MICE for CATs: Model-Internal Confidence Estimation for Calibrating Agents with Tools

Nishant Subramani[♠*] **Jason Eisner**[◇] **Justin Svegliato**[◇]
Benjamin Van Durme[◇] **Yu Su**[◇†] **Sam Thomson**[◇†]

[♠]CMU LTI [◇]Microsoft
[♠]nishant2@cs.cmu.edu
[◇]{jason.eisner,jsvegliato,ben.vandurme,
yusu2,samuel.thomson}@microsoft.com

Abstract

Tool-using agents that act in the world need to be both useful and safe. Well-calibrated model confidences can be used to weigh the risk versus reward of potential actions, but prior work shows that many models are poorly calibrated. Inspired by interpretability literature exploring the internals of models, we propose a novel class of **model-internal confidence estimators (MICE)** to better assess confidence when calling tools. MICE first decodes from each intermediate layer of the language model using *logit lens* (nostalgebraist, 2020) and then computes similarity scores between each layer's generation and the final output. These features are fed into a learned probabilistic classifier to assess confidence in the decoded output. On the simulated trial and error (STE) tool-calling dataset using Llama3 models, we find that MICE beats or matches the baselines on smoothed expected calibration error. Using MICE confidences to determine whether to call a tool significantly improves over strong baselines on a new metric, **expected tool-calling utility**. Further experiments show that MICE is sample-efficient, can generalize zero-shot to unseen APIs, and results in higher tool-calling utility in scenarios with varying risk levels. Our code is open source, available at https://github.com/microsoft/mice_for_cats.

1 Introduction

Language models are increasingly being used as tool-using agents, where they can generate executable API calls that can change external environments (Schick et al., 2024; Yan et al., 2024; Wang et al., 2024; Roy et al., 2024). Sometimes the generated tool calls are relatively safe, and mistakes will have minimal impact (e.g., if *"how many grand slams has Serena Williams won?"* resulted in the incorrect tool call `tennis_reference_count_grand_slams(name="venus williams")`, then the user would just be misinformed). But other times, incorrect tool calls can be more harmful (e.g., if *"please remove slash.txt"* resulted in the incorrect tool call `cli(args="rm -rf /")`, then the user would lose the contents of their filesystem).

A *confidence estimator* estimates the probability that another model's output is correct. A simple confidence estimator for a language model would be based on the probability that the model itself assigns to its output (i.e., the product of token probabilities) or to its output's semantic equivalence class (Zhong et al., 2023; Farquhar et al., 2024). Yet prior work has shown that this method can be *poorly calibrated* (Jiang et al., 2021; Mielke et al., 2022; Kadavath et al., 2022; Yin et al., 2023). A probabilistic classifier is *well calibrated* if on an unseen test distribution, it is correct about as often as it thinks it is (Dawid, 1982; Guo et al., 2017; Desai and Durrett, 2020; Zhao et al., 2021; Hashemi et al., 2024). For example, of those unseen examples that it predicts to be positive with $\approx 25\%$ probability, $\approx 25\%$ really are positive. Well-calibrated probabilities can be used to guide downstream decisions, but calibration should never be one's only engineering target, as even a highly unsure classifier may be well-calibrated (see §3.1).

To that end, we introduce a class of **model-internal confidence estimators (MICE)** and an end-to-end metric, **expected tool-calling utility (ETCU)**, to evaluate a tool-calling agent that consults a confidence estimator to decide when to launch the predicted tool call.[1] MICE extracts fea-

[*] Work performed during an internship at Microsoft.
[†] Equal mentors.

[1] We train and test confidence estimators specifically on the generation of *tool calls*—a new setting for confidence estimation. However, MICE could equally well be applied to well-studied confidence estimation settings in NLP, such as machine translation (Blatz et al., 2004; Kumar and Sarawagi, 2019; Wang et al., 2020), long-form generation (Band et al., 2024), and semantic parsing (Stengel-Eskin and Van Durme, 2023a).

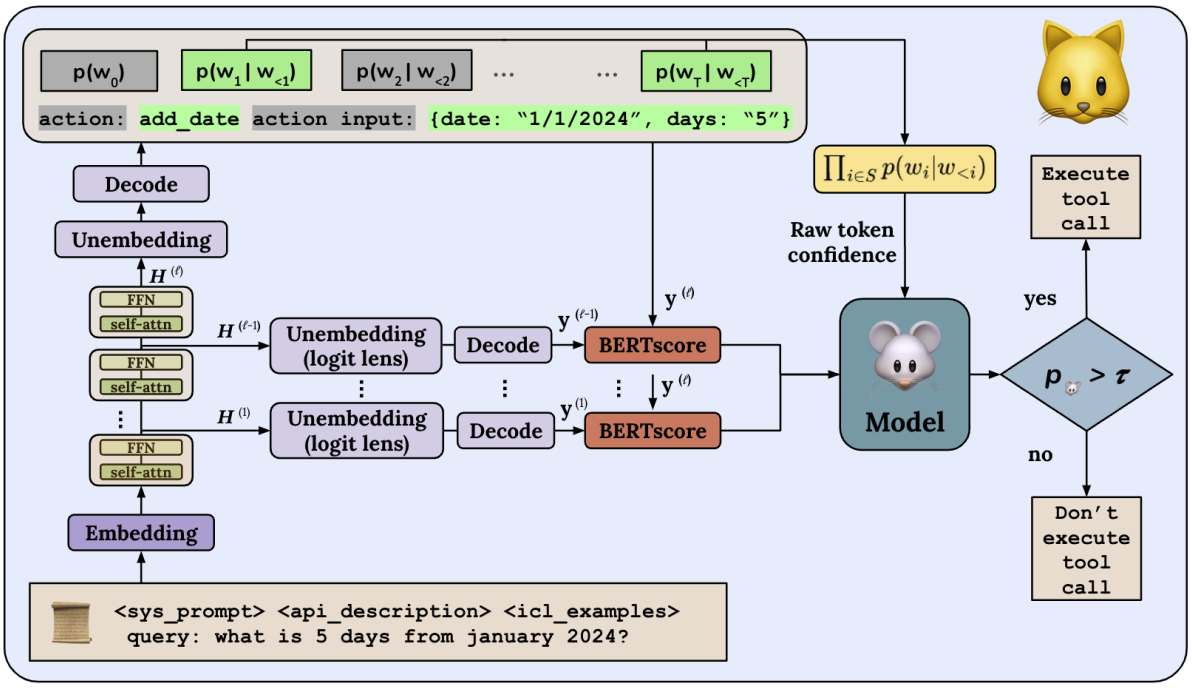

Figure 1: The MICE architecture.

tures by decoding from the intermediate layers of a transformer-based large language model (LLM) and computes the similarities of those generations to the output of the final layer. Based on these features and the LLM's raw confidence, it learns a model that outputs a confidence score. MICE excels on ETCU, increasingly outperforming two strong baselines as the cost of incorrect tool calls increases, without increasing calibration error.

This paper makes the following contributions: We propose a class of model-internal confidence estimators (MICE) that are empirically well-calibrated on the task of assessing generated tool calls (§2). We introduce a new metric, expected tool-calling utility, that combines accuracy and calibration to better evaluate tool-calling agents (§3). Finally, we show that MICE is sample-efficient and can generalize to new tools, even in a zero-shot setting (§5).

2 Model-Internal Confidence Estimators

MICE is a simple learned probabilistic classifier whose features are derived from model-internal activations. Prior work on understanding the internals of transformer language models has shown that intermediate layers at different depths encode different types of information, and that the activation spaces at various layers of these models can be nudged to generate sequences in targeted ways (Tenney et al., 2019; Subramani et al., 2019; Subramani and Suresh, 2020; Subramani et al., 2022; Turner et al., 2023). Decoding from the layers of a transformer language model has provided insight into the underlying mechanisms and has been used in early-exit algorithms for faster generation (nostalgebraist, 2020; Geva et al., 2022; Schuster et al., 2022; Belrose et al., 2023). For question answering tasks, decoding from roughly the first half of the layers of the language model produces unintelligible results, but in later layers the model's predictions slowly refine into a plausible answer (Merullo et al., 2024).

We hypothesize that features from intermediate layers' hidden states could provide useful signal for calibration. Drastic changes in the final few layers could indicate the inability for the LLM to pinpoint a tool call. As a result, we may trust a prediction that was slowly refined into an answer over the final 50% of layers more than one that drastically changed in the final few layers, even if they had the same distribution at the end.

BERTScore Features Since we hypothesize that intermediate layers' hidden states could be a useful signal for calibration beyond the confidences derived from the final layer, we decode from each layer, much as in *logit lens* (nostalgebraist, 2020), a model interpretation technique. We first decode the output string $\mathbf{y}$ at temperature 0. This is the

Layer number	Query: When is the date exactly one year after October 15, 2022?'	Query: What is the chance of rain in Sydney for the next 5 days?
Layer 5	.Accessible첵'gcinitializerライン 'gc'gcoldurCalibri oldsวรรณ'gc /Area'gc#ad.Accessible	šker Masc.userInteractionEnabled/***/'gc'gc Böl첵'gcกลางแจ้งعدد ...
Layer 15	argout.labelX/Area卓 \nزيزUnitTestCrLfUnitTest...	emale'gcскладділ#aaVISIBLE监听页面 ippetystackystackutowystackBeNull.tieo лю/...
Layer 25	in: when is the date exactly..\n after October..\n 202202Action...?\n -fontawesome input:\Migration..\n "202 ;\n -October15...\n ...	ready ready to help with/stdc\Migration: What is the chance of rain in..\n for next...\n five...?\n weather..\n _exempt input: {\n " Sydney...\n Australia..\n weather..\n/stdc..\n ...
Layer 31	When is the date..\n one.. \n after October 15, 2022? \n Action: add date \n Action in: {"date": "2022-10-15", "..\n ": in}\n ation: ..\n , 2023	Action:...\n weather\n Action in: {\n "location": "..\n dney", \n " days": 5}\n ation: The..\n ed Sydney over the next 5 days is a to of rain, with an.. of 30- chance of precipitation per day.Final..\n : The..\n of in in..\n for the next 5
Layer 32	Action: add_date Action Input: {"date": "2022-10-15", "days": 365}	Action: forecast_weather\n Action Input: {\n "location": "Sydney",\n "days": 5\n }

Figure 2: Example generations from the validation set across layers of the Llama3-8B-Instruct model. Generations from early layers (5, 15) are seemingly random, but later layers (25, 31) generate thematically relevant tokens. Layer 32 is the final layer.

usual way to obtain model output in a task like tool calling. Then at each layer $i < \ell$, we obtain a *preliminary output string* $\mathbf{y}^{(i)}$ of the same length by per-token argmax decoding:[2]

$$\mathbf{y}_t^{(i)} = \operatorname{argmax} \mathbf{h}_{t-1}^{(i)} W_{\text{out}} \tag{1}$$

where each t is a token position in $\mathbf{y}$, and the row vector $\mathbf{h}_{t-1}^{(i)} \in \mathbb{R}^d$ is the model's layer-i encoding at the previous position, whose product with the unembedding matrix $W_{\text{out}} \in \mathbb{R}^{d \times |V|}$ is a vector of logits $\in \mathbb{R}^{|V|}$. Here, d is the embedding size and $|V|$ is the vocabulary size. This results in ℓ strings, where ℓ is the number of layers of the model.

We then compute the BERTScore (Zhang et al., 2020) between $\mathbf{y}$ and each $\mathbf{y}^{(i)}$. These become the main input features to the MICE model.[3]

Raw Confidence Feature We also integrate the raw confidence of the language model in generating the tool call as a feature to the MICE model. We calculate this by computing the product of the probabilities of the tokens in the generated tool call. We notice that including formatting tokens, which are always present in the tool call, leads to increased noise and a less accurate estimate of confidence, so we omit the tokens associated with formatting. The gray tokens in Figure 1 were omitted, while the green ones were included.

[2]Note that taking $i = \ell$ in (1) would recover $\mathbf{y}$. Because the transformer uses residual connections, each layerwise encoding $\mathbf{h}_{t-1}^{(i)}$ has the same dimensionality d, so multiplication by the unembedding matrix $W_{\text{out}}$is defined even when $i < \ell$. All of these vector-matrix products can be computed in parallel by a matrix-matrix product, $H^{(i)} W_{\text{out}}$ where $H^{(i)} \in \mathbb{R}^{\ell \times d}$.

[3]BERTScore reencodes the strings $\mathbf{y}$ and $\mathbf{y}^{(i)}$ with a separate model (see §4.4) and aligns their tokens. We found that the alignments were not always trivial. BERTscore performed significantly better than methods we explored initially, which compared the $\operatorname{softmax}(\mathbf{h}_{t-1}^{(i)} W_{\text{out}})$ distributions rather than argmax-decoding single strings $\mathbf{y}^{(i)}$. See §9 for other options.

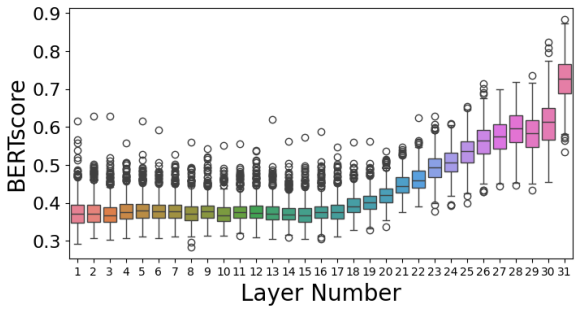

Figure 3: BERTScore similarities between the generated string $\mathbf{y}$ and the preliminary strings $\mathbf{y}^{(i)}$ from earlier layers, for Llama3-8B-Instruct on the STE validation set (Wang et al., 2024). See also Figure 8 in Appendix A.

Model Architecture We train a simple supervised classifier that predicts whether the generated tool call $\mathbf{y}$ is correct. It maps from the input features—the BERTScores and the raw confidence—to a probability of correctness (i.e., a confidence estimate). Any trainable model of this form could be used here; the specific architectures and baselines that we tried will be described in §4.4.

3 Metrics

Perhaps the most widely reported calibration metric is **expected calibration error** (§3.1). As mentioned in the introduction, however, minimizing ECE should not be our only goal. We also introduce a utility metric, **expected tool-calling utility** (§3.2), to assess the performance of a simple agent that makes call/no-call decisions by using our well-calibrated confidence estimates. This metric is parameterized by the cost of false positives relative to the reward of true positives.

3.1 Expected Calibration Error (ECE)

Expected calibration error (**ECE**; Naeini et al., 2014, 2015) is computed by constructing a histogram binned by predicted confidence, $\hat{p}$. The accuracy of examples within a given bin is compared to the mean predicted confidence within that bin, $|\text{acc} - \bar{\hat{p}}|$. These absolute differences are then averaged across bins, with each bin weighted by the fraction of examples in that bin.

We use a recently improved variant of ECE, **smooth ECE** (**smECE**; Błasiok and Nakkiran, 2024), which replaces histogram binning with Nadaraya-Watson kernel regression (Nadaraya, 1964; Watson, 1964). A reflected Gaussian kernel is used; the kernel width is determined automatically from the data, yielding a consistent estimator.

However, ECE and smECE do not distinguish between an *oracle* classifier that returns $\hat{p} = 1.0$ on correct outputs and $\hat{p} = 0.0$ on incorrect outputs, and a *maximally uninformative* probabilistic classifier that always predicts the base accuracy rate. That is, if 70% of all predictions are correct, then a trivial system that gave $\hat{p} = 0.7$ on every example would be perfectly calibrated (ECE $= 0$), yet mostly useless! We would prefer a system that tends to return high $\hat{p}$ on correct tool calls and low $\hat{p}$ on incorrect ones, so that we can execute the former and avoid executing the latter.

3.2 Expected Tool-Calling Utility (ETCU)

We thus introduce a parameterized metric, *expected tool-calling utility*, which approximates actual utility in situations where a *calibrated* confidence score $\hat{p}$ is used to decide of whether or not to execute a specific tool call generated by the language model. We assume we know the expected utility for each of the four possible outcomes: **tp** > 0 (true positive), for when the agent executes a *correctly* predicted call; **fp** < 0 (false positive), for when the agent executes an *incorrectly* predicted call; **tn** ≈ 0 (true negative), for when the agent avoids executing an *incorrect* call; **fn** ≈ 0 (false negative), for when the agent fails to execute a *correct* call. tn and fn may be slightly negative to account for time wasted making the unused prediction. fp may be highly negative, e.g., if the agent erroneously deletes all of the user's documents, makes a large unintended purchase, or sends an offensive email.

The exact values of tp, fp, tn, and fn will depend on the specific task that the agent must perform, and could be assigned by a domain expert or learned from data, like human preferences (Christiano et al., 2017).[4] Once these values have been assigned, the minimum Bayes risk (**MBR**; Bickel and Doksum, 1977, p. 27) decision can be calculated; it is to execute the predicted tool call if and only if the estimated confidence $\hat{p}$ is above the threshold

$$\hat{p} > \tau \stackrel{\text{def}}{=} \frac{\text{tn} - \text{fp}}{(\text{tp} - \text{fn}) + (\text{tn} - \text{fp})} \tag{2}$$

Calibration ensures that of all predicted tool calls with confidence $\approx \hat{p}$, about $\hat{p}$ are correct. The decision rule (2) makes either *all* such calls or *none* of them, according to whether the expected utility per call is higher with *all* ($\hat{p}\,\text{tp} + (1-\hat{p})\,\text{fp}$) or with *none* ($\hat{p}\,\text{fn} + (1-\hat{p})\,\text{tn}$). The threshold τ is high (> 0.5) if avoiding bad calls (benefit tn $-$ fp) is more important than executing good calls (benefit tp $-$ fn).

Normalizing: These four values can be scaled by any positive constant, and translated by any real constant, without affecting the optimal threshold or the utility (modulo that affine transform) (Gleave et al., 2021). That is, we can choose a measurement scale for our utilities (without loss of generality) such that tp $= 1$ and fn $= 0$. Two degrees of freedom still remain (tn and fp). In most tool-using scenarios, tn will be extremely close to fn, because in both cases the immediate action by the agent is the same (do not execute) and thus has the same effect regardless of the predicted action.[5] If we further assume (*with* loss of generality) that tn $=$ fn $= 0$, the intuitive interpretation is that the agent gets 1 "credit" (an arbitrary utility unit) for completing its task, 0 credits for doing nothing (regardless of whether that was the best decision), and fp < 0 credits for doing something wrong. The single remaining degree of freedom fp *is* the risk/utility ratio, defining how costly it is to attempt and fail. In this (slightly less general) case, the MBR decision rule (2) simplifies to:

$$\hat{p} > \tau \stackrel{\text{def}}{=} \frac{-\text{fp}}{1 + -\text{fp}} = \frac{\text{fp}}{\text{fp} - 1} \tag{3}$$

Settings for expected tool-calling utility: To understand how confidence estimators perform at different risk levels, we choose three different values of fp under which to measure normalized risk (Table 1). Each setting of fp determines a threshold τ that the Bayes-optimal policy will use.

High Risk: Tasks where executing an incorrect tool call is much worse than the reverse error. We choose fp $= -9$ for this setting, giving $\tau = 0.9$.

Medium Risk: For these tasks, executing an incorrect tool call is as bad as executing the correct tool call is good (fp $= -$tp $= -1$), giving $\tau = 0.5$.

[4] They will also often depend on the predicted API and arguments. A more careful MBR practitioner would ideally condition on these and assign utilities to each possible pair (gold specific action, chosen specific action). Our coarser expectations $\{\text{tp}, \text{fp}, \text{tn}, \text{fn}\}$ result in cruder decisions.

[5] tn might differ from fn because *subsequent* actions may diverge, and each utility should ideally include the expected future reward over all possible rollouts. For example, it might be slightly easier to ask clarifying questions when the original prediction was correct (implying fn $>$ tn and raising τ).

Model	Confidence Estimator	smECE (↓)	Tool-Calling Utility (↑)			
			Low risk	Medium risk	High risk	AUC
Llama3-8B-Instruct	Raw Confidence	0.184	0.351	0.015	-0.323	0.001
	HRE	0.041	**0.356**	0.033	0.000	0.110
	NWKR	0.039	**0.356**	0.027	0.000	0.118
	MICE LR Zeroshot	**0.036**	**0.356**	0.025	0.000	0.114
	MICE RF Zeroshot	0.065	0.349	0.011	-0.016	0.096
	MICE LR	0.037	**0.356**	0.055	0.000	0.125*
	MICE RF	0.040	**0.356**	**0.100**†	**0.024***†	**0.144***†
Llama3.1-8B-Instruct	Raw Confidence	0.229	0.240	-0.129	-0.579	-0.127
	HRE	0.050	0.229	0.017	0.000	0.060
	NWKR	**0.032**	0.239	0.000	0.000	0.063
	MICE LR Zeroshot	0.054	0.239	0.019	0.000	0.061
	MICE RF Zeroshot	0.050	0.233	0.051†	-0.008	0.073
	MICE LR	0.050	0.239	0.051*†	0.000	0.071*†
	MICE RF	**0.032**	**0.248**	**0.072***†	**0.023***†	**0.104***†
Llama3.2-3B-Instruct	Raw Confidence	0.312	0.231	-0.209	-1.528	-0.458
	HRE	0.036	0.229	0.000	0.000	0.057
	NWKR	0.027	0.233	0.000	0.000	0.057
	MICE LR Zeroshot	0.032	0.230	0.008	0.000	0.064
	MICE RF Zeroshot	0.061	0.232	0.021	-0.068	0.049
	MICE LR	**0.025**	0.232	0.016	0.000	0.073*†
	MICE RF	0.041	**0.237**	**0.071***†	**0.013**	**0.095***†

Table 1: Results on the test set. Lower smECE is better, while higher tool-calling utility is better. **Bold** indicates the best result in each category and underline indicates the second best result in each category. * indicates statistical significance compared to HRE and † indicates significance compared to NWKR (p-value < 0.05, permutation test).

Low Risk: These are tasks where executing an incorrect tool call has relatively low potential downside. We choose $\text{fp} = -\frac{1}{9}$, giving $\tau = 0.1$.

Area Under Curve (AUC): More generally, we can compute an expected value for *any* $\tau \in (0, 1)$. This yields an "expected tool-calling utility" curve (Figure 4) for a given confidence estimator on a given dataset. Any given applied setting may only be interested in a single τ along the curve. Still, to compare estimators overall, it may be useful to consolidate the curve into a single number, summarizing an estimator's performance across all risk levels. Taking inspiration from the area under the receiver operating characteristic (ROC) curve (Marcum, 1960), we take the average of the expected tool-calling utility values at every point along the curve, which can be regarded as the (signed) **area under the curve** (**AUC**).[6] Since our formulation sets $\text{tn} = \text{fn} = 0$, always abstaining gets a expected tool-calling utility score of 0 regardless of risk level, and thus an AUC of 0. Because utilities can be negative, AUC values can also be negative. This occurs when model is overconfident (due to poor calibration) in too many high-risk predictions.

[6]In practice, we approximate AUC by evaluating expected tool-calling utility at each $\tau \in \{0.001, 0.002, \ldots, 0.999\}$.

4 Experiments

We now look at training MICE and using it at test time to measure both smooth expected calibration error (smECE) and expected tool-calling utility.

4.1 Dataset

Our experiments use the simulated trial-and-error (STE) dataset (Wang et al., 2024). The dataset was synthetically generated by simulating plausible tool-using scenarios for a given API and using GPT3.5-turbo with execution feedback to identify (presumptively) correct tool calls.

The dataset consists of English-language queries that require calling 50 distinct APIs. For tool call generation, we few-shot prompt an off-the-shelf LLM with examples from a *demonstration set* consisting of 4,520 examples taken from the STE training set. An alternative would have been to fine-tune the LLM on this demonstration set.

To train MICE, we use the rest of the STE training set, split into a *training set* of 1500 examples (30 from each API) and a *validation set* of 750 examples (15 from each API). We then evaluate MICE on STE's *test set* of 750 examples. In all cases, we label a generated tool call as correct if and only if it exactly matches the one given by STE.

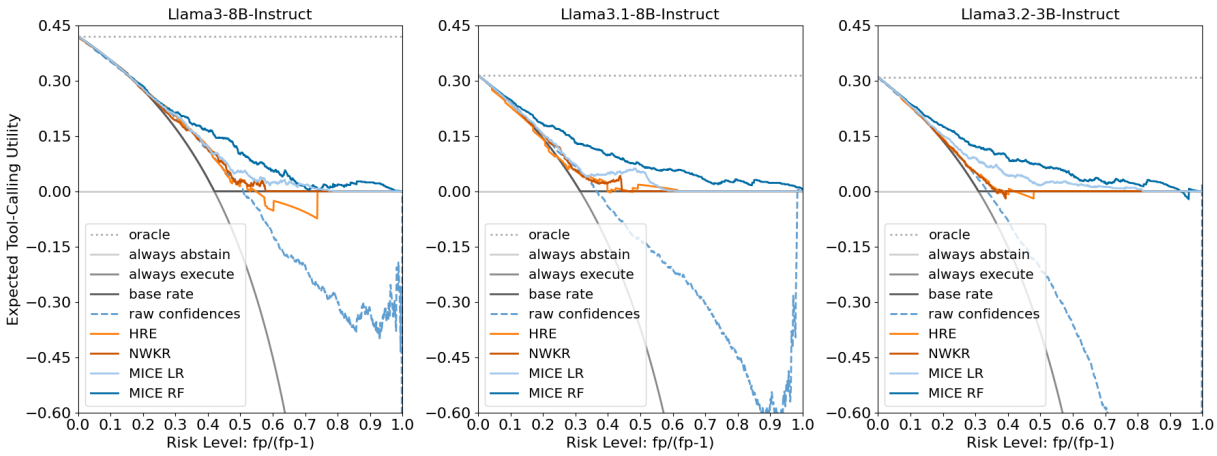

Figure 4: Expected tool-calling utility on the test set at varying risk levels. We include four trivial policies for reference: `oracle` executes only when the underlying model is correct (an upper bound); `always abstain` never executes, getting reward 0; `always execute` never abstains; and the `base rate` policy switches from `always execute` to `always abstain` when the risk level exceeds the base accuracy. All policies perform similarly at low risk levels, where `always execute` is close to optimal and hard to improve on. MICE models show clear improvements in the medium and high risk regimes.

4.2 LLMs

We consider three LLMs: Llama3-8B-Instruct, Llama3.1-8B-Instruct, and Llama3.2-3B-Instruct (Dubey et al., 2024). We build and evaluate a separate MICE classifier for each LLM.

4.3 Experimental Settings

We run each LLM on our validation and test sets in an 8-shot in-context learning setting, following Wang et al. (2024), using greedy decoding to generate the tool calls $\mathbf{y}$. For each evaluation example, the 8 in-context learning examples are selected from our demonstration set according to the procedure of Wang et al. (2024), which computes similarity to the evaluation example using SentenceBERT (Reimers and Gurevych, 2019).

We train a baseline or MICE regressor on the training set to predict whether tool calls are correct, and use the validation set for hyperparameter and model selection. Features used by MICE regressors were described in §2. We then evaluate the regressor on the test set, using the metrics of §3.

4.4 MICE Configurations & Baselines

Raw Confidence Our first baseline is the raw confidence score from §2, which can be used directly as a confidence estimate $\hat{p}$. Recall that we defined this as $\prod_{i \in S} p(w_i | w_{<i})$, where S is the subset of token indices that are relevant to the tool call. S omits the tokens associated with formatting (`"action:"` and `"action input:"`, which are generated for every tool call), and also omits tokens that are generated *after* the arguments of the tool call. We observed in initial experiments that including these irrelevant tokens resulted in worse calibration. We also observed that taking the minimum probability across generated tokens instead of the joint probability (as in Zhou et al. (2022); Stengel-Eskin and Van Durme (2023a)) resulted in little effective difference. Note that calculating raw confidence does not require any learning, so neither the training nor validation set is used. Raw confidence is also used as a base feature in the estimators described below.

Histogram Regression Estimator (HRE; Nobel, 1996) For our second (stronger) baseline, we use a standard method to calibrate the previous baseline. We use the training set to construct a histogram binned by raw confidence scores. We use 25 bins: $[0, 0.04), [0.04, 0.08), \ldots, [0.96, 1.0]$. To map from a raw confidence score c to a recalibrated estimate $\hat{p}$, we look up c's bin, and return the percentage of examples in that bin that are correct. Note that this is the same histogram construction used to calculate traditional ECE (except here constructed on the training set), and so should be expected to perform well on ECE metrics.

Kernel Regressor (NWKR) Here, rather than using a histogram with fixed bins to recalibrate, we use Nadaraya–Watson kernel regression (Nadaraya, 1964; Watson, 1964), following the exact procedure Błasiok and Nakkiran (2024) used to compute smECE. Analogously to above, since this follows

the exact same procedure as in smECE, we should expect it to perform well under that metric.[7]

MICE Models We extract features as described in §2, using DeBERTa-xlarge-mnli to compute the BERTScore features as it is the strongest BERTScore base model (He et al., 2021). This gives $\ell - 1$ BERTScore features along with the raw confidence feature. There are $\ell = 32$ layers for Llama3 and 3.1 and 28 layers for Llama3.2.

MICE Logistic Regressor (MICE LR): We train a logistic regression model with an L2 regularization strength of 2 to predict whether the tool call is correct or not.

MICE Random Forest (MICE RF): We train a random forest classifier using 1000 trees each with a maximum depth of 20 and a maximum of 10 features to use at each split, using the Scikit-Learn package (Pedregosa et al., 2011). Other hyperparameters are set to defaults. This model is also trained to predict whether the tool call is correct.[8]

5 Results

Smooth Expected Calibration Error Lower smECE is better. The first numeric column of Table 1 shows that all of the confidence estimators are well-calibrated—their smECE values are small and not significantly different—except for the raw confidences, which have smECEs 3–10x higher than the others. This is not surprising: HRE and NWKR are explicitly designed to calibrate the raw confidences, while logistic regression and random forest training are known to produce well-calibrated classifiers (Niculescu-Mizil and Caruana, 2005).

Expected Tool-Calling Utility Figure 4 shows the expected tool-calling utility curve for each confidence estimator and each model. We find that raw confidence performs dangerously poorly at and above moderate risk levels. HRE and NWKR both degrade quickly toward 0 as risk increases. The MICE models also degrade, but more slowly: matching performance of HRE and NWKR at the lower risk levels and outperforming at medium and higher risk levels. Across all three LLMs, MICE RF performs best at nearly every risk level.

Table 1 displays how well confidence estimators perform at three specific risk level settings (low, medium, and high) and across the full range of risk levels using AUC (see §3.2). For all of these metrics, a higher score is better. For each risk level, MICE RF always has the highest reward, outperforming HRE, NWKR, and MICE LR. Raw token confidence nearly always performs worst. For lower risk levels, most strategies perform comparably, with relatively high reward. This is expected: executing an incorrect tool call (fp) gives a low penalty relative to a correct tool (tp), so aggressively biasing for execution is optimal, garnering a high reward. As risk levels increase, the penalty for executing an incorrect tool call grows and using raw confidences nearly always incurs a negative reward when the risk level is greater than 0.5 (fp < -1). Across the three risk levels, we find that the MICE models outperform both baselines for each of the three tool-calling LLM agents.

We run permutation tests for each metric in Table 1 for each MICE method as compared to HRE and NWKR. In summary, MICE RF is always significant (p-value < 0.05) at the medium risk level, never significant at the low risk level, and significant at the high risk level for Llama3 and 3.1, but not 3.2. MICE LR outperforms the baselines, but is only significant for the medium risk level for Llama3.1. For the summary statistic AUC–ETCU, both MICE models are nearly always significantly better than HRE and NWKR for all three Llamas.

Zero-Shot Generalization to New APIs To test MICE's out-of-domain generalization, we simulate encountering new APIs by holding one out during training. Since there are 50 APIs present in the STE dataset, we train 50 MICE RF and 50 MICE LR models. Each model is trained on data from 49 APIs and evaluated solely on the held-out API. We combine the predictions from each of the models to get predictions across the entire test set.[9] These confidence estimates are solely constructed by MICE models that have never seen that specific API before, so every tool is unseen. MICE does worse in this setting, but only degrades to the level of HRE and NWKR models trained on the full data; they are statistically indistinguishable from them.

[7] HRE and NWKR learn to map a single confidence input feature to a recalibrated output confidence. Any confidence estimator can be calibrated in this way on held-out data. Other common approaches to this problem include isotonic regression and Platt scaling (§7).

[8] Note that HRE and MICE LR use a similar number of parameters, but in HRE they are devoted to closer analysis (binning) of the raw confidence dimension, rather than to additional BERTScore dimensions.

[9] This resembles 50-fold cross validation, where each fold is constructed solely with data from one API. However, for comparability with other methods, we evaluate on the corresponding fold in the test set, not the training set.

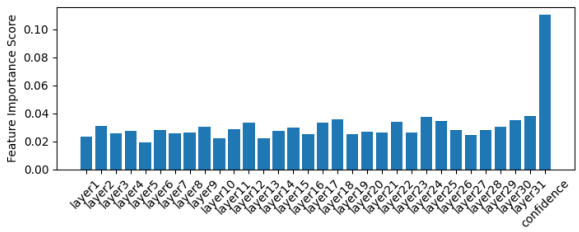

Figure 5: Feature importance for BERTScore features and confidence on the trained MICE RF model on the STE dataset for the Llama3 LLM.

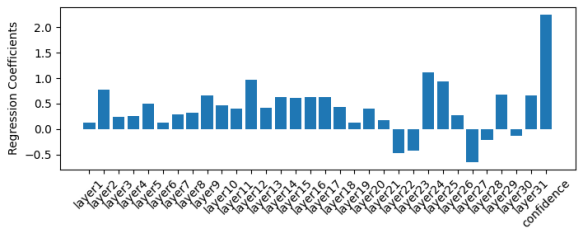

Figure 6: Coefficients for the trained MICE LR model on the STE dataset for the Llama3 LLM.

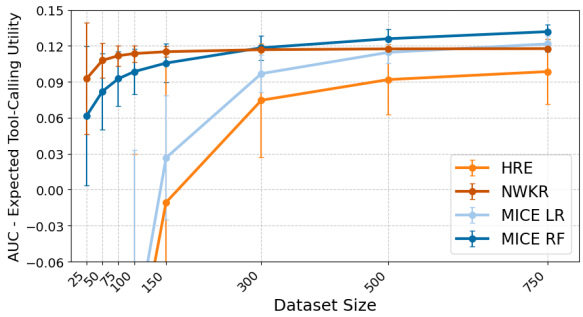

Figure 7: Sample complexity: AUC of MICE models and HRE baselines as the size of the training set varies on the Llama3-8B-Instruct model. Error bars are one standard deviation.

6 Analysis

What is generated by decoding from intermediate layers? Here, we look at what the LLM generates from intermediary layers. Using the logit lens, we find that models slowly evolve their predictions throughout the layers to get closer to the final output generation. Figure 2 shows sample generations. Qualitatively, the first two-thirds of the layers tend to generate seemingly random strings. After this point, the generations get increasingly closer to the final generation, but significant refinement still occurs in the final layer.

The box-and-whisker plot in Figure 3 shows that BERTScore tends to increase with layer number. Figure 8 in Appendix A shows that at some layers, the distribution of BERTScores tends to be shifted *slightly* higher on correct outputs, providing signal to the classifier.

What is learned? To better understand how the MICE features are used, we examine our MICE models trained on the STE dataset with Llama3-8B-Instruct. For MICE RF we calculate Gini coefficients, and for MICE LR we analyze the feature weights, as suggested by reviewers. Figures 5 and 6 indicate that confidence is the most important feature in both MICE models: roughly 3 times as important as other features in MICE RF and 2 times as important as other features in MICE LR.[10] There is no obvious other pattern in the estimated weights, and it is possible that they are underdetermined.

Feature Ablations To better understand which features contribute most to confidence estimation, we performed feature ablations for both MICE models for the three LLMs in our study. In addition to the original setting with all features, we tested four new settings: confidence only;[11] first half of the layers' BERTScores plus confidence; second half of the layers' BERTScores plus confidence; and all of the layers' BERTScores, but no confidence. See Table 2 in Appendix A for details.

MICE RF: Confidence alone performed extremely poorly. The second half of the layers plus confidence performed better than the first half plus confidence, but using all layers without confidence performed worse than using half the layers with confidence. This suggests that features from the second half of the model are more useful than the first half, and confidence is an important feature.

MICE LR: Confidence alone performed comparably to other settings, indicating that confidence accounts for much of the performance; unlike RF, LR learned how to use this feature. Additionally, for Llama3 and 3.1, using confidence alone outperformed using all the layers' BERTScore features. Using the second half of the layers' BERTScore features outperformed using the first half of the layers' features, similar to MICE RF.

How sample efficient is MICE? To measure sample efficiency, we vary the size of the training set to be 25, 50, 75, 100, 300, 500, and 750. For each size, we randomly partition the 1,500 training examples into disjoint groups of that size (e.g., 15

[10] Perhaps calibrated confidence would have worked even better as a feature.

[11] For LR, this is exactly Platt scaling with L2 regularization.

groups of 100 examples, or 3 groups of 500 examples). We then train on each group, measure AUC–ETCU, and compute the mean and variance across groups. We repeat this 100 times and average across trials, plotting results in Figure 7.

For dataset sizes of 150 or below NWKR performs best, but it saturates at this level and does not improve further with more data. MICE LR and HRE perform poorly with small datasets, but as size increases, they get closer to MICE RF and NWKR. For larger dataset sizes, MICE RF and MICE LR overtake NWKR. In fact, with as few as 300 examples (20% of the training data), MICE RF outperforms NWKR trained on the full dataset.

7 Related Work

Model Internals Tenney et al. (2019) show that different layers of models encode different aspects of the classical NLP pipeline. Moreover, intermediate layer activations can be nudged via steering vectors to control output generations (Subramani et al., 2019, 2022; Turner et al., 2023). The activation spaces of models are relatively well-formed and there exist directions in these latent spaces that correlate with interpretable properties (Subramani and Suresh, 2020; Li et al., 2024). These act as part of the basis for our hypothesis that the model internals could contain a trustworthiness signal, although we did not attempt to discover specific directions in these spaces.

Intermediate Decoding We can view language models with multiple layers as doing iterative inference, where each successive layer refines the predictions of the previous layer. With this lens, decoding from intermediate layers provides signal albeit noisy: the first half of the layers generate uninterpretable text, but after this predictions refine toward a plausible answer (Belrose et al., 2023; Yom Din et al., 2024; Merullo et al., 2024). Other work has focused on inference efficiency by early exiting from transformers (Teerapittayanon et al., 2017; Geva et al., 2022; Schuster et al., 2022; Elhoushi et al., 2024). Our work decodes from intermediary layers as a signal for better calibration.

Calibration Prior work has measured the calibration of off-the-shelf models, including neural networks (Niculescu-Mizil and Caruana, 2005; Wang, 2024), large language models (Kadavath et al., 2022; Yin et al., 2023), and semantic parsers (Stengel-Eskin and Van Durme, 2023a; Zhou et al., 2022).

A line of machine learning work focuses on calibrating binary classifiers while conditioning only on their predicted confidence. Platt scaling transforms a real-valued output (like that of an SVM classifier) into probabilities using logistic regression (Platt, 1999), which is proven to be equivalent to beta calibration up to preprocessing (Böken, 2021). Isotonic regression (Ayer et al., 1955) is a non-parametric approach that learns a best fit to data making only a monotonic non-decreasing assumption. HREs are popular, and there has been work on adaptive binning strategies (Nobel, 1996). We chose HRE and NWKR as strong baselines from this class of models. MICE LR could be viewed as an extension to Platt scaling because MICE conditions on model internals in addition to the original confidence.

Applications of Well-Calibrated Confidences The `DidYouMean` system can rephrase a query and ask for confirmation when the model is unconfident (Stengel-Eskin and Van Durme, 2023b). Like us, they frame the competing concerns in terms of safety and utility, weighing wrongly predicted actions against the cost of asking clarifying questions. While they tune a single confidence threshold, we transform confidences into calibrated probabilities so that a Bayes-optimal threshold can be dynamically derived for any risk/reward ratio. LACIE (Stengel-Eskin et al., 2024) communicates its fine-tuned confidences to users. APEL (Zhong et al., 2023) reduces its uncertainty about a semantic parse by asking questions of a user, using *raw* confidences to identify informative questions; calibrated confidences should work better, allowing it to finish with fewer questions.

8 Conclusion

In this work, we introduce model-internal confidence estimators (MICE), which improve the trustworthiness and safety of language models as tool-calling agents. We introduce a new metric, expected tool-calling utility, that combines calibration and usefulness to better evaluate the safety and utility of tool calls. We show that MICE matches or beats both regression baselines (HRE and NWKR) when measured by smooth ECE, and significantly improves expected tool-calling utility, especially in medium and high-risk regimes. Finally, we find that MICE is sample efficient and can generalize to unseen APIs in a zero-shot setting.

9 Limitations

Like logit lens, MICE assumes a transformer language model whose intermediate layers have the same shape as the final layer. More generally, MICE requires access to model internals, ruling out some of the most capable current LLMs, which are closed.

In principle, MICE is a general-purpose confidence estimation recipe for transformer language models. However, we evaluated MICE exclusively in one setting: a tool-calling task on one dataset. Other settings such as machine translation and question answering (see footnote 1) have been left to future experiments.

As footnote 3 hinted, there are many other possible ways to compute MICE features. We do not claim to have found the best variant even for the setting we studied. While we settled on BERTScore for this paper, there are several other possible choices for how to *encode*, *align*, *compare*, and *aggregate* the tokens at each layer. We remark that one possible encoding trick would be to learn a linear transform of each layer i so that $\mathbf{h}_{t-1}^{(i)}$ is maximally similar to $\mathbf{h}_{t-1}^{(\ell)}$ or maximally predictive of y_t, as in the *tuned lens* of (Belrose et al., 2023).

There are also various ways to build a classifier that uses MICE features. We also experimented with SVMs with different kernels (not reported). Other options could also be tried.

10 Impact Statement

Better calibrated models can help people make safer decisions. We hope to bring increased focus to risk/reward tradeoffs; we have intentionally framed the task and metric in a way that highlights the cost of false positives. Decision theory and reward functions are not a substitute for careful design, however; practitioners must exercise great care before hooking up an LLM to a tool with real effects in the world, including taking care to set appropriate rewards such as tp, fp, tn, fn.

References

Miriam C. Ayer, Hugh D. Brunk, George M. Ewing, W. T. Reid, and Edward Silverman. 1955. An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, 26:641–647.

Neil Band, Xuechen Li, Tengyu Ma, and Tatsunori Hashimoto. 2024. Linguistic calibration of long-form generations. In *Forty-first International Conference on Machine Learning*.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor V. Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. 2023. Eliciting latent predictions from transformers with the tuned lens. *ArXiv*, abs/2303.08112.

Peter Bickel and Kjell Doksum. 1977. *Mathematical Statistics: Basic Ideas and Selected Topics.*, volume 56. Holden-Day Inc.

Jarosław Błasiok and Preetum Nakkiran. 2024. Smooth ECE: Principled reliability diagrams via kernel smoothing. In *The Twelfth International Conference on Learning Representations*.

John Blatz, Erin Fitzgerald, George Foster, Simona Gandrabur, Cyril Goutte, Alex Kulesza, Alberto Sanchis, and Nicola Ueffing. 2004. Confidence estimation for machine translation. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 315–321, Geneva, Switzerland. COLING.

Björn Böken. 2021. On the appropriateness of Platt scaling in classifier calibration. *Information Systems*, 95:101641.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30.

A. Philip Dawid. 1982. The well-calibrated Bayesian. *Journal of the American Statistical Association*, 77:605–610.

Shrey Desai and Greg Durrett. 2020. Calibration of pre-trained transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 295–302, Online. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. 2024. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 30–45, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Adam Gleave, Michael D Dennis, Shane Legg, Stuart Russell, and Jan Leike. 2021. Quantifying differences in reward functions. In *International Conference on Learning Representations*.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. *Preprint*, arxiv:1706.04599 [cs].

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. LLM-Rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 13806–13834.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. 2021. How can we know when language models know? On the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 9:962–977.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. 2022. Language models(mostly) know what they know. *arXiv preprint*. ArXiv:2207.05221 [cs].

Aviral Kumar and Sunita Sarawagi. 2019. Calibration of encoder decoder models for neural machine translation. *ArXiv*, abs/1903.00802.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2024. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in Neural Information Processing Systems*, 36.

J.I. Marcum. 1960. A statistical theory of target detection by pulsed radar. *IRE Transactions on Information Theory*, 6(2):59–267.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. 2024. Language models implement simple Word2Vec-style vector arithmetic. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5030–5047, Mexico City, Mexico. Association for Computational Linguistics.

Sabrina J. Mielke, Arthur Szlam, Emily Dinan, and Y-Lan Boureau. 2022. Reducing conversational agents' overconfidence through linguistic calibration. *Transactions of the Association for Computational Linguistics*, 10:857–872.

Elizbar A Nadaraya. 1964. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. 2014. Binary classifier calibration: Non-parametric approach. *ArXiv*, abs/1401.3390.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. 2015. Obtaining well calibrated probabilities using bayesian binning. *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, 2015:2901–2907.

Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*.

Andrew Nobel. 1996. Histogram regression estimation using data-dependent partitions. *The Annals of Statistics*, 24(3):1084 – 1105.

nostalgebraist. 2020. Interpreting GPT: The logit lens. Blogpost.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

John C. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Subhro Roy, Sam Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. 2024. BenchCLAMP: A benchmark

for evaluating language models on syntactic and semantic parsing. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Elias Stengel-Eskin, Peter Hase, and Mohit Bansal. 2024. LACIE: Listener-aware finetuning for confidence calibration in large language models. *Preprint*, arXiv:2405.21028.

Elias Stengel-Eskin and Benjamin Van Durme. 2023a. Calibrated interpretation: Confidence estimation in semantic parsing. *Transactions of the Association for Computational Linguistics*, 11:1213–1231.

Elias Stengel-Eskin and Benjamin Van Durme. 2023b. Did you mean . . . ? Confidence-based trade-offs in semantic parsing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2621–2629, Singapore.

Nishant Subramani, Samuel Bowman, and Kyunghyun Cho. 2019. Can unconditional language models recover arbitrary sentences? *Advances in Neural Information Processing Systems*, 32.

Nishant Subramani and Nivedita Suresh. 2020. Discovering useful sentence representations from large pretrained language models. *arXiv preprint arXiv:2008.09049*.

Nishant Subramani, Nivedita Suresh, and Matthew Peters. 2022. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581, Dublin, Ireland. Association for Computational Linguistics.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2017. BranchyNet: Fast inference via early exiting from deep neural networks. *Preprint*, arxiv:1709.01686 [cs]. Version: 1.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. 2023. Activation addition: Steering language models without optimization. *arXiv preprint arXiv:2308.10248*.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604, Bangkok, Thailand. Association for Computational Linguistics.

Cheng Wang. 2024. Calibration in deep learning: A survey of the state-of-the-art. *Preprint*, arXiv:2308.01222.

Shuo Wang, Zhaopeng Tu, Shuming Shi, and Yang Liu. 2020. On the inference calibration of neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3070–3079, Online. Association for Computational Linguistics.

Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Geoffrey S Watson. 1964. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. 2023. Do large language models know what they don't know? In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8653–8665, Toronto, Canada. Association for Computational Linguistics.

Alexander Yom Din, Taelin Karidi, Leshem Choshen, and Mor Geva. 2024. Jump to conclusions: Short-cutting transformers with linear transformations. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 9615–9625, Torino, Italia. ELRA and ICCL.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *Proceedings of the International Conference on Learning Representations*.

Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. Technical Report arXiv:2102.09690, arXiv. ArXiv:2102.09690 [cs] type: article.

Ruiqi Zhong, Charlie Snell, Dan Klein, and Jason Eisner. 2023. Non-programmers can label programs indirectly via active examples: A case study with text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5126–5152, Singapore. Association for Computational Linguistics.

Jiawei Zhou, Jason Eisner, Michael Newman, Emmanouil Antonios Platanios, and Sam Thomson. 2022. Online semantic parsing for latency reduction in task-oriented dialogue. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1554–1576, Dublin, Ireland. Association for Computational Linguistics.

A Feature Ablation & Analysis

Base LLM	MICE Model	smECE (↓)	Tool-Calling Utility (↑)			
			Low risk	Medium risk	High risk	AUC
Llama3-8B-I	RF Confidence Only	0.186	0.330	-0.037	-0.217	0.003
	RF First Half Layers + Confidence	0.035	0.353	0.065	-0.007	0.127
	RF Second Half Layers + Confidence	0.036	**0.357**	0.097	0.019	0.143
	RF All Layers	0.044	0.356	0.081	-0.005	0.129
	RF	0.040	0.356	**0.100**	**0.024**	**0.144**
	LR Confidence Only	0.037	0.356	0.035	0.000	0.120
	LR First Half Layers + Confidence	**0.034**	0.356	0.043	0.000	0.122
	LR Second Half Layers + Confidence	0.041	0.356	0.048	0.000	0.124
	LR All Layers	0.055	0.356	0.039	0.000	0.108
	LR	0.037	0.356	0.055	0.000	0.125
Llama3.1-8B-I	RF Confidence Only	0.183	0.182	-0.097	-0.061	-0.012
	RF First Half Layers + Confidence	0.039	0.242	0.057	0.003	0.086
	RF Second Half Layers + Confidence	**0.031**	0.247	0.067	0.017	0.098
	RF All Layers	**0.031**	0.243	0.051	0.020	0.095
	RF	0.032	**0.248**	**0.072**	**0.023**	**0.104**
	LR Confidence Only	0.043	0.239	0.016	0.000	0.064
	LR First Half Layers + Confidence	0.045	0.239	0.028	0.000	0.065
	LR Second Half Layers + Confidence	0.050	0.239	0.049	0.000	0.070
	LR All Layers	0.047	0.239	0.000	0.000	0.061
	LR	0.050	0.239	0.051	0.000	0.071
Llama3.2-3B-I	RF Confidence Only	0.158	0.196	-0.065	0.005	0.010
	RF First Half Layers + Confidence	0.034	0.236	0.064	0.011	0.089
	RF Second Half Layers + Confidence	0.041	0.235	0.059	0.009	0.084
	RF All Layers	0.046	0.236	**0.075**	**0.013**	**0.095**
	RF	0.041	**0.237**	0.071	**0.013**	**0.095**
	LR Confidence Only	**0.022**	0.233	0.000	0.000	0.057
.	LR First Half Layers + Confidence	0.033	0.231	0.023	0.000	0.072
	LR Second Half Layers + Confidence	0.026	0.233	0.012	0.000	0.064
	LR All Layers	0.044	0.233	0.037	0.000	0.071
	LR	0.025	0.232	0.016	0.000	0.073

Table 2: Results of the feature ablation on the STE test set (Wang et al., 2024). Lower smECE is better, while higher tool-calling utility is better. **Bold** indicates the best result in each category and underline indicates the second best result in each category.

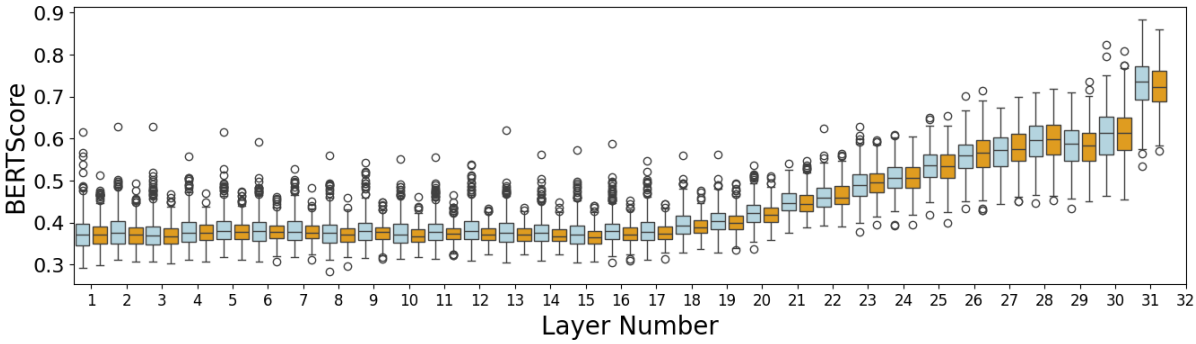

Figure 8: A version of Figure 3 that shows how positive examples (blue) and negative examples (orange) may have slightly different BERTScore distributions at each layer (see §6), which is apparently enough to inform the confidence estimator. In both figures, we use the standard boxplot function in the Seaborn package with default hyperparameters (Waskom, 2021).