VERDICT: A Library for Scaling Judge-Time Compute

Nimit Kalra
Haize Labs
New York City
nimit@haizelabs.com

Leonard Tang
Haize Labs
New York City
leonard@haizelabs.com

Abstract

The use of LLMs as automated judges ("LLM-as-a-judge") is now widespread, yet standard judges suffer from a multitude of reliability issues. To address these challenges, we introduce VERDICT[1], an open-source[2] library for scaling *judge-time compute* to enhance the accuracy, reliability, and interpretability of automated evaluators. VERDICT leverages the composition of modular reasoning units—such as verification, debate, and aggregation—and increased inference-time compute to improve LLM judge quality. Across a variety of challenging tasks such as content moderation, fact-checking, and hallucination detection, VERDICT judges achieve state-of-the-art (SOTA) or near-SOTA performance, surpassing orders-of-magnitude larger fine-tuned judges, prompted judges, and reasoning models. Ultimately, we hope VERDICT serves as a useful framework for researchers and practitioners building scalable, interpretable, and reliable LLM-based evaluators.

1 Building LLM Judges that Actually Work

Automated evaluation using LLMs, a.k.a. "LLMs-as-a-judge", is a widely adopted practice for both developers and researchers building LLM-powered applications. However, LLM judges still face a variety of reliability issues, such as inconsistent output formats, missing or miscalibrated uncertainty quantification, biases towards superficial qualities such as answer positioning, style and tone, safety, numerical frequency and preferences, the type of underlying LLM being judged, and numerous other failure modes.

To mitigate these shortcomings, we developed VERDICT, a library for building compound LLM judge systems. VERDICT provides both the primitives (`Units`) and execution framework for building

[1]Project page at https://verdict.haizelabs.com/.
[2]Code available at https://github.com/haizelabs/verdict.

such systems. Instead of a single LLM call to produce a judge result, VERDICT Judges combine multiple units of reasoning, verification, debate, and aggregation into a single judge system. When applied, these judge architectures leverage additional inference-time compute to yield impressive results on automatic evaluation of LLMs and LLM applications.

VERDICT's primary contributions are as follows:

1. VERDICT provides a **unified interface** for a potpourri of prompting strategies, bias mitigation methods, architectures, and other principles grounded in frontier research. We support ideas from the disciplines of automated evaluation, scalable oversight, safety and content moderation, fact-checking, generative reward modeling, and more.
2. VERDICT introduces **powerful reasoning primitives and patterns** for automated evaluation, such as hierarchical reasoning verification and debate-aggregation.
3. VERDICT is **fully composable**, allowing arbitrary reasoning patterns to be stacked into expressive and powerful architectures.
4. VERDICT judges require *minimal fitting* but achieve **SOTA or near-SOTA** performance on a wide variety of challenging automated evaluation evaluation tasks spanning safety moderation, factual and logical correctness, and hallucination detection.

2 Prior Art in LLM Judging

2.1 Prompted Judges

Prompted out-of-the-box LLMs are the earliest and most straightforward approach to automted evaluation. This method involves providing an LLM with specific instructions or criteria to assess outputs [2, 12, 23]. The effectiveness of prompted judges has been demonstrated in both pairwise comparison and single output scoring scenarios. One key advantage of prompted judges is their flexibility. They can be easily adapted to evaluate different aspects of LLM outputs, such as linguistic quality, content accuracy, and task-specific metrics. However, the reliability of prompted judges is often inhibited by factors such as prompt design, formatting, and in-context example selection. In the same way that underlying LLMs suffer from sensitivity, brittleness, and hallucinations, so too do prompted LLM judges [15, 18, 21].

2.2 Fine-Tuned Judges

One alternative to prompted judges is fine-tuned custom judge models. These fine-tuned judges alleviate the biases and shortcomings of prompted judges, and sometimes are also cheaper and smaller than prompted judges. Indeed, it is possible to produce fine-tuned judge models that are only 7B parameters with performance on-par with leading closed-source models [8, 9, 24]. Oftentimes, custom judges are fine-tuned to improve performance on specific tasks like fact-checking and hallucination detection, rather than general preference modeling and evaluation [17, 20].

The emerging study of generative verifiers and reward models has also provided new insights for LLM judging. Unlike traditional discriminative verifiers, generative reward models utilize next-token prediction for training, allowing it to tap into the benefits of generative LLMs. Research has demonstrated that generative reward models outperform discriminative verifiers, DPO verifiers, and LLM-as-a-Judge approaches [1, 22]. On algorithmic and math reasoning tasks, generative reward models show a 16-40% improvement in the number of problems solved using the Best-of-N method.

While these models can often times appear strong, they have been shown that to overfit their training data distribution [5, 16].

2.3 Early Compound Judges

Recent approaches have leveraged multiple LLM calls to enhance the accuracy, consistency, and robustness of evaluations. One such method is LLM debate, where multiple instances argue from different perspectives, presenting arguments and counterarguments. A final judge LLM analyzes these debates to provide a comprehensive and balanced conclusion [3, 4, 10]. This technique has contemporaneously gained traction in the scalable oversight community, particularly for improving the accuracy of less-capable judges [6, 14].

Ensemble judging, which independently queries multiple LLM judges and aggregates their assessments, has also become widely adopted. Aggregation strategies range from simple majority voting to weighted averaging based on model confidence, as well as more sophisticated fusion techniques that account for the strengths of different models [11, 19].

These early compound judge systems have demonstrated significant improvements over single-LLM methods. Smaller, weaker models can often be combined to rival or even surpass the performance of larger models. VERDICT unifies and builds upon these insights to yield even more reliable, accurate, and powerful compound judge systems.

3 VERDICT Core Concepts: Units and Orchestration

At the heart of VERDICT are:

1. Primitives for judging (`Units`), and
2. Methods for linking, orchestrating, and executing systems of `Units`.

3.1 Basic Anatomy of a Unit

A `Unit` is the fundamental building block of a judge system. They are composed of the following:

- **Prompt:** The instructions for how a `Unit` should function.
- **Model:** The model that the `Unit` calls to generate responses.
- **Scale:** The domain that generated values must be restricted to. For example, the `1-5` Likert Scale, or the `Yes/No` Scale, or the `Safe/Unsafe` Scale, or any other ordinal or categorical Scale.
- **Input Schema:** The values and types the `Unit` can accept, either from a previous `Unit` or from raw user-provided input.
- **Response Schema:** The values and types the `Unit`'s model can generate.
- **Output Schema:** The values that are ultimately generated by the `Unit`. This usually involves either postprocessing values produced by the Response Schema or updating a cumulative state from prior `Units`. These values get passed to the subsequent `Unit`.

The structure of a `Unit` solves two major pain points of standard LLM judges:

1. The output structure of the judge is *predictable* even for small language models, governed in particular by the Scale, Response Schema, and Output Schema.
2. Format and function are specified separately. The Prompt implements the evaluation logic—the function—of the `Unit`, while the Scale and Schemas manage the format of the `Unit`.

In the context of a VERDICT system, `Units` that are chained together are also automatically type-checked. The Output Schema of a `Unit` must match the Input Schema of its subsequent `Unit`. This allows for information to flow through a compound judge system in a stable and predictable fashion.

3.2 Uncertainty Quantification with Extracting Logprobs

While simple Scales like the `1-5` Likert Scale are straightforward to interpret and implement, they miss out on the rich information from a model's underlying probability distribution.

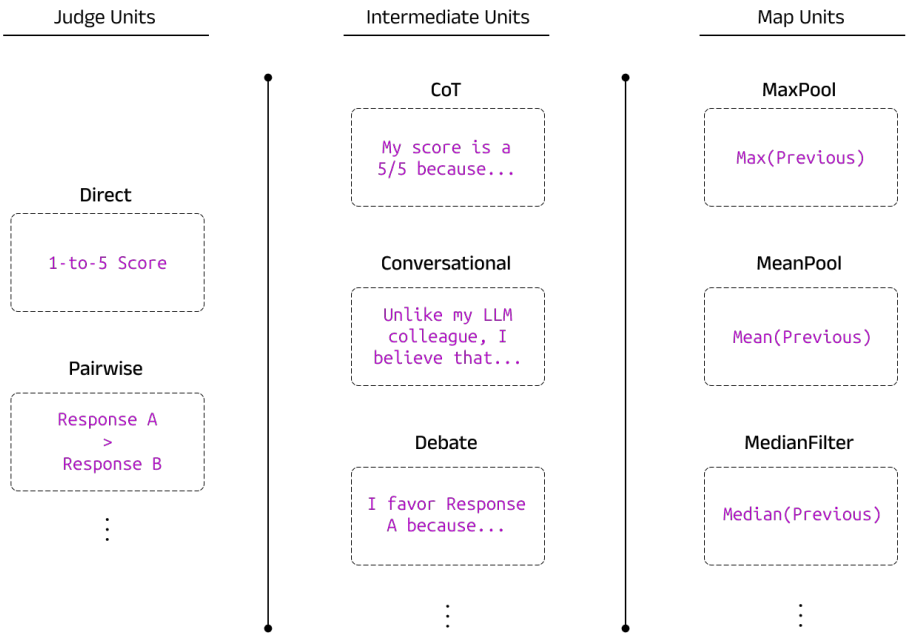

Figure 1: A few common `Units` in VERDICT, segregated by functionality. `JudgeUnits` produce a judge result (e.g. a float score or selected category); `IntermediateUnits` produce reasoning as part of a broader VERDICT system; `MapUnits` aggregate intermediate results into a final verdict.

For more precise, calibrated, and powerful judging, `Units` support accessing, managing, and aggregating the log probabilities of their corresponding Scale. For example, if using a 1−5 Likert Scale, we can access not just the token output (an integer 1 through 5), but the log probabilities across the full distribution—in this case, the log probability of each integer 1 through 5.

One can then *extract* the log probability as a final score, or *extract* a weighted sum (with potentially learned weights to further reduce bias) from the distribution. For simplicity, probabilities are not managed by default unless a `Unit` invokes an Extractor.

3.3 Common Units

`Units` are meant to be maximally flexible and amenable to any practitioner or researcher use case. Nonetheless, we implement the following `Units` that appear frequently across use cases:

Judge Units:

- **JudgeUnit:** A judge that supports any discrete Scale, such as a Likert Scale, categorical Scale, or ordinal Scale. This is the standard LLM-as-a-judge.
- **PairwiseJudgeUnit:** A judge that takes as input two strings and outputs a preferred choice between the two strings. This may also effectively be used as a categorical judge.

Intermediate Units:

- **CoTUnit:** Generates a Chain of Thought in the usual sense.
- **ConversationalUnit:** Takes as input an existing conversation (list of messages) and generates the next turn of the conversation. Generally used in tandem with other `ConversationalUnits`.
- **DebateUnit:** A subclass of `ConversationalUnit` meant for debate (arguing opposing stances) between other `DebateUnits`. Standard `ConversationalUnits` are more neutral in their discourse.

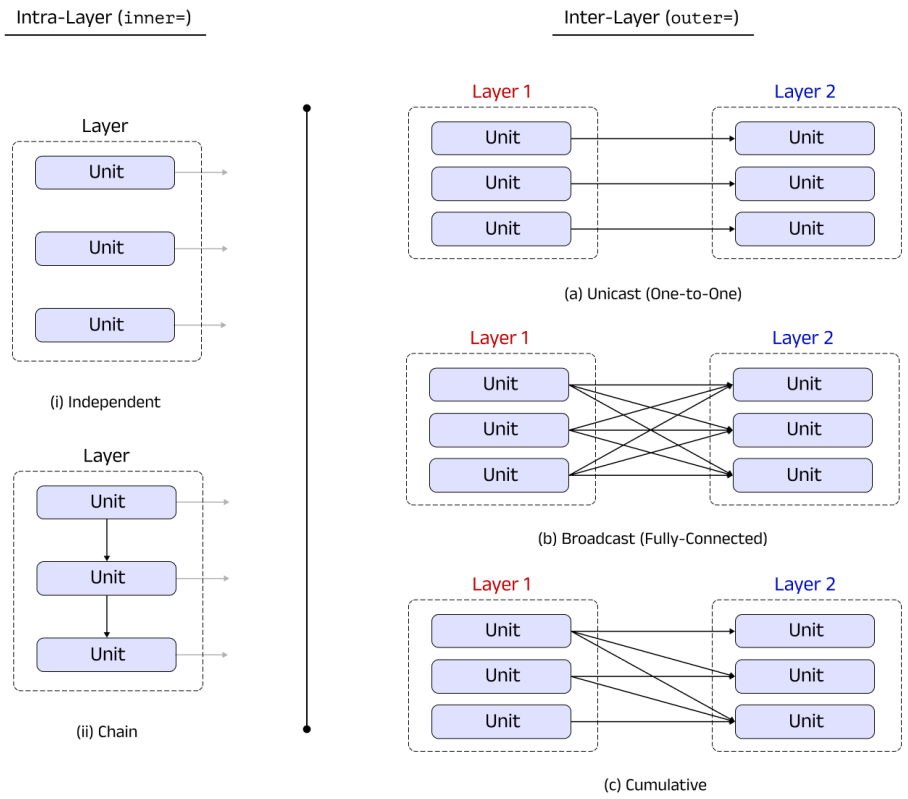

Figure 2: A few common VERDICT patterns for stitching `Units` together within and between `Layers`.

Map/Aggregate Units:

- **MapUnit:** A generic way to aggregate outputs from a previous `Layer's Units` (and optionally across several `Layers`).
- **MedianFilterUnit:** Takes the median of outputs from the previous layer of `Units`.
- **MeanPoolUnit:** Takes the mean of outputs from the previous layer of `Units`.
- **MaxPoolUnit:** Takes the max of outputs from the previous layer of `Units`.
- **MeanVariancePoolUnit:** Takes the mean and variance of outputs from the previous layer of `Units`.

3.4 Stitching Units Together

`Units` on their own only re-implement existing LLM-judge methods, albeit in a more reliable and consistent fashion. Stitching them together, however, unlocks the full power of VERDICT and increased inference-time compute. Combining `Units` with the right architecture priors can yield impressive results across a variety of evaluation, judging, and reward modeling tasks.

VERDICT draws inspiration from deep learning libraries like PyTorch vis-à-vis managing `Unit` groups and interactions. The most fundamental organizational principle is that of a `Layer`, which is a list of `Units`. This is analogous to how neural network layers are a list of neurons.

A standard `Layer` propagates information through a judge system in a *feedforward* fashion. By default, subsequent `Layer's Units` receive the output of previous `Layer's Units` in a one-to-one fashion, and `Units` within a `Layer` are fully independent of one another. However, it is possible to customize `Unit` behavior both *within* a `Layer` (using keyword `inner=`) and *between* current and subsequent `Layers` (using keyword `outer=`). Below are common instances of how `Units` can be stitched together from `Layer` to `Layer`.

4 Examples of Compound Judge Systems

To illustrate the expressiveness and ease of VERDICT, we use it to implement various examples from the LLM Judge literature with just a few lines of code[3].

4.1 Debate

In the field of scalable oversight, *debates* are often used to evaluate LLM reasoning, resolve conflicting perspectives, and identify the most robust and well-supported answers by having multiple models critique and counter-argue each other's outputs [7, 14].

The debate method involves multiple LLMs (typically two) arguing different sides of a question or task, with a judge LLM or human evaluating the debate to determine the most convincing argument. Typically, the process is as follows:

1. A question or task is presented to the debater LLMs.
2. Each debater LLM provides an initial answer and supporting arguments.
3. In subsequent rounds, debaters respond to each other's arguments, providing additional evidence or pointing out flaws in the opponent's reasoning.
4. After a set number of rounds (often 3–5), the judge reviews the entire debate transcript.
5. The judge determines which debater presented the most convincing argument and selects a winner.

VERDICT implements this relatively complex process with a `Pipeline` comprising `ConversationalUnits`, a `MapUnit`, and a `JudgeUnit`. `ConversationalUnits` sequentially on top of each to execute several rounds of debate. The `MapUnit` aggregates information from the debate history into a string format, and finally the `JudgeUnit` performs the binary classification to determine which `ConversationalUnits` prevails.

```
debate_prompt = ""
    You are participating in a debate as the {unit.role_name}.
    Here is the debate transcript thus far:
    {input.conversation}
""

Pipeline("Debate") \
    >> Layer(
        [
            ConversationalUnit(role_name="Proponent").prompt(debate_prompt),
            ConversationalUnit(role_name="Opponent").prompt(debate_prompt)
        ],
        repeat=3,
        inner="chain",
        outer="last"
    ) \
    >> JudgeUnit(BooleanScale()).prompt(""
        Evaluate the following debate transcript:
        {previous.conversation}
    "")
```

Figure 3: A VERDICT `Pipeline` for the Debate protocol. Leveraging `ConversationalUnits`, `MapUnits`, and `JudgeUnits` makes for a quick and easy implementation.

4.2 Jury: Ensemble of Judges

Instead of using a single large model—like `o1`—as a judge, one can instead use a Panel of LLM evaluators (PoLL) composed of multiple smaller models [19]. The PoLL approach of ensembling smaller, diverse language models offers a more cost-effective (seven times less expensive than GPT-4), less biased, and potentially more accurate method for evaluating LLM outputs.

[3]These judge systems are also accessible via the Haize Labs API

```
Pipeline("Jury") \
    >> Layer(
        [
        JudgeUnit(DiscreteScale([1, 5])).prompt(qa_judge_prompt).via(model)
            for model in [
                "gpt-4o-mini",
                "gpt-4o",
                "command-r",
                "command-r-plus",
                "claude-3-5-sonnet-20241022",
            ]
        ]
    ) \
    >> MeanPoolUnit()
```

Figure 4: A VERDICT `Pipeline` implementation of Jury. Creating an ensemble of models is as simple as specifying their names in a list. Individual `Unit` results are aggregated via the `MeanPoolUnit`.

4.3 G-Eval

G-Eval is an automated evaluation framework for natural language generation (NLG) tasks that uses LLMs with chain-of-thought (CoT) reasoning and a form-filling paradigm [13]. It was developed to assess the quality of NLG outputs, such as text summaries and dialogue responses, with better human alignment than previous methods. Indeed, G-Eval with GPT-4 achieves a Spearman correlation of 0.514 with human evaluations on summarization tasks.

VERDICT can implement the G-Eval `Pipeline` using a `CoTUnit`, `JudgeUnit`, and `MeanVariancePoolUnit`. Critically, both here and for all pipelines implemented with VERDICT, the input and output schemas of each `Unit` are validated and enforced throughout the `Pipeline`. This allows the ML practitioner to focus on core judging logic without needing to worry about the brittleness of message passing between LLMs.

```
scale = DiscreteScale((1, 5))
pipeline = Pipeline("GEval") \
    >> Layer(
        CoTUnit().prompt(f""
            ### Generate evaluation steps for the following task:
            {TASK}
            ### Evaluation Criteria:
            {criteria_name} ({scale}) - {criteria_description}
            ### Evaluation Steps:
        "").via("gpt-4o", retries=3, temperature=0.6).pin() \
        # .pin(): run CoTUnit once and pass shared result to JudgeUnit across all samples

        >> JudgeUnit(scale).prompt(f""
            {TASK}
            ### Evaluation Criteria:
            {criteria_name} ({scale}) - {criteria_description}
            ### Evaluation Steps:
            {{previous.thinking}}
            ### Source Text:
            {{source.source}}
            ### Summary:
            {{source.summary}}
            ### Evaluation Form (scores ONLY):
            - {criteria_name}:
        "").extract(WeightedSummedScoreExtractor()).via("gpt-4o-mini", retries=3, temperature
     =0.0)
    , repeat=5) \
    >> MeanVariancePoolUnit("score")
```

Figure 5: A VERDICT `Pipeline` implementation of G-Eval using a `CoTUnit`, `JudgeUnit`, and `MeanVariancePoolUnit`. Structure is enforced via the `Scale` property, model parameters are easily managed on each `Unit`, and the prompt can be flexibly defined inline.

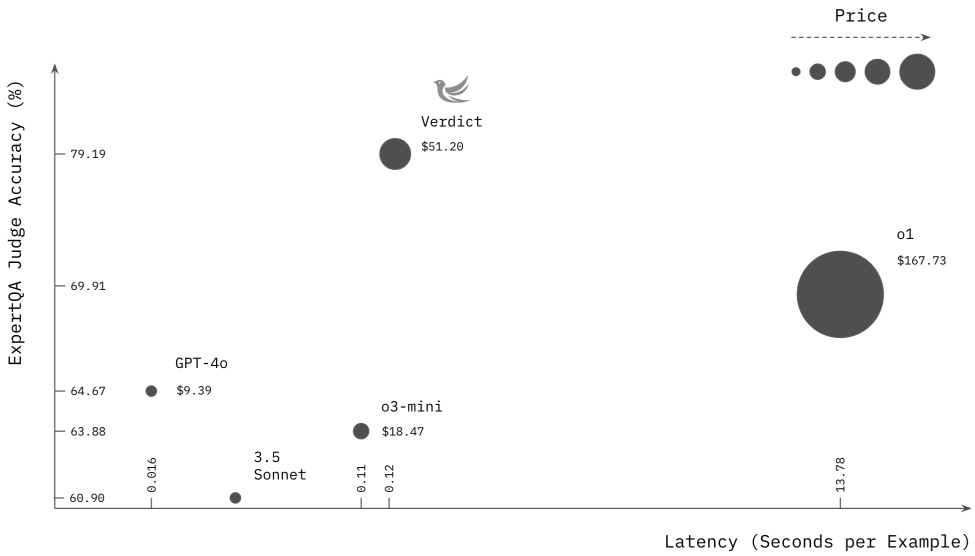

Figure 6: A simple Verdict judge beats out reasoning models like o1 (+9.28%) on the ExpertQA evaluation task while only requiring a fraction of the cost and latency.

5 Strong Results & Broad Applicability

VERDICT systems are **powerful**, **reliable**, and **versatile**.

On the tasks of content moderation, factual and logical validation, and hallucination detection, VERDICT systems achieve SOTA or near-SOTA results. This is possible simply by specifying a high-level architectural design and, critically, does not require any bespoke prompt, formatting, or hyperparameter tuning.

5.1 Safety Moderation

We first demonstrate the power of VERDICT on XSTest, a test suite containing challenging, borderline prompts that induce False Positive and False Negative safety classifications by LLMs. XSTest comprises:

- 250 safe prompts across ten prompt types that well-calibrated models should not refuse to comply with.
- 200 unsafe prompts as contrasts, which models should typically refuse for most applications

XSTest includes diverse prompt types, such as homonyms and figurative language, to test various aspects of model behavior. Each prompt is crafted as a single English sentence in question format to simulate dialogue. XSTest helps identify exaggerated safety behaviors where models refuse to respond to safe prompts, evaluates model calibration by testing responses to both safe and unsafe prompts, and highlights systematic content moderation failure modes across leading LLMs. Table 1 shows the results of the VERDICT judge vis-à-vis other judges. Our judge is a system consisting of one `CoTUnit`, two `JudgeUnits`, and one `MeanPoolUnit`.

Notably, our VERDICT judge achieves SOTA on XSTest, eclipsing even the o1 family of models.

5.2 Factual and Logical Validation

VERDICT is also useful for evaluating the factual and logical correctness of LLMs. In Table 2, we showcase how VERDICT judges can achieve competitive performance on JudgeBench, a benchmark designed to assess LLM-based judges on complex responses spanning knowledge, reasoning, math, and coding. Each example in JudgeBench consists of a response pair with corresponding preference

Table 1: Results for VERDICT Judge vs. Other Judges on XSTest. The reported VERDICT system is a pipeline of `CoTUnit` + `JudgeUnits` + `MeanPoolUnit`.

JUDGE MODEL	XSTEST SCORE
VERDICT ⇒ `CoTUnit` + `JudgeUnits` + `MeanPoolUnit`	**96.44%**
O1	96.00%
GPT-4O	96.00%
O1 MINI	95.56%
WILDGUARD	95.11%
O1 PREVIEW	94.89%
GPT-4O MINI	93.33%
LLAMA-GUARD-3-8B	90.44%
CLAUDE 3.5 SONNET	87.33%
CLAUDE 3.5 HAIKU	84.44%
LLAMA-GUARD-2-8B	83.56%
CLAUDE 3 OPUS	80.89%

labels reflecting *objective correctness*. This is unlike previous benchmarks that implicitly emphasize instruction-following and style preferences. JudgeBench is a challenging benchmark—many frontier models perform only marginally better than random guessing. However, a VERDICT architecture performs well, only second to the `o1` models.

Table 2: Results for VERDICT Judge vs. Other Judges on JudgeBench. Our VERDICT system consists of 4 `ArgumentUnits` that argue for and against each response in a response pair, and a `JudgeUnit` that determines the final VERDICT based on these arguments. All `Units` use GPT-4o.

MODEL	OVERALL SCORE
O1-PREVIEW	**75.43%**
O1-MINI	65.71%
CLAUDE-3.5-SONNET	64.29%
VERDICT ⇒ `ArgumentUnits` + `JudgeUnit`	63.55%
LLAMA-3.1-405B-INSTRUCT	56.86%
GPT-4O	56.57%
LLAMA-3.1-70B-INSTRUCT	52.29%
GPT-4O-MINI	50.00%
GEMINI-1.5-PRO	47.14%
LLAMA-3.1-8B-INSTRUCT	40.86%
GEMINI-1.5-FLASH	39.71%
CLAUDE-3-HAIKU	33.14%

5.3 Hallucination Detection

Verdict systems set a new state-of-the-art (SOTA) in hallucination detection for expert-curated QA. Table 3 compares a VERDICT system—a triplet ensemble of `JudgeUnit` models verified by another `JudgeUnit` and aggregated with a `MaxPoolUnit`—against other judges on the ExpertQA dataset.

The ExpertQA dataset is designed to evaluate the factuality of AI-generated responses in domain-specific contexts. It consists of 2,177 expert-curated questions across medicine, law, history, and engineering. The dataset includes:

- Expert-written questions that reflect real-world professional scenarios.
- Model-generated answers, evaluated by experts for factuality, attribution, and reliability.
- Expert-revised answers, ensuring factual correctness and alignment with credible sources.

An answer that is deemed not factual by an expert is considered to be a hallucination. Notably, our VERDICT system using `GPT-4o` outperforms the SOTA judge (`GPT-4o`) by +14.5%. The same VERDICT system, using a weaker backbone of `GPT-4o-mini`, still outperforms `GPT-4o` by +3.05%.

Table 3: Results for our VERDICT Judge vs. Other Judges on the LLM-AggreFact ExpertQA split. Results are taken directly from the LLM-AggreFact leaderboard. Our VERDICT Judge is an ensemble of three instances of a `JudgeUnit` with explanation followed by a self-verifier `JudgeUnit`. The results of each instance are aggregated via a `MaxPoolUnit`.

JUDGE MODEL	ACCURACY
VERDICT (GPT-4O) ⇒ `JudgeUnit` + `JudgeUnit` + `MaxPoolUnit`	**79.17%**
O1	69.91%
VERDICT (GPT-4O-MINI) ⇒ `JudgeUnit` + `JudgeUnit` + `MaxPoolUnit`	67.72%
GPT-4O	64.67%
O3-MINI	63.88%
CLAUDE-3.5 SONNET	60.9%
MISTRAL-LARGE 2	60.8%
QWEN2.5-72B-INSTRUCT	60.1%
QWQ-32B-PREVIEW	60.0%
GPT-4O-2024-05-13	59.6%
BESPOKE-MINICHECK-7B	59.2%
MINICHECK-FLAN-T5-L	59.0%
LLAMA-3.1-405B-INSTRUCT	58.5%
LLAMA-3.3-70B-INSTRUCT	58.3%
TÜLU-3-70B	55.7%

6 Interlude: VERDICT Judges as Verifiers

Ostensibly, Verdict judges are used for offline evaluation, but practically speaking Verdict judges can be used anywhere to replace human feedback and verification. Naturally, they apply to at least the following scenarios:

1. **Automated Evaluation** of AI Applications. Verdict judges enable tailored and automated evaluation of AI applications.
2. Run-Time **Guardrails**. Verdict judges are guardrails that sit on top of AI applications running in production.
3. **Test-Time Compute Scaling**. Verdict judges are verifiers that help rank, prune, and select candidates during test-time compute scaling.
4. **Reward Modeling** & Reinforcement Learning. Verdict judges provide signal in reinforcement learning — particularly in settings where rewards are not easily verifiable.

Given the recent innovation around scaling inference-time compute, reinforcement learning, and reasoning models, it is prudent to point out that VERDICT judges can indeed be leveraged as *verifiers* in these settings. VERDICT is especially well-suited for verification, given that VERDICT judges are:

1. More **general** than fine-tuned reward models. VERDICT judges readily apply across different tasks and domains, as seen by our experiments on safety moderation, checking for factual and logical correctness, and hallucination detection.
2. More **stable** and reliable compared to simple LLM judges. VERDICT judges beat out all simple LLM judges (and fine-tuned evaluators), barring the `o1` models on JudgeBench, on the three tasks presented here.
3. Capable of generating **soft rewards**, unlike formal verifiers. This is critical for extending reasoning models beyond verifiable domains like mathematics and programming.
4. Relatively **low-latency** and **cost-efficient** compared to similarly powerful judges, which is necessary for methods leveraging heavy inference-time compute.

7 Conclusion

We introduce VERDICT: a modular, expressive, and flexible approach to automated evaluation of LLM outputs. By enabling the composition of diverse reasoning units—such as verification, debate, and aggregation—VERDICT enhances the robustness, interpretability, and accuracy of LLM judges.

VERDICT judges achieve SOTA or near-SOTA across a wide range of challenging evaluation tasks, including safety moderation, factual and logical verification, and hallucination detection. Notably, VERDICT-based judges—without any bespoke customization—surpass both 1) models that are fine-tuned specifically for each evaluation task, as well as 2) orders-of-magnitude larger judge models. This highlights Verdict's potential as an efficient and scalable alternative for AI evaluation.

Beyond the immediate demonstrated performance gains, Verdict serves as a unified framework for scaling judge-time compute. We hope Verdict will enable researchers and practitioners to develop reliable, accurate, and scalable AI evaluators to advance our field into the next era of major progress.

Acknowledgments

We would like to thank Omar Khattab, Julian Michael, Jon Saad-Falcon, William Brown, Hamel Husain, Eugene Yan, and Shi Feng for their discussion and insights on scalable oversight protocols, automated evaluation, and generative reward models. We would also like to thank the gracious staff at Siena Bakehouse for fueling our work with delectable coffee and pastries.

References

[1] Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D Chang, and Prithviraj Ammanabrolu. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*, 2024.

[2] Yushi Bai, Jiahao Ying, Yixin Cao, Xin Lv, Yuze He, Xiaozhi Wang, Jifan Yu, Kaisheng Zeng, Yijia Xiao, Haozhe Lyu, et al. Benchmarking foundation models with language-model-as-an-examiner. *Advances in Neural Information Processing Systems*, 36, 2024.

[3] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.

[4] Zhumin Chu, Qingyao Ai, Yiteng Tu, Haitao Li, and Yiqun Liu. Pre: A peer review based large language model evaluator. *arXiv preprint arXiv:2401.15641*, 2024.

[5] Hui Huang, Yingqi Qu, Jing Liu, Muyun Yang, and Tiejun Zhao. An empirical study of llm-as-a-judge for llm evaluation: Fine-tuned judge models are task-specific classifiers. *arXiv preprint arXiv:2403.02839*, 2024.

[6] Zachary Kenton, Noah Y Siegel, János Kramár, Jonah Brown-Cohen, Samuel Albanie, Jannis Bulian, Rishabh Agarwal, David Lindner, Yunhao Tang, Noah D Goodman, et al. On scalable oversight with weak llms judging strong llms. *arXiv preprint arXiv:2407.04622*, 2024.

[7] Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Tim Rocktäschel, and Ethan Perez. Debating with more persuasive llms leads to more truthful answers. 2024.

[8] Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2023.

[9] Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. Prometheus 2: An open source language model specialized in evaluating other language models. *arXiv preprint arXiv:2405.01535*, 2024.

[10] Ruosen Li, Teerth Patel, and Xinya Du. Prd: Peer rank and discussion improve large language model based evaluations. *arXiv preprint arXiv:2307.02762*, 2023.

[11] Sirui Liang, Baoli Zhang, Jun Zhao, and Kang Liu. Abseval: An agent-based framework for script evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12418–12434, 2024.

[12] Yen-Ting Lin and Yun-Nung Chen. Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models. *arXiv preprint arXiv:2305.13711*, 2023.

[13] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*, 2023.

[14] Julian Michael, Salsabila Mahdi, David Rein, Jackson Petty, Julien Dirani, Vishakh Padmakumar, and Samuel R Bowman. Debate helps supervise unreliable experts. *arXiv preprint arXiv:2311.08702*, 2023.

[15] Shreya Shankar, JD Zamfirescu-Pereira, Björn Hartmann, Aditya Parameswaran, and Ian Arawjo. Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–14, 2024.

[16] Leonard Tang. Sphynx hallucination induction. https://github.com/haizelabs/sphynx, 2024.

[17] Liyan Tang, Philippe Laban, and Greg Durrett. MiniCheck: Efficient fact-checking of LLMs on grounding documents. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8818–8847, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[18] Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges. *arXiv preprint arXiv:2406.12624*, 2024.

[19] Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*, 2024.

[20] Binjie Wang, Steffi Chern, Ethan Chern, and Pengfei Liu. Halu-j: Critique-based hallucination judge. *arXiv preprint arXiv:2407.12943*, 2024.

[21] Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023.

[22] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

[23] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[24] Lianghui Zhu, Xinggang Wang, and Xinlong Wang. Judgelm: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*, 2023.