcal{M}_j \leftarrow$ Random initialization of $\boldsymbol{G}_j, \boldsymbol{W}'_j, \boldsymbol{b}'_j$
        $q \leftarrow e - 2, d \leftarrow 1$    ▷ Standard softmax for first epoch
        for $e \in 1, \ldots,$ max epochs do
            Minimize $\mathcal{L}(\boldsymbol{G}, \boldsymbol{W}', \boldsymbol{b}'; \mathcal{D}_{\text{tr}})$    ▷ Eq. 8
            Update $q, d$ for each $\boldsymbol{x} \in \mathcal{D}_{\text{tr}}$
            metric $\leftarrow$ mean balanced (across $c \in \mathcal{Y}$) median $q$ over $\mathcal{D}_{\text{ca}}$
            if metric $\geq \text{metric}_{\text{j}}$ then
                $\text{metric}_{\text{j}} \leftarrow$ metric
                $\mathcal{M}_{j*} \leftarrow \mathcal{M}_j$
            if $\text{metric}_{\text{j}} \geq \text{metric}_*$ then
                $\text{metric}_* \leftarrow \text{metric}_{\text{j}}$
                $\mathcal{M}_* \leftarrow \mathcal{M}_{j*}$
                $\mathcal{D}_{\text{tr}*}, \mathcal{D}_{\text{ca}*} \leftarrow \mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{ca}}$    ▷ Data splits for calculating $q, d$ at test-time & model checks
        $\mathcal{M}_{j*} \leftarrow$ update with $\boldsymbol{W}''$ from TRAIN-RESCALER($\cdot$)    ▷ Alg. 2
        stats $\leftarrow$ update with FIND-MIN-RESCALED-Q($\cdot$)    ▷ Alg. 3
    $\mathcal{E} \leftarrow$ Constructed from globally best model $\mathcal{M}_*$ (and associated values, e.g., $\tilde{q}_{\text{min}*}$) and stats
    return $\mathcal{M}_*, \mathcal{D}_{\text{tr}*}, \mathcal{D}_{\text{ca}*}, \mathcal{E}$
Output: $\mathcal{M}_*, \mathcal{D}_{\text{tr}*}, \mathcal{D}_{\text{ca}*}, \mathcal{E}$
```

4.2 From sdm Activation Functions to sdm Calibration

Given a fixed underlying network, the sdm activation function in Eq. 7 encodes strong signals of the epistemic uncertainty of a single instance for a single model $\mathcal{M}_* \in \mathbb{M}$, but a priori, it is not sufficient alone for calibration without additional exogenous information, since it does not explicitly take into account the epistemic uncertainty from the splitting of $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$; the stochasticity of parameter initialization; and the stochasticity of the learning process, more generally. Relatedly, to enable the interpretability of the calibration process (e.g., to perform model checks), we need a stable mapping of test points to the relevant partitions of $\mathcal{D}_{\text{ca}}$.

In service of achieving these additional properties, we first need to specify a definition of calibration, of which there are conflicting quantities, definitions, and evaluation metrics (Vaicenavicius et al., 2019; Kull et al., 2019; Gupta and Ramdas, 2022). Fortunately, in real-world settings with LLMs, we are primarily concerned with reliably detecting high-probability regions, which significantly simplifies the evaluations and removes much of the ambiguity in the definitions. To motivate our definition, we first consider two under-specified definitions of calibration, in which the true long-run frequencies of the ground-truth labels match the probability estimates from the estimator, $\mathcal{E}$, stratified by the predicted class, $\hat{y}$, and the true class, y, respectively, given some un-specified binning of the real-valued probabilities:

Definition 1 *An estimator, $\mathcal{E}$, of $p(y \,|\, \boldsymbol{x})$ is prediction-conditional calibrated, if $\forall\ \alpha' \in [0,1]$: $p(y = \hat{y} \,|\, \hat{y}, \mathcal{E}(\boldsymbol{x}) = \alpha') = \alpha'$.*

Definition 2 *An estimator, $\mathcal{E}$, of $p(y \,|\, \boldsymbol{x})$ is class-conditional calibrated, if $\forall\ \alpha' \in [0,1]$: $p(y = \hat{y} \,|\, y, \mathcal{E}(\boldsymbol{x}) = \alpha') = \alpha'$.*

Assuming no distribution shifts, and setting aside conditioning on additional attributes and the method of binning, the source of the under-specification, Def. 2 is a generally more informative quantity, but cannot be meaningfully estimated across all points since the true label, y, is not available at test-time. Thus, calibration becomes a tension between the quantities desired and the regions—and the size (sharpness) of those regions—that can be partitioned. Most works are premised on a variation of Def. 1; an alternative compromise is taken by frequentist conformal estimators by changing the quantity to coverage over a discrete prediction set. We will instead seek the following quantity, which aligns with the quantity needed for selective classification for conditional branching of LLM compute and final human decision-making dependent on the presence of high-probability predictions:

Definition 3 *An estimator, $\mathcal{E}$, of $p(y \,|\, \boldsymbol{x})$ is index-conditional calibrated at $\alpha' \in (\frac{1}{C}, 1]$ if: $p(y = \hat{y} \,|\, \hat{y}, \mathcal{E}(\boldsymbol{x}) \geq \alpha') \geq \alpha' \wedge p(y = \hat{y} \,|\, y, \mathcal{E}(\boldsymbol{x}) \geq \alpha') \geq \alpha'$.*

To evaluate this quantity, we only consider the points for which the estimator assigns a high-probability of at least α', which is typically near 1, such as $1 - \alpha = \alpha' = 0.95$ in our experiments. We refer to this set of points as the *admitted*, or *non-rejected*, set. Then, given ground-truth values for $\mathcal{D}_{\text{te}}$, we assess whether the conditional accuracies of the admitted set are at least α' when stratifying by the predicted labels, $\hat{y}$, and the true labels, y. Unlike evaluating Def. 1, there is thus no ambiguity with regard to the choice of binning the probabilities.

The estimator that rejects all points is index-conditional calibrated. Given two estimators that are index-conditional calibrated, we prefer that which rejects fewer points, ceteris paribus. In other words, we seek estimators that meet our reliability condition and are informative (i.e., maximize the number of points that are properly admitted), but when the estimator is uncertain, we prefer rejection over unexpectedly falling under the desired α' probability threshold.

The key compromise is that we will not be able to reliably calculate a probability for all points; however, for LLM tasks, there is typically not an actionable notion of partial acceptability for final decision-making, so it is a reasonable compromise. Either the complex

LLM output is verified as correct, or some separate, remedial action must be taken, such as dividing the task into simpler tasks, reformatting and re-cross-encoding, or seeking outside information retrieval, among others, where again for each of these sub-tasks, we seek index-conditional calibrated estimators at the level of the available labels, where the stopping condition is eventually deferment to human adjudication.

Despite the aforementioned compromise, and although evaluation is unambiguous, it may still seem mysterious that the second condition of Def. 3 can be meaningfully estimated. To do so, we will need to perform a series of transforms over the already strong uncertainty signals from the SDM activation function and re-visit the behavior of partitioning empirical CDFs, to which we turn next.

4.2.1 Rescaling SDM Activation Output to Account for Effective Sample Sizes

A disadvantage of using $\text{SDM}(\boldsymbol{z}')$ directly as an estimator is that it only has an indirect, relative notion of the effective sample size of $\mathcal{D}_{\text{ca}}$. Intuitively, the confidence in a prediction should be commensurate with the number of comparable points in $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$, which the SDM activation captures via Similarity, Distance, and Magnitude. For example, an out-of-distribution point will tend to have $d = 0$ and low values of q, reflecting a small effective sample size in the observed data. However, to further improve the robustness of the estimate, we can explicitly incorporate an additional, direct notion of the effective sample size via distributional statistics over $\mathcal{D}_{\text{ca}}$.

First, we calculate class-conditional empirical CDFs over $\mathcal{D}_{\text{ca}}$ of the output of $\text{SDM}(\boldsymbol{z}')$. For a given point, this will create a vector, $\boldsymbol{v} \in \mathbb{R}^C$, of the quantiles:

$$\boldsymbol{v} = \left[\text{eCDF}_{\text{ca}}^{y_1}(\text{SDM}(\boldsymbol{z}')_1), \ldots, \text{eCDF}_{\text{ca}}^{y_C}(\text{SDM}(\boldsymbol{z}')_C)\right] \tag{9}$$

Next, we rescale q to take into account these distributional statistics. The resulting value will be the basis for our stable mapping between new, unseen test points and $\mathcal{D}_{\text{ca}}$:

$$\tilde{q} = log_e\left((2+q)^{\boldsymbol{v}_{\hat{y}}}\right) \tag{10}$$

We seek a normalized distribution both to present to users and to enable the subsequent transform described in § 4.2.3. Toward this end, we rescale with a linear layer, without a bias, the training of which we detail in § 4.2.2: $\boldsymbol{v}' = \boldsymbol{W}''^T\boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^C$. This is normalized using $2 + \tilde{q}$ as the base, $\boldsymbol{o} \in \mathbb{R}^C$:

$$o_i = \frac{(2+\tilde{q})^{\ v_i'}}{\sum_{c=1}^{C}(2+\tilde{q})^{\ v_c'}}, 1 \leq i \leq C \tag{11}$$

Unlike the output of an SDM activation, $\arg\max \boldsymbol{o}$ is not necessarily (but typically will be) equivalent to $\hat{y} = \arg\max \boldsymbol{z}'$. When they are not equivalent, our convention is to set $\tilde{q} = 0$ for the point, which will in effect treat the point as out-of-distribution in downstream analyses.

Effective Sample Sizes via the DKW Inequality Eq. 11 is premised on the assumption that the empirical CDFs in Eq. 9 reflect the true, underlying conditional distributions, which are unspecified.[5] That would seem to be a relatively strong assumption as the final estimate,

[5] That is also true for the eCDFs in Eq 6, but we make the reasonable assumption that the higher-level transforms starting with Eq. 9 effectively account for the uncertainty in the distance eCDFs.

particularly for small sample sizes, even if empirically effective over existing datasets, and is the entry point for incorporating an explicit notion of the effective sample size in our estimates.

We make the following conservative assumption, parameterizing the prior belief that data points with a looser connection to $\mathcal{D}_{\rm tr}$ reflect smaller effective sample sizes, while also explicitly accounting for the count of observed points in $\mathcal{D}_{\rm ca}$:

Assumption 4 *We assume the effective sample size is increasing in $\tilde{q}$, class-wise over $\mathcal{D}_{\rm ca}$.*

For each $\boldsymbol{x} \in \mathcal{D}_{\rm te}$, using $\tilde{q}$, we calculate the vector of effective sample sizes across classes, $\hat{\mathbf{n}}$, relative to $\mathcal{D}_{\rm ca}$ as:

$$\hat{\mathbf{n}} = [\,|\,\mathcal{D}_{\rm ca}\,|^{y_1} \cdot \mathrm{eCDF}^{y_1}_{\rm ca}(\tilde{q}), \ldots, |\,\mathcal{D}_{\rm ca}\,|^{y_C} \cdot \mathrm{eCDF}^{y_C}_{\rm ca}(\tilde{q})] \tag{12}$$

where $|\,\mathcal{D}_{\rm ca}\,|^{y_c}$ is the count of calibration set points with true label $y = c$.

With these sample size estimates, we can then construct a band around the empirical CDFs using the sharp constant (Massart, 1990) of the distribution-free DKW inequality (Dvoretzky et al., 1956), calculating the error for each class $c \in \{1, \ldots, C\}$ from the corresponding index in $\hat{\mathbf{n}}$ if $\hat{n}_c > 0$:

$$\epsilon_c = \sqrt{\frac{1}{2 \cdot \hat{n}_c} \log_e \left(\frac{2}{1 - \alpha'} \right)} \tag{13}$$

If $\hat{n}_c = 0$ our convention is to set $\epsilon_c = 1$. We can then construct the lower and upper counterparts to the quantile vector of Eq. 9:

$$\begin{aligned} \boldsymbol{v}_{\rm lower} = [\,&\min\left(\max\left(\mathrm{eCDF}^{y_1}_{\rm ca}(\textsc{sdm}(\boldsymbol{z}')_1) - \mathbf{1}_{\hat{y}=1} \cdot \epsilon_1 + \mathbf{1}_{\hat{y}\neq 1} \cdot \epsilon_1, 0\right), 1\right), \ldots, \\ &\min\left(\max\left(\mathrm{eCDF}^{y_C}_{\rm ca}(\textsc{sdm}(\boldsymbol{z}')_C) - \mathbf{1}_{\hat{y}=C} \cdot \epsilon_C + \mathbf{1}_{\hat{y}\neq C} \cdot \epsilon_C, 0\right), 1\right)] \end{aligned} \tag{14}$$

$$\begin{aligned} \boldsymbol{v}_{\rm upper} = [\,&\min\left(\max\left(\mathrm{eCDF}^{y_1}_{\rm ca}(\textsc{sdm}(\boldsymbol{z}')_1) + \mathbf{1}_{\hat{y}=1} \cdot \epsilon_1 - \mathbf{1}_{\hat{y}\neq 1} \cdot \epsilon_1, 0\right), 1\right), \ldots, \\ &\min\left(\max\left(\mathrm{eCDF}^{y_C}_{\rm ca}(\textsc{sdm}(\boldsymbol{z}')_C) + \mathbf{1}_{\hat{y}=C} \cdot \epsilon_C - \mathbf{1}_{\hat{y}\neq C} \cdot \epsilon_C, 0\right), 1\right)] \end{aligned} \tag{15}$$

from which $\tilde{q}_{\rm lower}$ and $\tilde{q}_{\rm upper}$ follow:

$$\tilde{q}_{\rm lower} = log_e\left((2+q)^{\boldsymbol{v}_{{\rm lower}_{\hat{y}}}}\right) \tag{16}$$

$$\tilde{q}_{\rm upper} = log_e\left((2+q)^{\boldsymbol{v}_{{\rm upper}_{\hat{y}}}}\right) \tag{17}$$

Analogous to Eq. 11, we then construct our estimators after rescaling $\boldsymbol{v}'_{\rm lower} = \boldsymbol{W}''^{T}\boldsymbol{v}_{\rm lower}$, $\boldsymbol{v}'_{\rm lower} \in \mathbb{R}^C$ and $\boldsymbol{v}'_{\rm upper} = \boldsymbol{W}''^{T}\boldsymbol{v}_{\rm upper}$, $\boldsymbol{v}'_{\rm upper} \in \mathbb{R}^C$:

$$p(\hat{y})_{\rm lower} = \frac{\left(2 + \tilde{q}_{\rm lower}\right)^{v'_{{\rm lower}_{\hat{y}}}}}{\sum_{c=1}^{C} \left(2 + \tilde{q}_{\rm lower}\right)^{v'_{{\rm lower}_c}}} \tag{18}$$

$$p(\hat{y})_{\rm centroid} = o_{\hat{y}} \quad \rhd \text{ from Eq. 11} \tag{19}$$

$$p(\hat{y})_{\rm upper} = \frac{\left(2 + \tilde{q}_{\rm upper}\right)^{v'_{{\rm upper}_{\hat{y}}}}}{\sum_{c=1}^{C} \left(2 + \tilde{q}_{\rm upper}\right)^{v'_{{\rm upper}_c}}} \tag{20}$$

As with Eq. 11, the convention is to set $\tilde{q}_{\text{lower}} = 0$ and/or $\tilde{q}_{\text{upper}} = 0$ for the rare cases for which the transforms in Eq. 18 and/or Eq. 20, respectively, result in the arg max value of the normalized output vector not being equivalent to $\hat{y} = \arg\max \boldsymbol{z'}$. (In such cases, e.g., Eq. 18 is not re-calculated with $\tilde{q}_{\text{lower}} = 0$, but rather such values are treated separately in downstream analyses as out-of-distribution points.)

Base Estimators $p(\hat{y})_{\text{lower}} \in \mathbb{R}^1$ will be used as the basis of our primary test-time estimator of prediction-conditional uncertainty (see § 4.2.5 for the complete, index-conditional estimator). $p(\hat{y})_{\text{centroid}} \in \mathbb{R}^1$ (via Eq. 11) is a consequence of intermediate results needed in service of constructing $p(\hat{y})_{\text{lower}}$ (e.g., for training the re-scaler and setting a threshold on $\tilde{q}$, described below), whereas $p(\hat{y})_{\text{upper}} \in \mathbb{R}^1$ is primarily only of research interest, included here to analyze the behavior of the approach.[6]

4.2.2 Training the Rescaling Transform

We train the C^2 parameters of $\boldsymbol{W}''$ of the re-scaling linear layer over $\mathcal{D}_{\text{ca}}$ (*not* $\mathcal{D}_{\text{tr}}$) by minimizing the following loss (Alg. 2), which is the counterpart to Eq. 11, while all other parameters remain fixed:

$$\mathcal{L}(\boldsymbol{W}''; \mathcal{D}_{\text{ca}}) = -\frac{1}{|\,\mathcal{D}_{\text{ca}}\,|} \sum_{n}^{|\,\mathcal{D}_{\text{ca}}\,|} \log_{(2+\tilde{q})} \left(\frac{(2+\tilde{q})^{v'_{y_n}}}{\sum_{c=1}^{C} (2+\tilde{q})^{v'_c}} \right) \tag{21}$$

Our convention is to train with a batch size of 1 and conclude the learning process if $\mathcal{L}(\boldsymbol{W}''; \mathcal{D}_{\text{ca}})$ increases for a pre-specified (as a hyper-parameter) number of consecutive epochs.

4.2.3 Region-specific eCDFs

The estimators $p(\hat{y})_{\text{lower}}$, $p(\hat{y})_{\text{centroid}}$, and $p(\hat{y})_{\text{upper}}$ incorporate explicit notions of the effective sample sizes. Smaller effective sample sizes will be associated with lower probability estimates (and vice-versa). They also have strong relative notions of the highest probability regions of the output distribution by virtue of the original Similarity, Distance, and Magnitude signals, and the aggregated distributional statistics over these signals. However, what they lack is a human interpretable, principled cutoff, or threshold, by which we can have some assurance that the new points we see are reasonably comparable to the data we observed in deriving our estimators. This is a more subtle and foundational problem than it may initially seem; we must account for distribution shifts if we seek to realistically achieve our desired notion of index-conditional calibration (Def. 3). It will require an additional set of transforms to resolve, even with the already strong signals of prediction-conditional uncertainty from our estimators, to which we turn next.

It follows from Eq. 1 that the output of softmax($\boldsymbol{z}$) can be viewed as softmax($\boldsymbol{z}$) $= \triangle^{C-1}$, which is the $(C-1)$-dimension simplex, where the dimension reduction is a consequence of the output summing to 1. The same is true of the normalized value $\boldsymbol{o}$. If we instead consider the over-parameterized version in which each event probability of the categorical

[6] In practice, rather than using $p(\hat{y})_{\text{upper}}$, if a less stringent admission criteria is desired, the operative action is to reduce α' and re-estimate $p(\hat{y})_{\text{lower}}$.

Algorithm 2 Training the Weights of the Rescaling Transform

```
Input: cached $\boldsymbol{v}$ for $\mathcal{D}_{\text{ca}}$, rescaler max epochs, rescaler stopping condition
procedure TRAIN-RESCALER(cached $\boldsymbol{v}$ for $\mathcal{D}_{\text{ca}}$, rescaler max epochs, rescaler stopping condition)
    $\boldsymbol{W}_*^{''} \leftarrow \emptyset$                                              ▷ Final weights
    $\boldsymbol{W}^{''} \leftarrow$ random initialization
    metric $\leftarrow \infty$
    counter $\leftarrow 0$
    for $e \in 1, \ldots,$ rescaler max epochs do
        Minimize loss $\leftarrow \mathcal{L}(\boldsymbol{W}^{''}; \mathcal{D}_{\text{ca}})$                          ▷ Eq. 21
        if loss < metric then
            metric $\leftarrow$ loss
            $\boldsymbol{W}_*^{''} \leftarrow \boldsymbol{W}^{''}$
        if loss > metric then
            counter $\leftarrow$ counter + 1
            if counter > rescaler stopping condition then
                break
        else
            counter $\leftarrow 0$
    return $\boldsymbol{W}_*^{''}$
Output: $\boldsymbol{W}_*^{''}$
```

distribution (e.g., Eq. 2) is explicitly specified as an element of a vector of length C, the following indicator result directly follows:

Remark 5 *Given the C class-conditional CDFs over categorical distributions where the $1-\alpha'$ ($\alpha' \in (\frac{1}{C}, 1]$) quantile threshold ψ_c ($\psi_c \in [0,1]$) of each class $c \in \{1, \ldots, C\}$ is $> \frac{1}{C}$ (i.e., $\psi_c = \text{inverseCDF}^{y_c}(1-\alpha') > \frac{1}{C}\ \forall\ c\ \in \{1, \ldots, C\}$), a set of i.i.d. points sampled from the same distribution as the CDFs, each of whose event probability vector $\boldsymbol{e} = [e_1, \ldots e_C]$ has one (1) element at least the corresponding class threshold (i.e., $|[e_1, \ldots e_C] \geq [\psi_1, \ldots \psi_C]| = 1$, with the comparison taken element-wise), will have class-conditional accuracies $\geq \alpha'$, in expectation.*

Proof Partition the class-conditional CDFs of the categorical distributions, for which $\psi_c = \text{inverseCDF}^{y_c}(1-\alpha') > \frac{1}{C}\ \forall\ c\ \in \{1, \ldots, C\}$, at $[\psi_1, \ldots \psi_C]$. The resulting high-probability partitions—those $\geq \psi_c$—are C Bernoulli distributions each with success probability $p_c \geq \alpha'$. Take as $[n_1, \ldots n_C]$ the class-wise count of i.i.d. points whose event probability vector, $\boldsymbol{e}$, satisfies $|[e_1, \ldots e_C] \geq [\psi_1, \ldots \psi_C]| = 1$. Then by the definition of the expected value of a Binomial distributed random variable, it follows from these trials that $[\frac{n_1 \cdot p_c}{n_1}, \ldots, \frac{n_C \cdot p_C}{n_C}] = [\geq \alpha', \ldots, \geq \alpha']$, which is the desired class-conditional accuracy for this restricted set of points. Now, instead assume that one or more of the Bernoulli distributions has a success probability $p_c < \alpha'$. This implies that the class-conditional CDFs were constructed from a distribution whose event probabilities are not those of the $(C-1)$-dimension simplex since we require $\psi_c = \text{inverseCDF}^{y_c}(1-\alpha') > \frac{1}{C}\ \forall\ c\ \in \{1, \ldots, C\}$ with the CDFs constructed class-wise relative to the true labels, which is a contradiction of the definition of a categorical distribution since the sum of all event probabilities, each of which is a real value in $[0,1]$, must equal 1. ■

Note that when $\psi_c < \frac{1}{C}$ no such assurance across all classes necessarily results, since the resulting thresholding of the probability vectors may induce a complex dependence across the class-conditional CDFs.[7] In such cases, the thresholding of a new point may result in multiple classes above the threshold, and the subsequent stratification of this set of points to those for which $|[e_1, \dots e_C] \geq [\psi_1, \dots \psi_C]| = 1$ will not necessarily have class-conditional accuracies $\geq \alpha'$, in expectation.

Remark 5 thus differs from set-valued estimators such as conformal estimators (Vovk et al., 2005), which as previously mentioned (see § 4.2, introduction) are premised on a different calibration compromise. For example, with conformal estimators, there is a statistical assurance for coverage of the true class in a discrete prediction set (itself a distinct quantity from that considered here) across all points regardless of the distribution of the conformity score (e.g., instead of a categorical distribution, a conformity score can be an unnormalized scoring function), but no assurance conditional on the subset of high-probability points. We explore the implications of these tradeoffs in our empirical experiments.

Remark 5 can be viewed as a useful indicator function, but it is not particularly informative as an estimator alone. We will use it in service of dividing the output distribution into high probability regions via $\tilde{q}$, described next.

Corralling the high-probability region via exclusion of the observed high-epistemic-uncertainty points. Intuitively, higher values of $\tilde{q}$ correspond to points with a closer connection to the observed data and thus lower epistemic uncertainty, as this single value takes into account the Similarity, Distance, and Magnitude signals, and distributional statistics over those signals. The result in Remark 5 provides a principled basis for setting a threshold on $\tilde{q}$ over $\mathcal{D}_{\text{ca}}$ that we can then apply at test-time, without access to the true label, to constrain our estimates to the high-probability region of the distribution.

The value of $\tilde{q}$ is real-valued, but only $\leq |\, \mathcal{D}_{\text{ca}} \,|$ values are observed, so a simple iterative search algorithm is sufficient to find the value of $\tilde{q}$ that satisfies Remark 5 such that all thresholds, ψ_c, over the estimates of $\boldsymbol{o}$ (Eq. 11), are at least α'. By definition, $\alpha' > \frac{1}{C}$, so this more stringent requirement satisfies the condition in Remark 5, while also requiring $\tilde{q}$ to be restricted to the prediction-conditional estimates of $p(\hat{y})_{\text{centroid}} \geq \alpha'$. The full algorithm appears in Alg. 3, iteratively constructing class-wise eCDFs over $\mathcal{D}_{\text{ca}}$ restricted to progressively larger values of $\tilde{q}$. (These eCDFs over the $\boldsymbol{o}$ values of $\mathcal{D}_{\text{ca}}$ are only needed for Alg. 3 and are not needed at test-time, unlike those of Eq. 6, Eq. 9, and Eq. 12.) Note that we only consider values of $\lfloor \tilde{q} \rfloor > 0$, as points with $\lfloor \tilde{q} \rfloor = 0$ are considered out-of-distribution.[8] The search algorithm may fail to find a suitable final value, $\tilde{q}_{\text{min}}$, at which point the operative conclusion is that reliable estimates of index-conditional calibration (Def. 3) are not possible without reducing α', or acquiring additional data and/or a stronger model.[9]

[7] We leave to future work whether a similar result holds for a subset of the class-conditional accuracies if only *some*, rather than *all*, class-wise thresholds are at least $\frac{1}{C}$. In our primary verification setting over LLMs, the typical setting is $C = 2$, or some similarly small $C \in \mathbb{Z}^+$, where to ensure deployment reliability, the LLM would not be deployed until at least the verification task yields class-conditional accuracies, for all classes, at or above α' on the available labeled sets, so we do not consider this case here.

[8] The reason for the floor operation becomes evident in the next section. $\lfloor \tilde{q} \rfloor$ will serve as our hard-partitioned mapping between the observed data and new test points to enable estimates of uncertainty over iterations of the entire process described thus far.

[9] Alg. 3 could be readily modified to find an adaptive value of α', iteratively reducing α' if a suitable $\tilde{q}_{\text{min}}$ value is not found. However, in practice, determining α' for LLM settings is a decision made exogenous

When a value of $\tilde{q}_{\text{min}}$ can be found, the convention is to restrict our estimates of index-conditional calibration to the new, unseen test points that satisfy $\tilde{q}_{\text{lower}} \geq \tilde{q}_{\text{min}}$ after considering the final additional sources of uncertainty from the data splitting and learning processes, which we consider next.

Algorithm 3 Search Algorithm to Find $\tilde{q}_{\text{min}}$ to Detect High-Probability Regions

```
Input: cached $(\tilde{q}, \boldsymbol{o})$ for $\mathcal{D}_{\text{ca}}$, $\alpha' \in (\frac{1}{C}, 1]$
 1: procedure FIND-MIN-RESCALED-Q(cached $(\tilde{q}, \boldsymbol{o})$ for $\mathcal{D}_{\text{ca}}$, $\alpha' \in (\frac{1}{C}, 1]$)
 2:     $\tilde{q}_{\text{min}} \leftarrow \emptyset$    ▷ A suitable $\tilde{q}_{\text{min}}$ may not exist.
 3:     $[\psi_1, \dots \psi_C] \leftarrow [\emptyset, \dots, \emptyset]$    ▷ Needed at test-time, if applicable
 4:     $\tilde{q}s \leftarrow$ sorted $[\tilde{q} \in \mathcal{D}_{\text{ca}}$ s.t. $\lfloor \tilde{q} \rfloor > 0]$    ▷ Restricted to $\lfloor \tilde{q} \rfloor > 0$ to exclude OOD
 5:     for $\tilde{q}' \in \tilde{q}s$ do
 6:         Construct $\text{eCDF}_{\text{ca}}^{y_1}, \dots, \text{eCDF}_{\text{ca}}^{y_C}$ for all $\tilde{q} \geq \tilde{q}'$ in $\mathcal{D}_{\text{ca}}$ ▷ eCDFs for $\boldsymbol{o}$ (Eq. 11), stratified by $y$
 7:         Calculate $\psi_c = \text{inverseCDF}_{\text{ca}}^{y_c}(1 - \alpha')\ \forall\ c\ \in \{1, \dots, C\}$ ▷ Quantile functions are inverses of L. 6
 8:         if all( $[\psi_1, \dots \psi_C] \geq \alpha'$ ) then    ▷ Element-wise comparison
 9:             $\tilde{q}_{\text{min}} \leftarrow \tilde{q}'$ ▷ Satisfies Remark 5 at the prediction-conditional estimate (see text) of $\geq \alpha'$
10:             break
11:     return $\tilde{q}_{\text{min}}$, $[\psi_1, \dots \psi_C]$
Output: $\tilde{q}_{\text{min}}$, $[\psi_1, \dots \psi_C]$
```

4.2.4 Accounting for Uncertainty in the Data Splitting and Learning Processes

As a final step, we take into account uncertainty over the data splitting and learning processes. This will incur non-trivial additional computational costs, but these are one-time development costs for an estimator. At test-time, our estimates will be constant offsets on $\tilde{q}_{\text{min}}$ and $p(\hat{y})_{\text{lower}}$, the latter conditional on $\lfloor \tilde{q} \rfloor \in \mathbb{Z}^{0+}$, which will serve as a stable mapping between $\mathcal{D}_{\text{ca}}$ and new, unseen test points. In summary, in this section, we seek:

$$\tilde{q}_{\text{min}}^{\ \gamma} \qquad \triangleright \text{ A robust estimate of } \tilde{q}_{\text{min}} \tag{22}$$

$$\text{m}_{\lfloor \tilde{q} \rfloor}^{\hat{y}} \qquad \triangleright \text{ A class-wise, robust correction for } p(\hat{y})_{\text{lower}}, \text{ conditional on } \lfloor \tilde{q} \rfloor \tag{23}$$

Conceptually, the estimation process is straightforward. We repeat the training and estimation processes described above J times and derive our constant offsets via summary statistics over those estimates. The one complication that arises is that we will have to depart from the distribution-assumption-light approaches above, since J will typically not be large due to the computational expense. (The full process across J to construct a single estimator needs to remain reasonably computationally lightweight relative to an LLM training epoch, as it itself will be embedded into the learning loop of an LLM, described below.) Instead, we will estimate each of these processes as a Cauchy distribution, given its relatively wide tails and relatively robust scale parameter.

to the model development process. We seek to develop our models (and data) to meet a given α' value, rather than the other way around, so we do not consider that variation here.

A Cauchy distribution is defined by a location parameter, ν, and a scale parameter, γ:

$$\text{Cauchy}(\nu, \gamma) \tag{24}$$

The inverse CDF (i.e., quantile function) of a Cauchy distribution for a particular quantile, $\alpha \in [0, 1]$, can be calculated analytically as:

$$\text{inverseCDF}_{\text{Cauchy}(\nu,\gamma)}(\alpha) = \nu + \gamma \tan\left(\pi \left(\alpha - \frac{1}{2}\right)\right) \tag{25}$$

We take as our estimate of γ the median absolute deviation around the median of our sample (MAD).

Robust detection of high-probability regions. To calculate γ for $\tilde{q}^{\,\gamma}_{\text{min}} \in \mathbb{R}^1$, we take the MAD of the J estimates of $\tilde{q}_{\text{min}}$. The location parameter is taken as $\tilde{q}_{\text{min}*}$, the estimate of $\tilde{q}_{\text{min}}$ over the model with the final chosen weights (see Alg. 1). We can then analytically calculate our desired value via Eq. 25 at $\alpha' \in (\frac{1}{C}, 1]$:

$$\tilde{q}^{\,\gamma}_{\text{min}} = \text{inverseCDF}_{\text{Cauchy}(\tilde{q}_{\text{min}*},\gamma)}(\alpha') \tag{26}$$

Note that since α' corresponds to the right-tail of the distribution, $\tilde{q}^{\,\gamma}_{\text{min}} \geq \tilde{q}_{\text{min}*}$, i.e., a more restrictive threshold on the high-probability region. In scenarios (not considered in the experiments here) where the computational budget necessitates $J = 1$, the convention would be to take $\tilde{q}^{\,\gamma}_{\text{min}} \coloneqq \tilde{q}_{\text{min}*}$, with a tacit assumption that these additional sources of uncertainty have not been explicitly accounted for.

Robust output adjustment. To calculate γ for $\text{m}^{\hat{y}}_{\lfloor \tilde{q} \rfloor}$ conditional on $\hat{y}$ and $\lfloor \tilde{q} \rfloor$ (i.e., $\gamma \mid \hat{y}, \lfloor \tilde{q} \rfloor$), we take the MAD of the J *medians* (as written) of $p(\hat{y})_{\text{centroid}}$ over $\mathcal{D}_{\text{ca}}$, conditional on $\hat{y}$ and $\lfloor \tilde{q} \rfloor$.[10] Similar to above, we can then calculate:

$$\text{m}^{\hat{y}}_{\lfloor \tilde{q} \rfloor} = \text{inverseCDF}_{\text{Cauchy}(0,(\gamma \mid \hat{y}, \lfloor \tilde{q} \rfloor))}(\alpha') \tag{27}$$

In this case, ν is 0, as $\text{m}^{\hat{y}}_{\lfloor \tilde{q} \rfloor}$ will be subtracted from $p(\hat{y})_{\text{lower}}$ as an offset, an assumption that each distribution is centered on the given point. To simplify the presentation (and since the upper offset is not needed in practice), we only consider this as a lower offset on our base estimators.

As $\lfloor \tilde{q} \rfloor$ increases, the number of points in the sample will tend to decrease, but so will the MAD, so the estimates remain reasonable in practice. As we will see in our experiments, high values of $\lfloor \tilde{q} \rfloor$ (that are otherwise attested in $\mathcal{D}_{\text{ca}}$) are not uncommonly associated with MAD values that are within 0 of numerical precision.

As with $\tilde{q}^{\,\gamma}_{\text{min}}$, although it is generally recommend to take these additional sources of uncertainty into consideration, when $J = 1$, the convention would be to take $\text{m}^{\hat{y}}_{\lfloor \tilde{q} \rfloor} \coloneqq 0$.

[10] Implementation note: Unlike the eCDFs, which are constructed by stratifying on the true label, y, in $\mathcal{D}_{\text{ca}}$, this quantity is calculated by stratifying on the predicted label, $\hat{y}$, in $\mathcal{D}_{\text{ca}}$.

4.2.5 Index-Conditional Calibration

With the above models and estimators, we can now robustly calculate the index-conditional uncertainty of a new, unseen test point $\boldsymbol{x} \in \mathcal{D}_{\text{te}}$.

We first take as the prediction $\hat{y} = \arg\max \boldsymbol{z}'$. Then, with $\mathcal{D}_{\text{tr}}$ to calculate q and d_{nearest}; the cached class-wise empirical CDFs over $\mathcal{D}_{\text{ca}}$ of Eq. 6, Eq. 9, and Eq. 12; $\tilde{q}^{\,\gamma}_{\text{min}}$ and the thresholds $([\psi_1, \dots \psi_C])$; and $\mathrm{m}^{\hat{y}}_{\lfloor\tilde{q}\rfloor}$, the index-conditional uncertainty estimate of $p(y \,|\, \boldsymbol{x})$ at α' (Def. 3) is:

$$\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}} = \begin{cases} \max(0, p(\hat{y})_{\text{lower}} - \mathrm{m}^{\hat{y}}_{\lfloor\tilde{q}\rfloor}) & \text{if } \left[\tilde{q}_{\text{lower}} \geq \tilde{q}^{\,\gamma}_{\text{min}}\right] \wedge \left[\left(p(\hat{y})_{\text{lower}} - \mathrm{m}^{\hat{y}}_{\lfloor\tilde{q}\rfloor}\right) \geq \psi_{\hat{y}}\right] \\ \perp & \text{otherwise} \end{cases} \tag{28}$$

where $\perp$ indicates a rejected (non-admitted) point.[11]

As noted in the previous sections, in the rare cases when the transforms after the SDM activation result in the arg max index not matching $\hat{y}$, we set $\tilde{q}_{\text{lower}} = 0$, which effectively treats the point as out-of-distribution. In such cases, $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}} = \perp$, since $\tilde{q}^{\,\gamma}_{\text{min}} > 0$ as a consequence of Line 4 in Alg. 3.

Our convention in subsequent sections will be to refer to summary statistics and comparisons of $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ (Eq. 28), excluding the points assigned $\perp$, as estimates from the "estimator $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$". We do the same for the "estimator $\hat{p}(y \,|\, \boldsymbol{x})_{\text{centroid}}$" and the "estimator $\hat{p}(y \,|\, \boldsymbol{x})_{\text{upper}}$", but where the latter two quantities are calculated from the corresponding centroid and upper intermediate quantities, respectively.

Complexity. The added computational overhead over an underlying network with a softmax activation is dominated by calculating q (and by extension, d_{nearest}). The transforms after the SDM activation function add negligible additional overhead. For perspective, this is on the order of the additional computation needed for commonly used dense retrieval augmentations of LLMs, so it is readily achievable at interactive speeds in practice.

Sharpness. As noted in § 4.2, we seek estimators that are both informative (i.e., not unnecessarily rejecting correct predictions) and robust (i.e., we prefer rejection over falling under the expected α' accuracy). The above transforms seek to achieve this by taking the uncertainty signals from an SDM activation and further separating the high and low probability regions of the distribution, as well as providing a hard cut via $\tilde{q}^{\,\gamma}_{\text{min}}$ to altogether exclude predictions over high epistemic uncertainty regions. We explore these behaviors empirically in our experiments.

Next, we incorporate our estimators directly into LLM next-token training.

4.3 From SDM Calibration to SDM Networks

The above approach is already a very powerful and easily implemented mechanism for building complex LLM pipelines. We can treat an underlying network as fixed, add an SDM activation layer, and then use the SDM estimator for conditional branching for test-time compute, retrieval, tool-calling, and related.

[11] At test-time, the mapping to the $\lfloor\tilde{q}\rfloor$-conditioned statistics (i.e., $\mathrm{m}^{\hat{y}}_{\lfloor\tilde{q}\rfloor}$) is via $\lfloor\tilde{q}_{\text{lower}}\rfloor$ for the test instance.

However, earlier in the model development pipeline (e.g., as done by LLM model providers), we need a mechanism for fine-tuning a network after the initial unsupervised training stage.[12] In this section, we show how to incorporate the sdm mechanism directly into the LLM next-token training process. We will refer to this process and the resulting model as an sdm network.

Conceptually, an sdm activation and estimator over an averaged history of frozen hidden states and the token-level hidden state will be trained for binary classification at the unit of analysis of the available labels (e.g., the document-level). This estimator then provides the Similarity and Distance values for an sdm activation for next-token loss of the LLM during training. Because an sdm activation does not alter the arg max prediction, greedy token-level generation can proceed without the computational cost of the sdm activation at every token at test-time, with the global sdm estimator providing verification over the final generation. **This process shares the same goal of existing fine-tuning approaches to increase overall accuracy, add information to a model, etc., as well as the new goal of increasing the proportion of verifiable high-probability generations from a model.** During training, we seek to penalize the model for verification mistakes, and reward the model for increasing the cardinality of the set of admitted points.

We first introduce our data encoding scheme in § 4.3.1 for verification. Next, orthogonal to the sdm mechanism itself, we introduce a parsimonious regularization method (§ 4.3.2) to enable fine-tuning on a small amount of data while discouraging catastrophic forgetting. Finally, we introduce the process for training the sdm network (§ 4.3.3).

4.3.1 Universal Verification Encoding

In the abstract, our data is similar to that in the previous sections: Input documents accompanied with discrete labels. However, while we previously treated each document, $\boldsymbol{x}$, as an atomic whole, we will now also be concerned with the individual tokens of the document, for which we use the notation $\mathcal{D}_{\text{tr}} = \{(\boldsymbol{x}_n = [x_1, \ldots, x_T], y_n, [y_n^{\text{task}}])\}_{n=1}^{N}$ for our labeled training set, and similarly for our labeled calibration set, $\mathcal{D}_{\text{ca}}$. Each token, $x_t \in \{1, \ldots, |\mathcal{V}| \cdot 2\}$, is represented as an index into a vocabulary, where $\mathcal{V}$ is the vocabulary of the LLM trained during the initial unsupervised training stage. The reason for the factor of 2 is described in the next section. Implicit in our representation is that each instance will have a marker at some x_t indicating a "completion" (i.e., a sequence after an instruction prompt or prefix, more generally). Our document-level labels, $y \in \mathcal{Y} = \{0, 1\}$, are as in previous sections, but specifically restricted to binary classification, where the convention is to treat $y = 0$ as representing the `unverified` class and $y = 1$ as the `verified` class (i.e., an acceptable generation, conditional on the instruction or context).

For some documents, we have classification labels, $y^{\text{task}} \in \mathbb{Z}^{2+}$, for the underlying tasks encoded in the data. For example, for a sentiment classification task of negative and positive reviews, $y = 0$ for verification when the classification decision is wrong, whereas $y = 1$ for verification when the classification decision is correct. Among those for $y = 1$, $y^{\text{task}} = 0$ could indicate a negative review and $y^{\text{task}} = 1$ could indicate a positive review. These task-specific labels are predicted via the generated text of the LLM, and if available, we can use them

[12] In principle, the methods in this section can also be used for bootstrapping a randomly initialized LLM against an existing (possibly larger) model, which we leave to future work.

during training (e.g., as part of our stopping criteria to choose the best weights, by parsing the generated text and comparing to y^{task}). Unlike typical classification settings, these labels may—and typically will—cover multiple disparate tasks; hence, the designation of *universal* verification. When the distinction is potentially ambiguous, we will add a superscript to y for the binary verification labels: $y^{\text{verification}}$.

Unlike typical preference fine-tuning encodings, we do not require prefixes (or prompts) of $\boldsymbol{x}$ to be paired with different completions and opposing document-level labels. However, as in the above sections, we will assume that $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$ are balanced across y (i.e., an approximately equal number of documents with $y = 0$ and $y = 1$).

The SDM activation layer for verification will be trained with $\mathcal{D}_{\text{tr}}$, seeing all documents with $y = 0$ and $y = 1$ labels. However, the LLM's SDM activation for next-token training will only directly see documents with $y = 1$, with the signal of the `unverified` class coming indirectly via matching into $\mathcal{D}_{\text{tr}}$ to calculate Similarity and Distance. (There is an additional nuance with train-time generation vs. train-time force-decoding that will be clarified below.) As such, the additional SDM mechanisms enable a unification of preference fine-tuning, instruction fine-tuning, and supervised fine-tuning encodings since in all of the above, we always have at least the $y = 1$ documents, and it is typically straightforward to collect, or otherwise synthetically generate, unpaired examples to serve as $y = 0$ (i.e., generations we seek to avoid showing users).

4.3.2 Negative+Positive Vocabulary Normalization and Regularization

Before we can make progress on incorporating the SDM mechanisms, we need to address the matter of fine-tuning pre-trained LLMs without inducing catastrophic forgetting. This is critical, since each round of LLM training and fine-tuning is computationally expensive. We seek to make incremental changes to the model without having to run subsequent learning processes over all previously seen data. To address this, we first briefly recall the training of auto-regressive neural language models prior to the era of large-scale pre-training.

[L]LM Training Redux. Prior to the era of large-scale pre-training of LMs that emerged at the end of the 2010's, auto-regressive language models for transduction tasks (e.g., grammatical error correction) were successfully trained from random initialization using specialized input control tokens and output diff sequences (and associated output control tokens) that separated non-preferred (pre-transduction) and preferred (post-transduction) generated sequences (Schmaltz et al., 2017). Importantly, the bias on the diff control sequences could be modulated to control precision and recall over the absence and presence of the transduction operation (Schmaltz et al., 2016). In-effect, the sequence transduction model could be effectively used as a classifier without additional classification layers, while also having the expressivity to generate token sequences, unlike standard discrete classifiers.

Input and output control tokens are now prominent features of LLM vocabularies to structure prompts, instructions, and reasoning sequences. However, while the bias of individual tokens can be modified with an additive offset, current LLMs lack a mechanism to explicitly bifurcate the output distribution into non-preferred and preferred regions in the manner of the earlier models. This capability can be (re)-added to LLMs without direct training on diff transduction sequences, as follows.

Negative+Positive Vocabulary Normalization. Consider a pre-trained LLM model, $\mathcal{M}^{\text{ref}}$. Our reference model generates acceptable sequences over part of the data distribution, but it also produces non-preferred (NEGATIVE) generations; hence, our desire for further training. However, we only want to alter the behavior of $\mathcal{M}^{\text{ref}}$ over the space that produces NEGATIVE generations, otherwise we may unexpectedly cause the previously acceptable space of generations to also become NEGATIVE. In effect, we have two regions—a bifurcation—of the output distribution: The space of existing acceptable generations and the space of NEGATIVE generations. We seek to replace the NEGATIVE region with a new POSITIVE region of acceptable generations without (or at least minimally) impacting the existing acceptable region.

From $\mathcal{M}^{\text{ref}}$ create two clones, $\mathcal{M}^{\text{neg}}$ and $\mathcal{M}^{\text{pos}}$. Each model has a final linear layer that maps to the output vocabulary, $\mathcal{V}$, via a weight matrix[13]: $\boldsymbol{z}_{\text{ref}} = \boldsymbol{W}_{\text{ref}}^{T}\boldsymbol{h}_{\text{ref}}$, $\boldsymbol{z}_{\text{neg}} = \boldsymbol{W}_{\text{neg}}^{T}\boldsymbol{h}_{\text{neg}}$, and $\boldsymbol{z}_{\text{pos}} = \boldsymbol{W}_{\text{pos}}^{T}\boldsymbol{h}_{\text{pos}}$, respectively. During fine-tuning for the next-token loss, we then calculate the SDM activation (in-place of a standard softmax) as the concatenation of the un-normalized output of $\mathcal{M}^{\text{neg}}$ and $\mathcal{M}^{\text{pos}}$, $\text{SDM}(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})$, keeping the weights of $\mathcal{M}^{\text{neg}}$, $\boldsymbol{W}_{\text{neg}}$ and θ_{neg}, fixed and updating the weights of $\mathcal{M}^{\text{pos}}$, $\boldsymbol{W}_{\text{pos}}$ and θ_{pos}. For the $y = 1$ documents that participate in fine-tuning, we simply take the original token indexes and add an offset, $x_t + |\mathcal{V}|$, for the output tokens when calculating the loss over the joint, concatenated distribution. (Input tokens retain their original indexes.) At test-time, the $\arg\max$ output index mod $|\mathcal{V}|$ maps back to the original token symbol in the vocabulary. In this way, an additional set of token symbols is never explicitly instantiated.

In the most direct sense, this then requires a copy of the full weights to be present at test-time. However, in practice, $\mathcal{M}^{\text{pos}}$ need not be a copy of all the weights; $\mathcal{M}^{\text{pos}}$ can be represented by adaptor layers, or similar mechanisms (e.g., only updating a subset of the model's weights).

Regularization. To further prevent drift from the original reference distribution, we also add an L^2 regularization term in the $\log_{(2+q)}$ space of the normalized joint, concatenated distribution when calculating the next-token loss:

$$\text{r} = ||\boldsymbol{i} \odot \log_{(2+q)}\left(\text{SDM}(\boldsymbol{z}_{\text{ref}}, \boldsymbol{z}_{\text{ref}})\right) - \boldsymbol{i} \odot \log_{(2+q)}\left(\text{SDM}(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})\right)||_2 \tag{29}$$

where the Hadamard (element-wise) product ($\odot$) is with a mask vector $\boldsymbol{i} \in \mathbb{R}^{|\mathcal{V}|\cdot 2}$ that lessens the regularization on the peak of the distribution by not considering the $\arg\max$ indexes of the reference, negative, and positive distributions, as well as that of the ground-truth

[13] We ignore the bias terms here to simplify the presentation. In practice, it is not uncommon for $\boldsymbol{b} = \boldsymbol{0}$.

next-token label (here, represented as t), in the L^2 constraint:[14]

$$\begin{aligned} \boldsymbol{i} &= \mathbf{1} \in \mathbb{R}^{|\mathcal{V}|\cdot 2} \\ \boldsymbol{i}_{\arg\max\,(\boldsymbol{z}_{\text{ref}})} &= 0 \\ \boldsymbol{i}_{\arg\max\,(\boldsymbol{z}_{\text{ref}})+|\mathcal{V}|} &= 0 \\ \boldsymbol{i}_{\arg\max\,(\boldsymbol{z}_{\text{neg}})} &= 0 \\ \boldsymbol{i}_{\arg\max\,(\boldsymbol{z}_{\text{pos}})+|\mathcal{V}|} &= 0 \\ \boldsymbol{i}_t &= 0 \end{aligned} \tag{30}$$

We seek for our regularization term to be scaled relative to the loss, so we perform a simple re-scaling:

$$\begin{aligned} \mathrm{r}' &= \sqrt{\max(\mathrm{r}, 1)^{\min(\max(s,0),1)}}, \\ s &= \frac{\log_e \mathcal{L}(\boldsymbol{W}_{\text{pos}}, \theta_{\text{pos}}; \mathcal{D}_{\text{tr}})}{\log_e \mathrm{r}} \end{aligned} \tag{31}$$

After rescaling, r' is an additive term in the next-token training loss, described below. Next, we describe how sdm is calculated, and the structure of the network, more generally.

4.3.3 sdm network

The network makes use of two separate sdm activations. The first (verificationLayer) is over the binary verification task, trained at the document level. This is built as described in § 4.1, but specifically with an exemplar adaptor $g : \big(\text{mean}(\boldsymbol{h}_{\text{neg}}), \text{mean}(\boldsymbol{h}_{\text{pos}}), \boldsymbol{h}^{-1}_{\text{neg}}, \boldsymbol{h}^{-1}_{\text{pos}}\big) \in \mathbb{R}^{4D} \mapsto \boldsymbol{h'} \in \mathbb{R}^{M}$, trained over the concatenation of the mean of the final hidden states across tokens of both $\mathcal{M}^{\text{neg}}$ and $\mathcal{M}^{\text{pos}}$, as well as the hidden state (i.e., $\boldsymbol{h}^{-1}_{\text{neg}} \in \mathbb{R}^{D}$ and $\boldsymbol{h}^{-1}_{\text{pos}} \in \mathbb{R}^{D}$) that predicts the end of sequence delimiter[15], for which we use the superscript -1, all of which remain fixed when training the adaptor.[16] This has an associated sdm estimator, $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$, over the binary task (§ 4.2).

The second sdm activation is for normalizing the linear layer over the output vocabulary for next-token training, as described in § 4.3.2. In this case, the output Magnitude is determined by the concatenation of $(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})$, but the values of q and d are from the verificationLayer. In other words, for this second sdm activation, there is no exemplar adaptor inserted between the final hidden state of the LLM and the linear-layer over the vocabulary. This enables easily adapting this mechanism to existing architectures and pre-trained weights, if desired.

[14] This also accounts for the setting, not considered in our experiments, where $\mathcal{M}^{\text{neg}}$ is not identical to $\mathcal{M}^{\text{ref}}$, such as via multiple iterations of fine-tuning where $\mathcal{M}^{\text{neg}}$ is trained away and is replaced with $\mathcal{M}^{\text{pos}}$ for a subsequent fine-tuning round.

[15] This is the symbol that indicates the end of the sequence at the unit of analysis of the verification labels (e.g., the sentence or document level).

[16] In our small-scale experiments, we only train the final hidden layer of the LLM (i.e., θ_{pos} stays fixed, and we only update $\boldsymbol{W}_{\text{pos}}$), so we exclude the weights of $\mathcal{M}^{\text{neg}}$ as input to the exemplar adaptor, since they are identical to those of $\mathcal{M}^{\text{pos}}$.

SDM Network Next-token Loss. Holding the weights of the verificationLayer fixed, the next token loss to update the weights of $\mathcal{M}^{\text{pos}}$, $\boldsymbol{W}_{\text{pos}}$ and θ_{pos}, is then:

$$\mathcal{L}(\boldsymbol{W}_{\text{pos}}, \theta_{\text{pos}}; \mathcal{D}_{\text{tr}}, \beta, \mathcal{M}^{\text{ref}}) = -\frac{1}{N}\sum_{n}^{N} \log_{(2+q)}\left(\frac{(2+q)^{d\cdot z_{\text{neg},\text{pos}_{t_n}}}}{\sum_{v=1}^{|\mathcal{V}|\cdot 2}(2+q)^{d\cdot z_{\text{neg},\text{pos}_v}}}\right) + \beta\,\mathrm{r}' \tag{32}$$

where t_n is the index of the correct next token, and $\beta \in [0, \infty)$ linearly increases every mini-batch in an epoch from β_{min} (e.g., 0, in our experiments) to β_{max} (e.g., 0.1, in our experiments).

Train-time Generation vs. Train-time Force-decoding. The loss in Eq. 32 requires q and d, which are predicated on labels at the document-level, for each token prior to the model seeing the end of the document. In practice, for an $(\boldsymbol{x}, y = 1) \in \mathcal{D}_{\text{tr}}$, prior to calculating the loss, we decode a completion for $\boldsymbol{x}$ starting at the completion marker x_t (e.g., starting at the instruction prompt, or given prefix, as noted in § 4.3.1) with $q = e - 2, d = 1$. Then we derive q and d from the verificationLayer over this generated output. We otherwise discard the generated completion and calculate the loss using these updated values of q and d over the correct next token. (In the present work, q and d are the same for each token in a single document.) Note that the stored support set of the verificationLayer (which determines q and d) is constructed by force-decoding over $(\boldsymbol{x}, y = \{0, 1\}) \in \mathcal{D}_{\text{tr}}$. Thus, the loss has the desired semantics of rewarding the model to resemble the $y = 1$ data at the token-level (as in standard next-token fine-tuning), while penalizing generations that are challenging to verify.

SDM Network Training: verificationLayer + Next-token Loop. The next-token loss and the verificationLayer interact via q and d and the stopping criteria. However, the weight updates of each occur separately.

We seek the weights that maximize the admitted points over $\mathcal{D}_{\text{ca}}$ via $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ for $\hat{y} = 1$, and (if available), further restricting this set to those with correct y^{task} predictions (parsed from the generated text) for the underlying tasks encoded in the data.

The combined training loop is conceptually straightforward (Alg. 4). First, we construct the SDM estimator for binary verification (verificationLayer) via Alg. 1 by force-decoding over $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$. (The convention is to shuffle $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$ in the first training of the verificationLayer, itself a process over J iterations, and then use that final data split for all subsequent processes.) Next, we train one epoch of $\mathcal{M}^{\text{pos}}$. The next-token loss (Eq. 32) uses q and d from the verificationLayer over completions generated via greedy decoding (with $q = e - 2, d = 1$) using SDM$(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})$ starting at the completion marker.[17] Once the epoch concludes, we retrain the verificationLayer and update q and d for $\mathcal{D}_{\text{tr}}$. We then generate completions using SDM$(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})$ over $\mathcal{D}_{\text{ca}}$ and calculate the number of points for which $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ provides an index-conditional estimate for $\hat{y} = 1$, further restricted (if applicable) to the underlying task labels, y^{task}, and the predictions parsed for those tasks from the generated output. Next, we continue to the next epoch of updating $\mathcal{M}^{\text{pos}}$. This process continues until the max number of epochs has been reached.

[17] In practice, we cache q and d before each epoch, but in principle, they can be calculated dynamically during an epoch as the weights change. Caching simplifies the implementation at the expense of potentially biasing the estimates as an epoch proceeds, which is the motivation for increasing β through the course of an epoch.

Algorithm 4 SDM Network Training

```
Input: $\mathcal{D}_{\rm tr}$, $\mathcal{D}_{\rm ca}$, $\alpha'$, max epochs, $\mathcal{M}^{\rm ref}$, $\mathcal{M}^{\rm neg}$, $\mathcal{M}^{\rm pos}$
 1: procedure SDM-NETWORK-TRAIN($\mathcal{D}_{\rm tr}$, $\mathcal{D}_{\rm ca}$, $\alpha'$, max epochs, $\mathcal{M}^{\rm ref}$, $\mathcal{M}^{\rm neg}$, $\mathcal{M}^{\rm pos}$)
 2:     VerificationLayer, $\mathcal{D}_{\rm tr*}, \mathcal{D}_{\rm ca*}, \mathcal{E} \leftarrow$ SDM-ITERATIVE-TRAIN($\cdot$)    ▷ Alg. 4
 3:     $\mathcal{M}_* \leftarrow$ Initialized with $\mathcal{M}^{\rm neg}$, $\mathcal{M}^{\rm pos}$    ▷ Final trained model
 4:     metric$_* \leftarrow 0$    ▷ Determines final model
 5:     VerificationLayer$_* \leftarrow$ VerificationLayer    ▷ Final SDM activation layer for verification
 6:     $\mathcal{E}_* \leftarrow \mathcal{E}$    ▷ Final SDM estimator (i.e., $\hat{p}(y \,|\, \boldsymbol{x})_{\rm lower}$) for verification
 7:     $\beta_{\rm step} \leftarrow \frac{\beta_{\max} - \beta_{\min}}{\text{total mini batches}}$    ▷ Used to calculate $\beta$ as a function of epoch progress
 8:     Calculate $q, d$ for each $(\boldsymbol{x}, y = 1) \in \mathcal{D}_{\rm tr*}$ using VerificationLayer over generated output
        from SDM($\boldsymbol{z}_{\rm neg}, \boldsymbol{z}_{\rm pos}$) with $q = e - 2, d = 1$
 9:     for $e \in 1, \ldots,$ max epochs do
10:         Minimize $\mathcal{L}(\boldsymbol{W}_{\rm pos}, \theta_{\rm pos}; \mathcal{D}_{\rm tr}, \beta, \mathcal{M}^{\rm ref})$    ▷ Eq. 32
11:         VerificationLayer, _, _, $\mathcal{E} \leftarrow$ SDM-ITERATIVE-TRAIN($\cdot$) ▷ Without shuffling $\mathcal{D}_{\rm tr*}, \mathcal{D}_{\rm ca*}$
12:         Update $q, d$ for each $(\boldsymbol{x}, y = 1) \in \mathcal{D}_{\rm tr*}$    ▷ As in Line 8
13:         metric $\leftarrow$ cardinality of the admitted set from $\hat{p}(y \,|\, \boldsymbol{x})_{\rm lower}$ for $\hat{y} = 1$ over $\mathcal{D}_{\rm ca*}$    ▷
            Restricted to $y^{\rm task} = \hat{y}^{\rm task}$, if available
14:         if metric > metric$_*$ then
15:             metric$_* \leftarrow$ metric
16:             $\mathcal{M}_* \leftarrow$ Update with $\boldsymbol{W}_{\rm pos}, \theta_{\rm pos}$
17:             VerificationLayer$_* \leftarrow$ VerificationLayer
18:             $\mathcal{E}_* \leftarrow \mathcal{E}$
19:     return $\mathcal{M}_*, \mathcal{D}_{\rm tr*}, \mathcal{D}_{\rm ca*}$, VerificationLayer$_*, \mathcal{E}_*$
Output: $\mathcal{M}_*, \mathcal{D}_{\rm tr*}, \mathcal{D}_{\rm ca*}$, VerificationLayer$_*, \mathcal{E}_*$
```

SDM Network Test-time Generation. At test-time, we generate from SDM($\boldsymbol{z}_{\rm neg}, \boldsymbol{z}_{\rm pos}$) up to the output control token, or end-of-sequence token, at the unit-of-analysis of the verification labels, via greedy (i.e., arg max) decoding with $q = e - 2, d = 1$ (i.e., equivalent to softmax).[18] We then continue generation, or take other branching actions, based on $\hat{p}(y \,|\, \boldsymbol{x})_{\rm lower}$ from the VerificationLayer, which by extension, also provides interpretability-by-exemplar into $\mathcal{D}_{\rm tr}$ via matching (from q) and against similarly calibrated points in $\mathcal{D}_{\rm ca}$ via $\lfloor \tilde{q} \rfloor$. Each classification via the VerificationLayer requires on the order of the computation needed for commonly used dense retrieval augmentations of LLMs, so such test-time generation and verification is achievable even using edge devices.

5 Experiments

We comprehensively evaluate the uncertainty-awareness of our estimators across a representative set of the existing classes of estimators over LLMs. First, we compare SDM calibration to existing approaches in a standard classification setting, using open-source models at a scale that can be readily replicated with consumer-level compute (§ 5.1). Next, we show how an SDM estimator can be applied to a fully black-box LLM API, only with access to the top output logits and without a proxy model running in parallel, using the standard MMLU benchmark (§ 5.2). In this context, we also consider a data quality experiment in

[18] As previously noted, at test-time, $(\arg\max(\text{SDM}(\boldsymbol{z}_{\rm neg}, \boldsymbol{z}_{\rm pos})) \bmod |\mathcal{V}|)$ maps back to the original token symbol in the vocabulary when decoding over the joint distribution.

which we seek to detect errors in the carefully curated MMLU-Pro dataset. This serves as a natural, held-out blind evaluation of the estimator's capacity to separate aleatoric and epistemic uncertainty. Finally, we examine the universal verification behavior of an sdm network by training over a composition of the classification tasks examined in the first set of targeted experiments (§ 5.3).

5.1 Experiments: Classification

Before introducing the additional complications of LLM generation, we first isolate the core calibration behavior against existing classes of approaches in standard multi-class classification settings.

5.1.1 Task: Sentiment

Task. Our first task (Sentiment) is predicting the sentiment of movie reviews using the commonly used benchmark data of Maas et al. (2011). This is a binary classification task with $y \in \{0 = \text{negative}, 1 = \text{positive}\}$. $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$ are constructed from a total of 18k instances. The held-out set for evaluation, $|\,\mathcal{D}_{\text{te}}\,| = 1583$, is from the same distribution as $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$. This is a well-studied task for which the surface-level signals correlated with the target labels are expected to be effectively modeled by large parameter LLMs; as such, relatively high task accuracies are expected.

Models. Our base network, the parameters of which stay fixed and are used for all estimators, is the open-source, publicly available `Faster I` model from the on-device data analysis program Reexpress one[19] from Reexpress AI. This 1.2 billion-parameter model is a late fusion of the encoder and decoder of Flan-T5 large (Chung et al., 2022) and mT0-base (Muennighoff et al., 2023). We discard the existing adaptor layers that are part of the on-device program and only use the parameter fusion of the encoder and decoder, adding the adaptors and estimators introduced in this work. We take the mean of the hidden states across input tokens, resulting in a hidden state of $\boldsymbol{h} \in \mathbb{R}^{3774}$, as input to an exemplar adaptor, or an sdm activation layer, both with $M = 1000$. We use the label FasterI+adaptor for a standard exemplar adaptor over $\boldsymbol{h} \in \mathbb{R}^{3774}$ trained with a cross-entropy loss, and the label FasterI+sdm for the sdm activation layer over $\boldsymbol{h} \in \mathbb{R}^{3774}$.

Estimators. Holding the underlying network constant, we examine representative classes of estimators used with neural networks, seeking index-conditional calibration at $\alpha' = 0.95$. At the most basic, but also, perhaps the most commonly used in practice, representing the absence of a post-hoc calibration method, we simply threshold the output, $\text{softmax}(\boldsymbol{z}) \geq \alpha'$, where the temperature $\tau = 1$. As an established empirical approach for calibrating neural networks, we provide a comparison to temperature scaling (Guo et al., 2017), a single parameter version of post-hoc Platt-scaling (Platt, 1999), with the label tempScaling. In this case, the estimator is the thresholding of the output $\text{softmax}(\boldsymbol{z}; \tau) \geq \alpha'$ after learning a value for τ over $\mathcal{D}_{\text{ca}}$. We also provide a comparison to two representative conformal predictors, the APS method of Romano et al. (2020) and the adaptiveness-optimized RAPS algorithm of Angelopoulos et al. (2021). The admission criteria for the APS and RAPS estimators is prediction sets of size 1, using an $\alpha = 0.05$.

[19] `https://github.com/ReexpressAI/Reexpress_one`

We then compare to the primary SDM estimator $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$, as well as the reference comparisons $\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$, and $\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$, as defined in § 4.2.5. We train the SDM activation layer and estimator (Alg. 1) with $J = 10$, here and for the remaining experiments. Additional training hyper-parameters and details shared across all experiments are provided in Appendix A.3.

As a common point of reference, here and for all other experiments as well, we will use the label NO-REJECT to refer to the model predictions without any selective filtering (i.e., the raw output accuracies, either from a softmax or an SDM activation).

5.1.2 Task: SentimentOOD

To evaluate the behavior of the estimators over out-of-distribution data, we consider an additional task (SentimentOOD) that uses the same models and estimators as Sentiment, but an out-of-distribution evaluation set, $|\mathcal{D}_{\text{te}}| = 4750$. We use the SemEval-2017 Task 4a test set (Rosenthal et al., 2017), which consists of short-form social media posts that differ in the distribution of topics, language styles, and lengths relative to the movie reviews. We balance the test set, dropping the third class (neutral), setting the semantics of the true labels to be the same as that of the movie reviews: $y \in \{0 = \text{negative}, 1 = \text{positive}\}$.

5.1.3 Task: Factcheck

Task. As a more challenging binary classification task for LLMs, we consider the fact check data of Azaria and Mitchell (2023). The training and calibration sets, a combined total of 6k instances, consist of single sentence statements that have been semi-automatically generated via templates and a knowledge base. The task is to determine whether the statement is true or false, $y \in \{0 = \text{false}, 1 = \text{true}\}$. The held-out eval set, $|\mathcal{D}_{\text{te}}| = 245$, the focus of our analysis, has been constructed by having an LLM generate a statement continued from a true statement not otherwise in the dataset. These evaluation statements are checked manually and assigned labels by human annotators. In addition to being a relatively challenging task that evaluates—at least in principle—the latent knowledge stored within an LLM's parameters, the test set is representative of the types of distribution shifts over high-dimensional inputs that can be problematic for real applications, and challenging to characterize without model assistance and ground-truth labels. It was observed in Azaria and Mitchell (2023) that the accuracy of existing LLM classifiers is dramatically lower on this generated, held-out test set compared to the calibration set. However, these test sentences would seem to also be simple true-false statements, reflecting that it is not always immediately obvious for a human user to detect distribution shifts over high-dimensional inputs. As such, we seek for our models and estimators to reflect such shifts via the predictive uncertainty, as we will not, in general, have true labels at test-time.

Models and Estimators. Reflecting the more challenging task, our base network is the larger 3.2 billion parameter `Fast I` model from Reexpress one, which is a late fusion of the encoder and decoder of Flan-T5 xl and mT0-base. We additionally compose the `Fast I` model with `Mixtral 8x7B Instruct v0.1` (Jiang et al., 2024). This is achieved by constructing a simple re-ask verification prompt, and then a transform of the final layer of the `Mixtral` model and the output logits is concatenated to the mean of the hidden states across the input tokens of `Fast I`.[20] We use the label FastI+Mixtral+adaptor for a standard exemplar adaptor over the resulting $\boldsymbol{h} \in \mathbb{R}^{5854}$ trained with a cross-entropy loss, and the label FastI+Mixtral+sdm for the sdm activation layer over $\boldsymbol{h} \in \mathbb{R}^{5854}$. The estimators are otherwise the same as those used for the Sentiment task.

5.2 Experiments: Black-box LLM APIs

Next, we examine the behavior of the estimators when we only have access to a black-box API for an LLM that provides the generated text and the top-1 output log probabilities. In this context with a state-of-the-art model, we examine an additional class of estimators: Those that make use of uncertainty estimates explicitly encoded in the surface-level output vocabulary symbols. As a fully held-out test—and real-world use example—we also consider a data quality experiment in which we seek to uncover annotation errors in an existing carefully curated benchmark dataset.

5.2.1 Task: Question Answering

Task. Our evaluation is over the 4-choice question answering benchmark dataset MMLU (Hendrycks et al., 2021) and a 4-choice subset of the more challenging MMLU-Pro dataset (Wang et al., 2024)[21], for which we use the label MMLU-Pro-4qa. $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{ca}}$ are constructed from 102k instances from the `auxiliary_train`, `dev`, and `val` splits of MMLU and the MMLU-Pro validation set, the 4-choice subset of which only consists of 29 instances. For MMLU, $|\,\mathcal{D}_{\text{te}}\,| = 14042$. For MMLU-Pro-4qa, $|\,\mathcal{D}_{\text{te}}\,| = 5413$.

Models and Estimators. We use `gpt-4o-2024-08-06` (gpt-4o) (OpenAI et al., 2024) via the Microsoft Azure service[22] as the black-box LLM. Given the zero-shot question, the LLM is tasked with providing a structured response against the JSON Schema in Listing 1, and the top-1 log probability for each output token. The JSON is parsed for the answer letter, the surface-level symbol of which is the prediction for the no-reject estimator of gpt-4o. We consider the output probability for the answer letter, restricted to those estimates $\geq \alpha'$, as answerStringProb. The output JSON is also parsed for the model's real-valued verbalized uncertainty estimate, which when restricted to estimates $\geq \alpha'$, is the estimator verbalizedProb.[23]

As a final field, the output JSON also contains a short explanation for the response. We take the mean of the output probabilities corresponding to each value of the output JSON and concatenate those three values with a soft feature vector of length 4, where the activated index is that of the surface-level answer choice, for which we use verbalizedProb as the value, and all other indexes are 0. This length 7 vector than serves as $\boldsymbol{h} \in \mathbb{R}^{7}$ as input to an sdm activation layer with $M = 1000$. For the resulting gpt-4o+sdm model, we consider

[20] This process is the same as that described in the publicly available Reexpress one tutorial, but as with the Sentiment task, we discard the adaptors of the on-device application: `https://github.com/ReexpressAI/Example_Data/tree/main/tutorials/tutorial7_factcheck`.

[21] Both datasets are available via `https://huggingface.co/datasets`

[22] `https://azure.microsoft.com/en-us/`

[23] An additional class of estimators for black-box LLMs are those that require multiple test-time forward passes through the model, which are related to the Bayesian approaches of Gal and Ghahramani (2016, inter alia). We do not consider this class of approaches given the computational costs required of the estimators.

the NO-REJECT, $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$, $\hat{p}(y \,|\, \boldsymbol{x})_{\text{centroid}}$, and $\hat{p}(y \,|\, \boldsymbol{x})_{\text{upper}}$ estimators. Additional details appear in Appendix A.1.

5.2.2 Task: Data Quality Analysis

The MMLU-Pro dataset (MMLU-Pro-4qa) is a follow-up to the original MMLU benchmark designed to have more challenging questions and more reliable answer annotations. In the previously described experiment, we examine whether calibration can be maintained over this implied distribution shift. Separately, we consider here whether our method can uncover additional annotation errors, despite the relatively large amount of resources already spent to refine the dataset by the dataset constructors. MMLU-Pro reportedly underwent multiple rounds of review with experts and annotators, including LLM assistance for targeted error detection. We focus on the Computer Science category given that the questions should have unambiguous, objectively verifiable answers. This data quality test is a natural, fully held-out assessment of our approach compared to existing approaches used in practice, with direct, real-world applications. To do so we will examine the annotations among the set of admitted points sorted by $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ for which $y \neq \hat{y}$, where the desired behavior is for these points to reflect the aleatory uncertainty (exogenous to the model and estimator) of label annotation errors.

5.3 Experiments: Verified Generation

Next, given the context of the above experiments, we examine the behavior of the sdm network.

Task. We construct the verification task from the Sentiment, SentimentOOD, and Factcheck data described above (§ 5.1), taking the y labels of those earlier tasks as the y^{task} labels. The $y^{\text{verification}}$ (or simply y) labels (and associated instances) are constructed by synthetically inverting the text of the associated completions, as illustrated in Table 8. By design, under the assumption that it is a more challenging learning setting, we do not pair the completions. For example, given a single movie review, it will appear once as part of a user prompt and either the label $y^{\text{verification}} = 0$ or $y^{\text{verification}} = 1$, but not both.[24]

For analysis, we then have a standard binary classification task over the force-decoded output, $(\boldsymbol{x}, y^{\text{verification}} \in \{0, 1\}) \in \mathcal{D}_{\text{te}}$. We use the following labels for the corresponding datasets: SentimentVerification, with $|\, \mathcal{D}_{\text{te}} \,| = 1583$; SentimentOODVerification, with $|\, \mathcal{D}_{\text{te}} \,| = 4750$; and FactcheckVerification, with $|\, \mathcal{D}_{\text{te}} \,| = 245$. These test sets are useful for analyzing the behavior of the verificationLayer, but they do not reflect a real test-time scenario.

For final evaluation, we take the original test sets from Sentiment, SentimentOOD, and Factcheck (§ 5.1) and evaluate the output of the generated JSON for the underlying task labels, y^{task}, as in a standard evaluation of LLM output.

The corresponding system and user prompts appear in Listing 2. These design decisions enable examining the instruction-following setting across multiple underlying tasks while

[24] As noted in § 4.3, $y = 0$ instances (here, the constructed negatives) do not participate in next-token fine-tuning, but they are used for training the verificationLayer and the support set to determine q and d.

enabling reliable evaluation of verification, since there is no ambiguity (up to annotation errors in the original tasks) in y^{task} and $y^{\text{verification}}$, and we can readily parse the JSON output for the task predictions.[25]

Models and Estimators. For $\mathcal{M}^{\text{ref}}$ we use `Phi-3.5-mini-instruct` model (phi3.5) (Abdin et al., 2024), a 3.8 billion-parameter decoder-only Transformer model, via MLX (Hannun et al., 2023), version 0.21.1. To keep the experiments manageable at a level of compute that can be readily replicated on consumer hardware, while still being instructive for future larger-scale experiments, we only update the final linear-layer of $\mathcal{M}^{\text{pos}}$, $\boldsymbol{W}_{\text{pos}}$, in the next-token loss (Eq. 32); however, we update the full weight matrix of $\boldsymbol{W}_{\text{pos}}$ and not a lower-rank adaptor over these weights. This is instructive in this context, since our data is relatively small, but with $|\mathcal{V}| = 32064$ and the phi3.5 hidden dimension of 3072, the 100 million parameters of $\boldsymbol{W}_{\text{pos}}$ would be assumed to quickly overfit, leading to degenerate output. Because we only update $\boldsymbol{W}_{\text{pos}}$, while θ_{pos} stays fixed, we only need to train the verificationLayer once before the next-token training loop begins (i.e., Line 11 in Alg. 4 is not needed), and we exclude the weights of $\mathcal{M}^{\text{neg}}$ as input to the sdm activation layer, since they are identical to those of $\mathcal{M}^{\text{pos}}$. As such, the input to the sdm activation of the verificationLayer is $\left(\text{mean}(\boldsymbol{h}_{\text{pos}}), \boldsymbol{h}_{\text{pos}}^{-1}\right) \in \mathbb{R}^{2\cdot 3072}$, the concatenation of the average of the final hidden states (across tokens) with the final hidden state that predicts the end of sequence delimiter (here, the final closing bracket in the JSON output).

In this setting, our primary comparison is against the full model before fine-tuning, for which we use the label phi3.5+sdm. In this case, only the verificationLayer layer is trained, here for $J = 10$ iterations of 50 epochs, but the evaluation is still over completions generated via greedy decoding (i.e., arg max) over $\text{sdm}(\boldsymbol{z}_{\text{neg}}, \boldsymbol{z}_{\text{pos}})$ with $q = e - 2, d = 1$. The fine-tuned model (phi3.5+sdmNetwork) uses this same verificationLayer, but it is also trained for 5 epochs with $\beta_{\min} = 0$ and $\beta_{\max} = 0.1$ using the next-token loss of Alg. 4. We choose the model weights (as in Line 13 of Alg. 4) as those that maximize the count of admitted points over $\mathcal{D}_{\text{ca}}$ via $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$ for $\hat{y} = 1$, further restricted to $y^{\text{task}} = \hat{y}^{\text{task}}$, which is determined by parsing the generated JSON output. For both models, phi3.5+sdm and phi3.5+sdmNetwork, we consider the no-reject and $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$ estimators.[26]

6 Results

Across tasks and models, the sdm calibration process yields an estimator that achieves the desired notion of index-conditional calibration (Def. 3), in contrast to the existing classes of estimators over LLMs, which become unreliable in the presence of even modest distribution shifts. The $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$ estimator remains calibrated in the presence of distribution shifts due to the $\tilde{q}^{\gamma}_{\min}$ lower constraint on $\tilde{q}_{\text{lower}}$, which screens points that are unlike those seen during the calibration process. With existing methods, defining an out-of-distribution point has been task- and problem-specific, and generally challenging over high-dimensional inputs. In contrast, the sdm calibration process provides a principled approach for determining such cut-offs in a data- and model-driven manner, with minimal hyper-parameters, resulting in a clear separation of points over which the estimator is reliable (namely, the admitted points)

[25] Ill-formatted JSON output is treated as a wrong prediction.

[26] In this context, $\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$ and $\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$ are less meaningful as a comparison since $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$ is used as part of the aforementioned stopping criteria for phi3.5+sdmNetwork, and are thus excluded.

Table 1: Comparison of relevant estimators for the standard document classification setting, $\alpha' = 0.95$. N/A indicates all predictions were rejected, which is preferred over falling under the expected accuracy. $n = \lvert \text{Admitted} \rvert$, the count of non-rejected documents.

			Class-conditional				Prediction-conditional				Marginal	
			$y = 0$		$y = 1$		$\hat{y} = 0$		$\hat{y} = 1$		$y \in \{0, 1\}$	
Dataset	Model	Estimator	Acc.	$\frac{n}{\lvert \mathcal{D}_{\text{te}} \rvert}$	Acc.	$\frac{n}{\lvert \mathcal{D}_{\text{te}} \rvert}$	Acc.	$\frac{n}{\lvert \mathcal{D}_{\text{te}} \rvert}$	Acc.	$\frac{n}{\lvert \mathcal{D}_{\text{te}} \rvert}$	Acc.	$\frac{n}{\lvert \mathcal{D}_{\text{te}} \rvert}$
Sentiment	FasterI+adaptor	no-reject	0.982	0.50	0.953	0.50	0.955	0.51	0.982	0.49	0.968	1.
Sentiment	FasterI+adaptor	softmax	0.995	0.46	0.983	0.41	0.985	0.46	0.994	0.41	0.989	0.87
Sentiment	FasterI+adaptor	tempScaling	0.994	0.45	0.986	0.39	0.987	0.45	0.994	0.39	0.990	0.84
Sentiment	FasterI+adaptor	APS	0.993	0.47	0.973	0.45	0.975	0.48	0.993	0.44	0.983	0.92
Sentiment	FasterI+adaptor	RAPS	0.989	0.47	0.972	0.44	0.974	0.48	0.988	0.44	0.981	0.92
Sentiment	FasterI+sdm	no-reject	0.971	0.50	0.966	0.50	0.966	0.50	0.971	0.50	0.968	1.
Sentiment	FasterI+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	0.996	0.32	0.996	0.32	0.996	0.32	0.996	0.32	0.996	0.65
Sentiment	FasterI+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	0.996	0.36	0.993	0.35	0.993	0.36	0.996	0.35	0.995	0.71
Sentiment	FasterI+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	0.997	0.38	0.993	0.37	0.993	0.38	0.997	0.37	0.995	0.75
SentimentOOD	FasterI+adaptor	no-reject	0.992	0.5	0.394	0.5	0.621	0.80	0.979	0.20	0.693	1.
SentimentOOD	FasterI+adaptor	softmax	1.	0.37	0.251	0.08	0.854	0.44	1.	0.02	0.861	0.46
SentimentOOD	FasterI+adaptor	tempScaling	1.	0.34	0.223	0.07	0.869	0.39	1.	0.01	0.874	0.41
SentimentOOD	FasterI+adaptor	APS	1.000	0.43	0.346	0.19	0.770	0.55	0.997	0.07	0.795	0.62
SentimentOOD	FasterI+adaptor	RAPS	0.999	0.43	0.336	0.20	0.761	0.56	0.991	0.07	0.786	0.63
SentimentOOD	FasterI+sdm	no-reject	0.570	0.5	0.966	0.5	0.944	0.30	0.692	0.70	0.768	1.
SentimentOOD	FasterI+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	N/A	0.	N/A	0.	N/A	0.	N/A	0.	N/A	0.
SentimentOOD	FasterI+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	N/A	0.	N/A	0.	N/A	0.	N/A	0.	N/A	0.
SentimentOOD	FasterI+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	0.	0.06	1.	0.28	N/A	0.	0.819	0.35	0.819	0.35
Factcheck	FastI+Mixtral+adaptor	no-reject	0.365	0.51	0.908	0.49	0.807	0.23	0.574	0.77	0.629	1.
Factcheck	FastI+Mixtral+adaptor	softmax	0.211	0.08	0.975	0.33	0.667	0.02	0.839	0.38	0.828	0.40
Factcheck	FastI+Mixtral+adaptor	tempScaling	0.286	0.06	0.987	0.31	0.8	0.02	0.884	0.35	0.879	0.37
Factcheck	FastI+Mixtral+adaptor	APS	0.283	0.19	0.979	0.38	0.867	0.06	0.736	0.51	0.75	0.57
Factcheck	FastI+Mixtral+adaptor	RAPS	0.341	0.18	0.967	0.37	0.833	0.07	0.75	0.47	0.761	0.55
Factcheck	FastI+Mixtral+sdm	no-reject	0.397	0.51	0.899	0.49	0.806	0.25	0.585	0.75	0.641	1.
Factcheck	FastI+Mixtral+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	N/A	0.	1.	0.13	N/A	0.	1.	0.13	1.	0.13
Factcheck	FastI+Mixtral+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	N/A	0.	1.	0.17	N/A	0.	1.	0.17	1.	0.17
Factcheck	FastI+Mixtral+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	1.	0.02	0.980	0.21	0.8	0.02	1.	0.20	0.982	0.22

Table 2: MAD and $\mathrm{m}^{\hat{y}}_{\lfloor\tilde{q}\rfloor}$ by $\lfloor\tilde{q}\rfloor$ on $\mathcal{D}_{\mathrm{ca}}$ for the standard classification tasks, trained with $J = 10$ iterations, each of 50 epochs. As $\lfloor\tilde{q}\rfloor$ increases, the epistemic uncertainty decreases, and the variation among comparable points also decreases.

	SENTIMENT				FACTCHECK			
	$y = 0$		$y = 1$		$y = 0$		$y = 1$	
$\lfloor\tilde{q}\rfloor$	MAD	$\mathrm{m}^{0}_{\lfloor\tilde{q}\rfloor}$	MAD	$\mathrm{m}^{1}_{\lfloor\tilde{q}\rfloor}$	MAD	$\mathrm{m}^{0}_{\lfloor\tilde{q}\rfloor}$	MAD	$\mathrm{m}^{1}_{\lfloor\tilde{q}\rfloor}$
0	0.007	0.044	0.007	0.044	0.024	0.148	0.018	0.116
1	< 0.001	0.006	< 0.001	0.003	0.009	0.056	0.004	0.024
2	< 0.001	< 0.001	< 0.001	< 0.001	0.003	0.021	0.001	0.004
3	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	0.004	< 0.001	0.002
4	0.	0.	0.	0.	< 0.001	0.001	< 0.001	< 0.001
5	0.	0.	0.	0.	< 0.001	< 0.001	< 0.001	< 0.001
6	0.	0.	0.	0.	< 0.001	< 0.001	0.	0.
7	0.	0.	0.	0.	-	-	-	-

Table 3: $\tilde{q}^{\,\gamma}_{\mathrm{min}}$ on $\mathcal{D}_{\mathrm{ca}}$ for the standard multi-class classification experiments. The more challenging FACTCHECK task has a commensurately higher $\tilde{q}^{\,\gamma}_{\mathrm{min}}$.

SENTIMENT		FACTCHECK	
MAD	$\tilde{q}^{\,\gamma}_{\mathrm{min}}$	MAD	$\tilde{q}^{\,\gamma}_{\mathrm{min}}$
7.9e-05	1.004	0.100	2.447

and those over which the estimates themselves are unreliable (i.e., the rejected points). The SDM network takes this behavior to its logical conclusion by incorporating it into the LLM architecture and fine-tuning process to serve as a universal verifier, suggesting a principled basis for building large, complex LLM systems and pipelines that are both interpretable and reliable.

6.1 Results: Classification

Table 1 displays the results for the binary classification tasks. The results for SENTIMENT vs. those of the other datasets are indicative of the under-appreciated point in the existing calibration literature of the importance of comparisons over—at least modest—distribution-shifts. On in-distribution benchmark data with high accuracy models, the differences can be difficult to discern; after all, the class-wise accuracy of the model is itself $\geq \alpha'$. However, even in these otherwise straightforward binary classification settings, the existing classes of estimators all but fall apart in the presence of distribution shifts, which are common in practice with high-dimensional data, such as text. In this light, the existing classes of estimators are not demonstrably more effective than simply using an un-calibrated threshold on the output (SOFTMAX). In contrast, the $\hat{p}(y \mid \boldsymbol{x})_{\mathrm{lower}}$ estimator achieves index-conditional calibration in all cases, correctly rejecting documents over which the estimates are unreliable, and admitting points for which the class- and prediction-conditional accuracies are $\geq \alpha'$.

Table 4: Comparison of relevant estimators combined with gpt-4o, $\alpha' = 0.95$. The sdm estimator, $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$, remains well-calibrated even over the much more challenging MMLU-Pro-4qa dataset. Importantly, $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ is not vacuously conservative; the yield of admitted points is higher on MMLU even when the verbalized uncertainty of gpt-4o is well-calibrated (see underline).

Dataset	Model	Estimator	Acc.	$\frac{\lvert\text{Admitted}\rvert}{\lvert\mathcal{D}_{\text{te}}\rvert}$
MMLU	gpt-4o	no-reject	0.832	1.
MMLU	gpt-4o	answerStringProb	0.921	0.74
MMLU	gpt-4o	verbalizedProb	0.953	0.35
MMLU	gpt-4o+sdm	no-reject	0.835	1.
MMLU	gpt-4o+sdm	$\hat{p}(y \,\|\, \boldsymbol{x})_{\text{lower}}$	0.957	0.38
MMLU	gpt-4o+sdm	$\hat{p}(y \,\|\, \boldsymbol{x})_{\text{centroid}}$	0.956	0.39
MMLU	gpt-4o+sdm	$\hat{p}(y \,\|\, \boldsymbol{x})_{\text{upper}}$	0.954	0.41
MMLU-Pro-4qa	gpt-4o	no-reject	0.648	1.
MMLU-Pro-4qa	gpt-4o	answerStringProb	0.870	0.51
MMLU-Pro-4qa	gpt-4o	verbalizedProb	0.857	0.16
MMLU-Pro-4qa	gpt-4o+sdm	no-reject	0.683	1.
MMLU-Pro-4qa	gpt-4o+sdm	$\hat{p}(y \,\|\, \boldsymbol{x})_{\text{lower}}$	0.958	0.22
MMLU-Pro-4qa	gpt-4o+sdm	$\hat{p}(y \,\|\, \boldsymbol{x})_{\text{centroid}}$	0.957	0.23
MMLU-Pro-4qa	gpt-4o+sdm	$\hat{p}(y \,\|\, \boldsymbol{x})_{\text{upper}}$	0.942	0.24

Central to the unique behavior of the sdm estimator is that the epistemic uncertainty decreases as $\tilde{q}$ increases. Furthermore, $\lfloor \tilde{q} \rfloor$ can be used as a mapping between $\mathcal{D}_{\text{ca}}$ and a new, unseen test point, because the variation among comparable points also decreases as $\tilde{q}$ increases. Table 2 shows this for the standard multi-class classification tasks with summary statistics over the $J = 10$ iterations. The corresponding $\tilde{q}^{\,\gamma}_{\text{min}}$ used by the $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ estimator (Eq. 28) appears in Table 3. Comparing these $\tilde{q}^{\,\gamma}_{\text{min}}$ values with Table 2 makes it clear that the $\tilde{q}^{\,\gamma}_{\text{min}}$ values are effectively change points w.r.t. the epistemic uncertainty: Points below have high variation and points above have increasingly low variation to the point that $\text{m}^{\hat{y}}_{\lfloor \tilde{q} \rfloor}$ reaches 0, within numerical error.

This behavior is remarkable for an estimator over high-dimensional inputs, because it demonstrates there are regions of the distribution that are low variation and high-probability that can be reliably detected. Existing estimators marginalize over the distinctions in these regions, which can cause unexpected behavior at test-time, as demonstrated in our empirical results.

6.2 Results: Black-box LLM APIs

Table 4 contains the results of the estimators over gpt-4o, the baseline accuracy of which is in-line with existing reported results for the zero-shot setting, and gpt-4o+sdm. Neither answerStringProb nor verbalizedProb are reliable estimators across these datasets, even though the multiple-choice QA task is a common setting for LLM development and evaluation. Conceptually, both can be viewed as encoding the output Magnitude, without explicitly controlling for the Similarity and Distance, as with a softmax estimator in a standard classification setting. Their over-confidence on MMLU-Pro-4qa reflect this.

The results of $\hat{p}(y \,|\, \boldsymbol{x})_{\text{lower}}$ on MMLU-Pro-4qa are indicative of the real-world use of the sdm estimator. gpt-4o has a dramatically lower overall accuracy on the MMLU-Pro-4qa

Table 5: Verified generation results, $\alpha' = 0.95$. Task datasets are identical to those in Table 1. Predictions are parsed from the JSON *generated* by the model, with parsing errors counted as wrong predictions. N/A indicates all predictions were rejected, which is preferred over falling under the expected accuracy. Verification via an sdm estimator is reliable regardless of fine-tuning the model, but fine-tuning with sdm (phi3.5+sdmNetwork) can increase the task accuracy (see **bold**) and the yield of admitted points (see underline).

Dataset	Model	Estimator	Acc.	$\frac{\lvert\text{Admitted}\rvert}{\lvert\mathcal{D}_{\text{te}}\rvert}$
Sentiment	phi3.5+sdm	no-reject	0.751	1.
Sentiment	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	0.997	$\underline{0.39}$
Sentiment	phi3.5+sdmNetwork	no-reject	**0.876**	1.
Sentiment	phi3.5+sdmNetwork	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	0.996	$\underline{0.42}$
SentimentOOD	phi3.5+sdm	no-reject	0.815	1.
SentimentOOD	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	1.	<0.01
SentimentOOD	phi3.5+sdmNetwork	no-reject	**0.896**	1.
SentimentOOD	phi3.5+sdmNetwork	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	1.	<0.01
Factcheck	phi3.5+sdm	no-reject	0.706	1.
Factcheck	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	0.973	0.15
Factcheck	phi3.5+sdmNetwork	no-reject	**0.743**	1.
Factcheck	phi3.5+sdmNetwork	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	0.973	0.15

questions, which would come as a surprise to an end-user who was expecting behavior similar to that over MMLU. In contrast, the $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$ estimator remains calibrated. For the rejected documents, the user would then know to take additional action. Alternatively, if part of an automated pipeline, additional test-time compute-based branching decisions (such as re-asking the model, or seeking outside information via retrieval) could be taken in the background before presenting a final result.

Data Quality Analysis. For MMLU-Pro-4qa, we examine the 5 questions in the Computer Science category that were in the $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$ index-conditional admitted set, but for which the predicted answers do not match the ground-truth annotations, $y \neq \hat{y}$. The top 4 questions sorted by $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$, all of which have $\hat{p}(y \mid \boldsymbol{x})_{\text{lower}} \geq 0.99$, all clearly have annotation errors where the model predictions are correct and the ground-truth annotations are incorrect. We include the question id's in Table 6. This provides an exogenous evaluation of the method: The sdm estimator has successfully separated the aleatoric and epistemic uncertainty among the high-probability predictions.

6.3 Results: Verified Generation

The results for the sdm network indicate effective verification of instruction following (Table 5). Our small-scale experiment confirms that the verificationLayer reliably yields a calibrated estimator regardless of fine-tuning, but the fine-tuning process improves overall task accuracy. That is, the results confirm that Alg. 4, which chose epoch 3 of 5 as the final model, is a viable fine-tuning loss and process. Importantly in this context, the cardinality of the set of admitted points is non-decreasing relative to before fine-tuning, despite updating 100 million parameters on a small training set. Leveraging the behavior of the sdm estimator, the sdm network is, in this way, the first statistically principled and robust approach to construct an LLM with an intrinsic ability to verify its own instruction-following and generated output.

7 Conclusion

There has been renewed interest in deep learning as a focus of research for language modeling over the last decade, and a growing number of efforts to scale data and model compute for various applications. However, brittleness to distribution shifts and lack of reliable uncertainty quantification have precluded—or otherwise diminished the potential of—the use of neural network language models in most real-world settings. In this work, we have addressed these foundational limitations by introducing SDM activation functions, SDM calibration, and SDM networks.

Appendix A.

We provide additional experimental details and results for the black-box LLM API experiments in § A.1 and the verified generation experiments in § A.2. Additional training details are included in § A.3.

Code to replicate our results is available at the URL provided in the main text. For the reader, we provide a few key highlights here. We include an implementation of the SDM activation function in § A.4. We provide our conventions for calculating empirical CDFs in § A.5, and we provide code scaffolding for an example implementation of an SDM network training loop in § A.6.

A.1 Black-box LLM APIs

The results of the data quality analysis are included in Table 6. Following best practices, to avoid contaminating the test set since research articles are commonly used for LLM training, we only include the question id's and not the question and answer text, which can readily be retrieved from the Huggingface datasets database.

We include the prompts used for the experiments in the code repo. The prompt is a variation on the theme of that used in OpenAI's Simple Evals repo[27], with the addition of using structured outputs against the JSON Schema in Listing 1. The particular prompt and structuring of the JSON (and parsing of the JSON, described below) are not defining aspects of the approach and are not necessarily the optimal templates. We use a direct, zero-shot approach to examine the more challenging setting—arguably closer to real-world usage—than providing examples or systematically hill-climbing on prompts.

The embedding for input to the SDM activation layer is constructed by parsing the JSON schema mapped back to the top-1 probabilities of the output tokens. For each key, we average the log-probabilities in probability space of the tokens of the corresponding value. For example, for the key `"short_explanation_for_answer_confidence"`, we parse the output to isolate the tokens corresponding to the value, and take the average of the exponentiated log probabilities of the tokens. Given the 3 keys in the JSON schema, this results in 3 floating-point values. (The verbalized uncertainty object `"confidence_in_answer_letter"` has a value of type `number`, but the output itself corresponds to a sequence of discrete tokens (e.g., "`0`", "`.`", "`9`"), so this parsing process is the same as that for the values of type `string`.) Finally, we construct a soft one-hot vector of length 4 where the non-zero index (if any) of the predicted letter is set to the floating-point value of the verbalized uncertainty (i.e., the value of the object with key `"confidence_in_answer_letter"`). The input embedding is then the concatenation of these 7 values. Full refusals from the LLM's API, which are rare but can occur on some of the social science and humanities questions, are assigned vectors of 0's as embeddings for $\mathcal{D}_{\mathrm{tr}}$ and $\mathcal{D}_{\mathrm{ca}}$ instances, and treated as wrong predictions in the test evaluations.

The estimator ANSWERSTRINGPROB corresponds to the index of this embedding derived from the value of the object with key `"answer_letter"`. Often this is the probability of the single token (i.e., "`A`", "`B`", "`C`", "`D`"), but occasionally will be the average over additional

[27] `https://github.com/openai/simple-evals/blob/main/common.py`

Table 6: MMLU-PRO-4QA, Computer Science category. Predictions that met the index-conditional threshold but were marked incorrect according to the ground-truth labels. Examination of the data reveals the model is correct and the ground-truth annotations are incorrect . The digit significance of $\hat{p}(y \mid \boldsymbol{x})$ is not necessarily significant (and when shown to users, would typically be rounded, with a top ceiling to avoid 1.0), but provided for reference. $\hat{n}_{\hat{y}}$ is the effective sample size for the predicted class. The final question is arguably ambiguous.

Question ID	y	$\hat{y}$	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	$\hat{n}_{\hat{y}}$
10750	A	D	0.9999999029119694	0.999999946869715	0.999999963752475	11563
10682	D	C	0.9999995410050875	0.9999997504521737	0.9999998413548795	11774
10458	D	A	0.9997548501324156	0.9998610348657851	0.9999170919091862	9129
10533	B	C	0.9897059289074643	0.9936086673736274	0.9957749405311342	6891
10479	D	B	0.967751071803557	0.9791966686070331	0.9862756083406558	7684

tokens (e.g., "$"). The estimator VERBALIZEDPROB corresponds to the floating-point value of the verbalized uncertainty.

In our experiments, we aim for a controlled comparison with ANSWERSTRINGPROB and VERBALIZEDPROB; as such, the SDM activation layer is only given access to the 7 values above. In particular, we do not provide access to additional signal derived from composition with another model. In applications where the uncertainty is over multiple tasks (i.e., not just question answering of this particular format), to avoid a marginalization over tasks, we recommend either encoding the distinction across tasks in the JSON schema, or simply concatenating the LLM output with the hidden states of another large model. The latter is typically readily achievable by running another open-source model alongside the black-box LLM's API.

We train the SDM activation layer as a 4-class classification task, which is an effective but potentially sample-inefficient encoding, at least when assuming the absence of artifacts correlated with answer letters. An alternative would be to re-encode the task as binary classification, either as a leave-one-out classification or as binary verification (as in § 5.3). Since the choice of encoding, as with the structure of the prompt and JSON Schema, is orthogonal to the evaluation of the uncertainty estimates—other than with respect to effective sample sizes—we keep these aspects straightforward in this set of experiments to avoid complicating the presentation.

Given the results in the main text, a logical next step would be to use this behavior to build a *re-ask* pipeline. That is, predictions with low probability can be automatically routed to re-prompt the LLM conditional on the previous response, a potentially effective means of building test-time compute systems over otherwise black-box models. Such pipelines are not feasible without robust estimates of predictive uncertainty, but become conceptually straightforward—and straightforward to implement—given the behavior of SDM estimators. We leave such additional applied examples for future work to systematically analyze.

Table 7: Verification results on the *force-decoded test sets* for reference, $\alpha' = 0.95$. See Table 5 for generation results for the underlying tasks, which reflect real test-time usage. N/A indicates all predictions were rejected, which is preferred over falling under the expected accuracy. $n = |\text{Admitted}|$, the count of non-rejected documents. Additional resolution added to $\frac{n}{|\mathcal{D}_{\text{te}}|}$ columns for SentimentOODVerification for reference, but the number of admitted points is effectively 0.

			Class-conditional				Prediction-conditional				Marginal	
			$y = 0$		$y = 1$		$\hat{y} = 0$		$\hat{y} = 1$		$y \in \{0, 1\}$	
Dataset	Model	Estimator	Acc.	$\frac{n}{\lvert\mathcal{D}_{\text{te}}\rvert}$	Acc.	$\frac{n}{\lvert\mathcal{D}_{\text{te}}\rvert}$	Acc.	$\frac{n}{\lvert\mathcal{D}_{\text{te}}\rvert}$	Acc.	$\frac{n}{\lvert\mathcal{D}_{\text{te}}\rvert}$	Acc.	$\frac{n}{\lvert\mathcal{D}_{\text{te}}\rvert}$
SentimentVerification	phi3.5+sdm	no-reject	0.959	0.51	0.891	0.49	0.901	0.54	0.954	0.46	0.925	1.
SentimentVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	0.996	0.17	0.997	0.21	0.996	0.17	0.997	0.21	0.997	0.38
SentimentVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	0.996	0.18	0.997	0.22	0.996	0.18	0.997	0.22	0.997	0.40
SentimentVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	0.997	0.19	0.997	0.23	0.997	0.19	0.997	0.23	0.997	0.42
SentimentOODVerification	phi3.5+sdm	no-reject	0.978	0.51	0.639	0.49	0.738	0.68	0.966	0.32	0.812	1.
SentimentOODVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	1.	0.002	1.	0.0002	1.	0.002	1.	0.0002	1.	0.002
SentimentOODVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	1.	0.003	1.	0.0002	1.	0.003	1.	0.0002	1.	0.003
SentimentOODVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	1.	0.003	1.	0.0004	1.	0.003	1.	0.0004	1.	0.004
FactcheckVerification	phi3.5+sdm	no-reject	0.656	0.50	0.732	0.50	0.708	0.46	0.682	0.54	0.694	1.
FactcheckVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{lower}}$	N/A	0.	1.	0.07	N/A	0.	1.	0.07	1.	0.07
FactcheckVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{centroid}}$	N/A	0.	1.	0.08	N/A	0.	1.	0.08	1.	0.08
FactcheckVerification	phi3.5+sdm	$\hat{p}(y \mid \boldsymbol{x})_{\text{upper}}$	N/A	0.	1.	0.08	N/A	0.	1.	0.08	1.	0.08

Listing 1: JSON Schema for gpt-4o Structured Outputs.

```
{
    "properties": {
        "answer_letter": {
            "title": "Answer Letter",
            "type": "string"
        },
        "confidence_in_answer_letter": {
            "title": "Confidence In Answer Letter",
            "type": "number"
        },
        "short_explanation_for_answer_confidence": {
            "title": "Short Explanation For Answer Confidence",
            "type": "string"
        }
    },
    "required": [
        "answer_letter",
        "confidence_in_answer_letter",
        "short_explanation_for_answer_confidence"
    ],
    "title": "MultipleChoiceQuestionResponse",
    "type": "object"
}
```

A.2 Verified Generation

For reference, Table 7 provides the effectiveness over the force-decoded datasets. The support set of the verificationLayer is constructed from the force-decoded training and calibration data, so this table reflects the held-out classification ability over the verification data, which includes constructed negatives for $y^{\text{verification}} = 0$, as described in the main text and illustrated in Table 8. Listing 2 includes the system message and prompts used for the experiments.

Table 8: JSON structure for the verified generation experiments, with $\mathcal{M}^{\text{ref}} = \text{phi3.5}$. $y^{\text{verification}} = 1$ corresponds to the standard classification tasks, where, e.g., $y^{\text{task}} = 0$ corresponds to a negative review for the sentiment task, and $y^{\text{task}} = 1$ corresponds to a factually correct statement for the factcheck task. $y^{\text{verification}} = 0$ flips the parity, and is used for constructing negatives for training, and the contrastive basis for rejection at test-time. Recall that the LLM takes as input a system prompt, user prompt, and the document (see Listing 2). At test-time, we seek to generate the correct JSON output (i.e., that corresponding to the correct y^{task} label), for instances with $\hat{y}^{\text{verification}} = 1$ predicted by the verificationLayer layer.

Datasets	Labels	JSON output
Sentiment, SentimentVerification SentimentOOD, SentimentOODVerification		
	$y^{\text{task}} = 0, y^{\text{verification}} = 1$	`{"sentiment": "negative"}`
	$y^{\text{task}} = 1, y^{\text{verification}} = 1$	`{"sentiment": "positive"}`
	$y^{\text{task}} = 0, y^{\text{verification}} = 0$	`{"sentiment": "positive"}`
	$y^{\text{task}} = 1, y^{\text{verification}} = 0$	`{"sentiment": "negative"}`
Factcheck, FactcheckVerification		
	$y^{\text{task}} = 0, y^{\text{verification}} = 1$	`{"correctness": false}`
	$y^{\text{task}} = 1, y^{\text{verification}} = 1$	`{"correctness": true}`
	$y^{\text{task}} = 0, y^{\text{verification}} = 0$	`{"correctness": true}`
	$y^{\text{task}} = 1, y^{\text{verification}} = 0$	`{"correctness": false}`

Listing 2: System and user messages for the sentiment and factcheck datasets of the verified generation experiments, with $\mathcal{M}^{\text{ref}} = \text{phi3.5}$. The document text replaces TEXT for each instance.

```
<|system|>
You are a helpful AI assistant.<|end|>
<|user|>
Classify the sentiment of the following movie review. Respond using the following JSON: {"sentiment": str}. REVIEW: TEXT<|end|>
<|assistant|>

<|system|>
You are a helpful AI assistant.<|end|>
<|user|>
Check the following document for hallucinations and/or factual inaccuracies. Respond using the following JSON: {"correctness": bool}.
      DOCUMENT: TEXT<|end|>
<|assistant|>
```

A.3 Additional Training Details

Compute. The black-box LLM experiments require API calls, as detailed in the main text, but all other results can be reproduced locally on a single 2023 Mac Studio with an M2 Ultra chip with 128 GB of unified memory. These experiments are designed to fully assess the methods while still being replicable with consumer hardware.

Hyper-parameters. In the code repo, we include scripts for replicating our results. For all cases, we train the rescaling transform (Alg. 2) for up to 1000 epochs, with early stopping if the loss exceeds the min observed loss for 10 consecutive epochs. In all experiments, $M = 1000$ and we use a mini-batch size of 50. We mean center the input to g, the 1-D CNN of the sdm activation layer, via the mean and standard deviation over $\mathcal{D}_{\text{tr}}$. We train gpt-4o+sdm for

$J = 10$ iterations of 5 epochs, and the sdm models of Sentiment and Factcheck, as well as the verificationLayer of the sdm network, for $J = 10$ iterations of 50 epochs. The standard exemplar adaptors of the Sentiment and Factcheck classification experiments are trained with cross-entropy losses for 50 epochs. We use the Adam optimizer (Kingma and Ba, 2017) with a learning rate of 1×10^{-4} for training the rescaling transform (Alg. 2) and 1×10^{-5} for all other cases.

A.4 Example Implementation of the sdm Activation Function

We include an implementation of the sdm activation function using PyTorch (Paszke et al., 2019), version 2.3.0, in Listing 3.

Listing 3: Implementation of the sdm activation function in PyTorch, version 2.3.0.

```
def sdm_activation_function(batch_input, q, distance_quantile_per_class=None, log=False):
    """
    sdm activation function
    Parameters
    ----------
    batch_input
        torch.tensor
            shape == [batch size, number of classes]
    q
        torch.tensor
            shape == [batch size, 1], with each value in [0, max q]
    distance_quantile_per_class
        torch.tensor, or None
            If not None, shape == [batch size, number of classes], with each quantile in [0,1]. As a final layer
            activation function, with batch_input $\in \reals$, it is assumed that the quantiles are the same
            across classes, for a given instance. This ensures the argmax does not change relative to
            torch.argmax(batch_input, dim=1).
    log
        log with change of base, for training
    Notes:
        For context, with e.g. batch size = 1, the standard softmax is obtained by using q=torch.tensor([[torch.e-2]])
        and (distance_quantile_per_class=None or distance_quantile_per_class=torch.ones(1, number of classes) ).
    Returns
    -------
    [batch size, number of classes]
    """
    assert len(batch_input.shape) == 2
    assert batch_input.shape[0] == q.shape[0]
    assert q.shape[1] == 1
    if distance_quantile_per_class is not None:
        assert batch_input.shape == distance_quantile_per_class.shape
    q_rescale_offset = 2
    q_factor = q_rescale_offset + q
    batch_input = batch_input - torch.amax(batch_input, dim=1, keepdim=True) # for numerical stability
    if distance_quantile_per_class is not None:
        rescaled_distribution = q_factor ** (batch_input * distance_quantile_per_class)
    else:
        rescaled_distribution = q_factor ** batch_input
    if log: # log_base{q}
        kEPS = torch.finfo(torch.float32).eps # adjust as applicable for platform
        rescaled_distribution = torch.log(rescaled_distribution + kEPS) - torch.log(
            torch.sum(rescaled_distribution, dim=1) + kEPS).unsqueeze(1)
        return rescaled_distribution / torch.log(q_factor)
    else:
        return rescaled_distribution / torch.sum(rescaled_distribution, dim=1).unsqueeze(1)
```

A.5 Empirical CDF Function

Listing 4: An implementation of the empirical CDF conventions used in this work, using NumPy, version 1.26.4. See the text for a further discussion.

```python
def getCDFIndex(trueClass_To_CDF, val, prediction, reverse=False, val_in_0to1=False):
    # trueClass_To_CDF is a dictionary with a key for each class, the values of which are sorted ascending lists of numbers, since
         np.searchsorted assumes an ascending sort of its initial argument.
    if prediction not in trueClass_To_CDF or len(trueClass_To_CDF[prediction]) == 0:
        return 0.0
    if val_in_0to1 and len(trueClass_To_CDF[prediction]) > 0 and val >= trueClass_To_CDF[prediction][-1]: # saturation guard
        assert not reverse
        return 1.0
    index = np.searchsorted(trueClass_To_CDF[prediction], val, side="left") # will be 0 for len() == 0
    if reverse: # use for distances
        return 1 - index / len(trueClass_To_CDF[prediction])
    else:
        return index / len(trueClass_To_CDF[prediction])
```

The conventions for implementing the empirical CDF functions follow in the expected ways, but we briefly highlight the key considerations below, as they can impact the behavior of the estimators. An implementation in NumPy (Harris et al., 2020), version 1.26.4, appears in Listing 4.

1. The distance quantiles should be exclusionary at the boundaries. When $d_{\text{nearest}} = 0$, the $1 - \text{eCDF}^{\cdot}_{\text{ca}}(d_{\text{nearest}})$ quantile should be 1, and when d_{nearest} is greater than the maximum observed distance (across $\mathcal{D}_{\text{ca}}$ for $\boldsymbol{x} \in \mathcal{D}_{\text{te}}$ and $\boldsymbol{x} \in \mathcal{D}_{\text{ca}}$, and across $\mathcal{D}_{\text{tr}}$ for $\boldsymbol{x} \in \mathcal{D}_{\text{tr}}$, the latter case only occurring during training), the $1 - \text{eCDF}^{\cdot}_{\text{ca}}(d_{\text{nearest}})$ quantile should be 0.

2. For the quantiles over an SDM activation, as needed for calibration, saturated values at the high-end should be assigned a quantile of 1. In the example code, this is achieved by setting the argument `val_in_0to1=True`.

A.6 Example Implementation of the Negative+Positive Vocabulary Normalization and L^2 Regularization Term

The positive+negative vocabulary normalization and regularization loss (Eq. 32) are conceptually parsimonious and straightforward to implement. Code scaffolding for an example implementation of an SDM network training loop appears in Listing 5. For computational expediency, here (as in the experiments in the main text), the q values and distance quantiles are calculated after each epoch, although in principle, they can be calculated with updated network values as an epoch progresses.

Listing 5: Code scaffolding in PyTorch, version 2.3.0, for a basic training loop of an SDM network with the Negative+Positive Vocabulary Normalization and L^2 regularization term, where the q values and distance quantiles are updated after each epoch.

```
pdist = nn.PairwiseDistance(p=2)
criterion = nn.NLLLoss()
for e in range(total_epochs):
    total_mini_batches = len(range(0, train_size, batch_size))
    beta = min_beta
    beta_step = (max_beta-min_beta) / total_mini_batches
    for i in range(0, train_size, batch_size):
        optimizer.zero_grad()
        model.train()
        batch_genai_y = # the next-token labels with applicable index+|V| offsets
        # the sdm activations for the negative+positive joint distribution and the concatenation of the reference
        # distribution with itself use the same q and distance quantiles for the corresponding instances:
        batch_f_genai = # log_base{q} sdm activation(negative+positive linear layers output), where + is pseudo-code for concatenation
        batch_f_original = # log_base{q} sdm activation(reference distribution+reference distribution linear layers output)
        with torch.no_grad():
            top_events_k = 1
            top_k_sort_by_largest = True
            # "negative" refers to indexes in the first half of the concatenated distributions, [0, |V|); "positive" to the second half
                  [|V|, |V|*2):
            neg_original_max_half_distribution_i = torch.topk(batch_f_original[:, 0:model.gen_ai_vocab],
                                                         top_events_k, dim=1, largest=top_k_sort_by_largest)[1]
            pos_original_max_half_distribution_i = torch.topk(batch_f_original[:, -model.gen_ai_vocab:],
                                                         top_events_k, dim=1, largest=top_k_sort_by_largest)[1] + model.gen_ai_vocab #
                                                               note the offset
            negative_max_half_distribution_i = torch.topk(batch_f_genai[:, 0:model.gen_ai_vocab],
                                                     top_events_k, dim=1, largest=top_k_sort_by_largest)[1]
            positive_max_half_distribution_i = torch.topk(batch_f_genai[:, -model.gen_ai_vocab:],
                                                     top_events_k, dim=1, largest=top_k_sort_by_largest)[1] + model.gen_ai_vocab # note
                                                           the offset
            distribution_mass_mask = (
                    torch.ones_like(batch_f_genai).scatter_(1, neg_original_max_half_distribution_i, 0.0) *
                    torch.ones_like(batch_f_genai).scatter_(1, pos_original_max_half_distribution_i, 0.0) *
                    torch.ones_like(batch_f_genai).scatter_(1, negative_max_half_distribution_i, 0.0) *
                    torch.ones_like(batch_f_genai).scatter_(1, positive_max_half_distribution_i, 0.0) *
                    torch.ones_like(batch_f_genai).scatter_(1, batch_genai_y.unsqueeze(1), 0.0)
            ).to(batch_f_genai.device)
        regularization_term = pdist(
            distribution_mass_mask * batch_f_original,
            distribution_mass_mask * batch_f_genai).mean()
        llm_loss = criterion(batch_f_genai, batch_genai_y)
        with torch.no_grad(): # rescaling factor for the regularization term
            regularization_scale_term = (torch.log(llm_loss + model.kEPS) /
                                      (torch.log(regularization_term + model.kEPS) + model.kEPS)
                                      ).item()
        loss = llm_loss + beta * torch.sqrt(
            torch.clamp(regularization_term, min=1.0) ** min(max(regularization_scale_term, 0.0), 1.0))
        loss.backward()
        optimizer.step()
        beta += beta_step
    # Before the next epoch, for each training instance, update q and distance quantiles using the sdm activation layer trained for
          verification.
```

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL `https://arxiv.org/abs/2404.14219`.

Anastasios Nikolas Angelopoulos, Stephen Bates, Michael Jordan, and Jitendra Malik. Uncertainty Sets for Image Classifiers using Conformal Prediction. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=eNdiU_DbM9`.

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. pages 967–976, Singapore, December 2023. doi: 10.18653/v1/2023.findings-emnlp.68. URL `2023.findings-emnlp.68`.

Glenn W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1 – 3, 1950. doi: 10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2. URL `https://journals.ametsoc.org/view/journals/mwre/78/1/1520-0493_1950_078_0001_vofeit_2_0_co_2.xml`.

C. K. Chow. An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, EC-6(4):247–254, 1957. doi: 10.1109/TEC.1957.5222035.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane

Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi: 10.1109/TIT.1967.1053964.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 933–941. JMLR.org, 2017.

A. P. Dawid. The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610, 1982. doi: 10.1080/01621459.1982.10477856. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1982.10477856`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423/`.

Luc Devroye, László Györfi, and Gábor Lugosi. A Probabilistic Theory of Pattern Recognition. In *Stochastic Modelling and Applied Probability*, 1996.

A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic Minimax Character of the Sample Distribution Function and of the Classical Multinomial Estimator. *The Annals of Mathematical Statistics*, 27(3):642 – 669, 1956. doi: 10.1214/aoms/1177728174. URL `https://doi.org/10.1214/aoms/1177728174`.

Rina Foygel Barber, Emmanuel J Candès, Aaditya Ramdas, and Ryan J Tibshirani. The limits of distribution-free conditional predictive inference. *Information and Inference: A Journal of the IMA*, 10(2):455–482, 08 2020. ISSN 2049-8772. doi: 10.1093/imaiai/iaaa017. URL `https://doi.org/10.1093/imaiai/iaaa017`.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/gal16.html`.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/4a8423d5e91fda00bb7e46540e2b0cf1-Paper.pdf`.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1321–1330. JMLR.org, 2017.

Chirag Gupta and Aaditya Ramdas. Top-label calibration and multiclass-to-binary reductions. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=WqoBaaPHS-.

Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on apple silicon, 2023. URL https://github.com/ml-explore.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825): 357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

J. T. Gene Hwang and A. Adam Ding. Prediction intervals for artificial neural networks. *Journal of the American Statistical Association*, 92(438):748–757, 1997. ISSN 01621459. URL http://www.jstor.org/stable/2965723.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Meelis Kull, Miquel Perello-Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. *Beyond Temperature Scaling: Obtaining Well-Calibrated Multiclass Probabilities with Dirichlet Calibration.* Curran Associates Inc., Red Hook, NY, USA, 2019.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf`.

Jing Lei and Larry Wasserman. Distribution-free prediction bands for non-parametric regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1): 71–96, 2014. doi: https://doi.org/10.1111/rssb.12021. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssb.12021`.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

P. Massart. The Tight Constant in the Dvoretzky-Kiefer-Wolfowitz Inequality. *The Annals of Probability*, 18(3):1269 – 1283, 1990. doi: 10.1214/aop/1176990746. URL `https://doi.org/10.1214/aop/1176990746`.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. Crosslingual generalization through multitask finetuning. pages 15991–16111, Toronto, Canada, July 2023. doi: 10.18653/v1/2023.acl-long.891. URL `2023.acl-long.891`.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel

Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino

Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/8558cb408c1d76621371888657d2eb1d-Paper.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

John C. Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

Yaniv Romano, Matteo Sesia, and Emmanuel J. Candès. Classification with valid and adaptive coverage. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Sara Rosenthal, Noura Farra, and Preslav Nakov. SemEval-2017 task 4: Sentiment analysis in Twitter. In Steven Bethard, Marine Carpuat, Marianna Apidianaki, Saif M. Mohammad, Daniel Cer, and David Jurgens, editors, *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2088. URL https://aclanthology.org/S17-2088/.

Allen Schmaltz. Detecting local insights from global labels: Supervised and zero-shot sequence labeling via a convolutional decomposition. *Computational Linguistics*, 47(4):729–773, December 2021. doi: 10.1162/coli_a_00416. URL https://aclanthology.org/2021.cl-4.25.

Allen Schmaltz, Yoon Kim, Alexander M. Rush, and Stuart Shieber. Sentence-level grammatical error identification as sequence-to-sequence correction. In Joel Tetreault, Jill Burstein, Claudia Leacock, and Helen Yannakoudakis, editors, *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 242–251, San Diego, CA, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-0528. URL `https://aclanthology.org/W16-0528/`.

Allen Schmaltz, Yoon Kim, Alexander Rush, and Stuart Shieber. Adapting sequence models for sentence correction. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2807–2813, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1298. URL `https://aclanthology.org/D17-1298/`.

Kim Sharp and Franz M. Matschinsky. Translation of ludwig boltzmann's paper "on the relationship between the second fundamental theorem of the mechanical theory of heat and probability calculations regarding the conditions for thermal equilibrium" sitzungberichte der kaiserlichen akademie der wissenschaften. mathematisch-naturwissen c. *Entropy*, 17: 1971–2009, 2015. URL `https://api.semanticscholar.org/CorpusID:17745806`.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=B1ckMDqlg`.

Juozas Vaicenavicius, David Widmann, Carl R. Andersson, Fredrik Lindsten, Jacob Roll, and Thomas Bo Schön. Evaluating model calibration in classification. In *International Conference on Artificial Intelligence and Statistics*, 2019. URL `https://api.semanticscholar.org/CorpusID:67749814`.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, nov 1984. ISSN 0001-0782. doi: 10.1145/1968.1972. URL `https://doi.org/10.1145/1968.1972`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 0387001522.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL `https://openreview.net/forum?id=y10DM6R2r3`.