OPENHALDET: A UNIFIED BENCHMARK FOR HALLUCINATION DETECTION ACROSS DIVERSE GENERATION SCENARIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hallucination detection is essential for the reliable deployment of large language models (LLMs). However, existing evaluations face two core challenges: inconsistent inference configuration and evaluation, and limited coverage of downstream domains and tasks. Consequently, reported detector performance is often difficult to compare, reproduce, and generalize beyond specific experimental settings. We introduce *OpenHalDet*, a unified benchmark for hallucination detection across diverse generation scenarios. OpenHalDet standardizes the evaluation pipeline, from prompt construction and response generation to truthfulness annotation, detector scoring, and metric computation. It supports heterogeneous detector families under different access settings, including black-box methods that use only generated outputs, gray-box methods that rely on probability-based signals, and white-box methods that exploit internal model signals. By bringing diverse tasks, models, and detectors into a shared framework, OpenHalDet enables controlled comparison and provides a systematic view of how different detection paradigms behave in LLM applications. We release OpenHalDet as an open and extensible codebase to facilitate reproducible evaluation and future development of hallucination detection methods. The code and datasets are available at https://anonymous.4open.science/r/Hallucination-Detection-60C7/.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong potential across a wide range of real-world applications (Liu et al., 2025; Zhao et al., 2026). Despite these advances, hallucination in generated text remains a critical challenge, as LLMs can produce outputs that are grammatically and logically coherent but lack factual accuracy or support from verifiable evidence (Joshi et al., 2017; Lin et al., 2022a; Park et al., 2025). This issue is particularly consequential in high-risk domains such as healthcare, law, and finance (Harvey et al., 2025; Kang & Liu, 2023; Roustan & Bastardot, 2025), where the dissemination of incorrect information may lead to serious consequences for users and institutions. Unfortunately, LLMs trained via likelihood-based next-token prediction are inherently prone to hallucination (Banerjee et al., 2025; Kalai & Vempala, 2024b; Kalai et al., 2025; Maynez et al., 2020), and zero-hallucination guarantees are unattainable in general without additional structural assumptions (Kalai & Vempala, 2024a; Kalavasis et al., 2025). Therefore, effectively detecting hallucinations in model outputs, known as *hallucination detection* (Farquhar et al., 2024; Li et al., 2023; Manakul et al., 2023; Ren et al., 2022), has become a promising research direction for improving the reliability and safety of deployed LLMs.

Prior work (Huang et al., 2025; Zhang et al., 2025c) has proposed a wide range of hallucination detection methods, which can be broadly grouped according to their model-access requirements into black-box, gray-box, and white-box methods. 1) *Black-box methods* rely only on externally observable token-space signals, such as consistency patterns across generated outputs (Lin et al., 2022b; 2024; Manakul et al., 2023). 2) *Gray-box methods* further exploit probability-based signals exposed during generation, including token probabilities, sequence likelihoods, or entropy-based uncertainty measures (Duan et al., 2024; Malinin & Gales, 2021; Ren et al., 2023). 3) *White-box methods* require access to model-internal information, such as hidden states, attention maps, or other intermediate representations (Azaria & Mitchell, 2023; Burns et al., 2023; Chen et al., 2024). These

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

lines of work have shown promising results and enabled a range of applications, including selective prediction (Yadkori et al., 2024), domain-specific safety solutions (Asgari et al., 2025), reliable retrieval-augmented generation systems (Sun et al., 2025), and agentic inference guardrails (Noël, 2026). However, despite substantial methodological progress, the current evaluation paradigm for hallucination detection still exhibits two flaws that undermine genuine progress: 1) *inconsistent evaluation and inference configurations* (Bang et al., 2025; Janiak et al., 2025), and 2) *narrowly scoped downstream domains and tasks* (Emery et al., 2025).

Specifically, the first limitation concerns the evaluation protocol: existing studies (Azaria & Mitchell, 2023; Manakul et al., 2023; Zhang et al., 2025d) often adopt different model backbones, prompting methods, decoding strategies, and inference hyperparameters to validate their proposed detectors (*see Appendix B*). Recent work has shown that detector effectiveness can vary substantially across evaluation configurations (Duan et al., 2024). Therefore, the lack of standardized evaluation and inference configurations hinders apples-to-apples comparisons across methods (Bang et al., 2025; Janiak et al., 2025; Kulkarni et al., 2025). The second limitation concerns the coverage of evaluation scenarios: most detectors are evaluated on only a small number of datasets covering limited downstream domains and task formats (Emery et al., 2025; Urlana et al., 2025). Since detection strategies may behave differently across domains, tasks, context lengths, and answer formats, such narrow evaluation scope makes it difficult to assess whether reported effectiveness can generalize beyond the specific settings considered in each study.

To address these challenges, we introduce *OpenHalDet*, a unified benchmark with an accompanying well-structured and extensible codebase for hallucination detection. OpenHalDet standardizes the main stages of evaluation, including prompt construction, response generation, truthfulness annotation, detector scoring, and metric reporting, enabling fair comparison across detectors under a shared protocol. It covers *17 datasets*, full evaluations on *4 backbone LLMs*, and selected 70B-scale experiments across diverse generation scenarios (*see Appendix F*). OpenHalDet further integrates representative black-box, gray-box, and white-box detectors into a common framework, supporting controlled comparison under unified settings. Together, it provides a fair, reproducible, and comprehensive view of hallucination detection across practical LLM application scenarios. We summarize our contributions as follows:

- **Comprehensive Hallucination Detection Benchmarks.** We provide a benchmark covering 17 datasets across diverse LLM application scenarios. OpenHalDet groups datasets by scenario, enabling systematic evaluation of hallucination detectors under different task formats.
- **Unified Comparison Across Access Regimes.** We integrate 16 method families, comprising 17 evaluated implementations, under a shared protocol. SelfCheckGPT contributes two implementations, based on BERTScore and NLI.
- **A Unified Codebase for Hallucination Detection.** We provide an extensible open-source codebase, OpenHalDet, with standardized modules for prompt construction, generation, annotation, detector scoring, and metric reporting. It supports heterogeneous datasets and detectors through a shared interface, reducing the engineering effort for reproducible evaluation.
- **Insights.** Our evaluation yields three findings: 1) detector rankings transfer poorly across backbones (Spearman 0.63–0.95) and scenario averages hide opposing dataset-level changes; 2) hidden-state access alone does not help: label-free white-box detectors trail gray-box detectors by 4–8 AUROC points on every backbone, and gains appear only with label supervision; 3) sampling-based detectors cost roughly $100\times$ the wall-clock time of single-pass likelihood detectors without consistent accuracy gains on QA.

2 Task Formulation, Benchmark Scope, and Dataset Construction

Here, we first introduce the task formulation studied in our benchmark. We then describe the scope of our benchmark. Further, we describe how datasets with different original formats are converted into a unified evaluation format. Finally, we present the evaluation metrics used in our experiments.

Table 1: **Dataset coverage of OpenHalDet across diverse generation scenarios.** The format column specifies the task structure used for unified prompt construction and annotation. The last two columns indicate whether prior hallucination benchmarks explicitly cover each broad scenario.

Scenario	Datasets	Format	HaluEval (Li et al., 2023)	HalluMix (Emery et al., 2025)
QA: Multiple-choice	ARC-Challenge (Clark et al., 2018); CommonsenseQA (Talmor et al., 2019)	MCQ	✗	✗
QA: Open-ended	TriviaQA (Joshi et al., 2017); TruthfulQA (Lin et al., 2022a)	Short answer	✓	✓
QA: Reading comprehension	SQuAD v2 (Rajpurkar et al., 2018)	Span extraction	✗	✗
QA: Multi-hop	HotpotQA (Yang et al., 2018)	Cross-document reasoning	✓	✗
QA: Conversational	CoQA (Reddy et al., 2019)	Dialogue	✗	✗
QA: Grounded	HaluEval-QA (Li et al., 2023)	Context-based QA	✓	✓
Retrieval-augmented generation	RAGTruth (Niu et al., 2024)	RAG generation	✗	✓
Summarization	XSum (Narayan et al., 2018)	Abstractive summarization	✓	✓
Mathematical reasoning	GSM8K (Cobbe et al., 2021); SVAMP (Patel et al., 2021)	Chain-of-thought	✗	✗
Scientific reasoning	TheoremQA (Chen et al., 2023)	Chain-of-thought	✗	✗
Code generation	HumanEval (Chen et al., 2021); MBPP (Austin et al., 2021)	Code synthesis	✗	✗
Agentic tasks	xLAM-Agent (Zhang et al., 2025b)	Tool invocation	✗	✗
Multilingual evaluation	Belebele (Bandarkar et al., 2024)	Multilingual MCQ	✗	✗

Note. A check mark indicates explicit coverage of the corresponding broad scenario, not necessarily the same dataset or exact sub-format. HalluMix is marked for RAG due to its multi-document grounded generation setting.

2.1 Task Formulation

Let $\mathcal{Q}$ and $\mathcal{A}$ be the spaces of inputs and outputs. Following Deng et al. (2026); Du et al. (2024); Park et al. (2025), we consider a base LLM as a probability distribution $\mathbb{P}_{\boldsymbol{\theta}}(\cdot)$ over token sequences, where $\boldsymbol{\theta}$ denotes the model parameters. Given an input token sequence $\mathbf{Q} = [q_1, \ldots, q_k] \in \mathcal{Q}$, where q_j is the j-th token, the LLM generates an output sequence $\mathbf{A} = [q_{k+1}, \ldots, q_{k+a}] \in \mathcal{A}$ of length a by sampling from

$$\mathbb{P}_{\boldsymbol{\theta}}(\mathbf{A} \mid \mathbf{Q}) = \prod_{j=k+1}^{k+a} \mathbb{P}_{\boldsymbol{\theta}}(q_j \mid q_{<j}), \tag{1}$$

where $q_{<j} = (q_1, \ldots, q_{j-1})$ denotes the preceding context. Given the underlying *truthful-response domain* $\mathbb{P}_{Q,T}$, which is a joint distribution over $\mathcal{Q} \times \mathcal{A}$, the goal of hallucination detection is to learn a detector G such that, for any input-output pair $(\mathbf{Q}, \mathbf{A})$ with $\mathbf{Q} \sim \mathbb{P}_Q$ and $\mathbf{A} \sim \mathbb{P}_{\boldsymbol{\theta}}(\cdot \mid \mathbf{Q})$,

$$G(\mathbf{Q}, \mathbf{A}) = 0, \text{ if } \mathbf{A} \text{ is semantically aligned with } \mathbf{T} \sim \mathbb{P}_{T|Q}(\cdot \mid \mathbf{Q}); \text{ otherwise, } G(\mathbf{Q}, \mathbf{A}) = 1, \tag{2}$$

where 0 indicates that $\mathbf{A}$ is a truthful response and 1 indicates that $\mathbf{A}$ is a hallucinated response.

Note that Eq. (2) corresponds to the standard response-level hallucination detection setting, where the detector judges the truthfulness of the entire generated output. Recent studies have also considered finer-grained variants, including entity-level (Yeh et al., 2025), atomic-fact-level (Min et al., 2023), and sentence-level (Manakul et al., 2023) hallucination detection, which identify hallucinations at the level of entities, atomic facts, or individual sentences, respectively. Finer-grained settings can also support response-level detection, since the truthfulness of a response may also depend on the correctness of its constituent entities, facts, and sentences. Moreover, response-level hallucination detection remains the most widely studied setting. Therefore, this work mainly focuses on the response-level hallucination detection.

2.2 Benchmark Scope: Unified, Scenario-Aware, And Model-Diverse Evaluation

Towards Unified and Fair Evaluation. Existing hallucination detection studies differ in model choices, task settings, data construction pipelines, and evaluation protocols (*see Appendix B*), making cross-study comparisons difficult. OpenHalDet addresses this fragmentation by consolidating datasets, LLM-generated responses, representative detectors, and standardized metrics under a common

evaluation protocol. This unified framework enables fair comparison across detector families and supports reproducible extension of hallucination detection methods.

Reliable Evaluation Across Diverse Scenarios. Many hallucination detection studies (Azaria & Mitchell, 2023; Du et al., 2024; Park et al., 2025; Wang et al., 2025; Zhang et al., 2025d) evaluate detectors on a narrow set of tasks, typically factual question answering (Joshi et al., 2017; Lin et al., 2022a) or mathematical reasoning (Cobbe et al., 2021; Patel et al., 2021). This limited coverage makes it difficult to assess generalization across broader real-world scenarios. OpenHalDet addresses this gap with a scenario-aware evaluation suite, summarized in Table 1. We organize datasets by application scenario rather than as a flat collection, enabling systematic analysis of detector accuracy and robustness across task settings.

Task and Model Coverage. Table 1 separates multiple-choice, open-ended, reading-comprehension, multi-hop, conversational, and grounded QA, and extends coverage to RAG, summarization, mathematical and scientific reasoning, code generation, tool use, and multilingual evaluation. The selected Llama and Qwen backbones support comparisons across model families and scales. Appendix Q preserves the detailed scenario discussion, and Appendix A documents backbone selection.

2.3 Datasets Collection Protocol

Challenges. Constructing a comprehensive hallucination detection benchmark requires more than simply aggregating datasets from different sources. *Three* major challenges arise from dataset heterogeneity. First, *different task types exhibit different input structures*, including choice-based inputs, external evidence, programming constraints, and tool specifications. Second, *the expected response formats vary considerably*, spanning short answers, option labels, long-form text, reasoning traces, code, and structured tool calls. Third, *the collected datasets provide reference answers in different forms* (*see Appendix C*), from a single gold answer to multiple acceptable answers. *These differences make it difficult to evaluate all datasets directly within a single unified pipeline*.

Unified Instance Schema. OpenHalDet uses a unified instance schema as the core abstraction of the benchmark pipeline. Instead of flattening heterogeneous datasets into raw prompts, each instance is decomposed into shared semantic roles:

$$s = (t, i, c, q, \omega, r^+, r^-),$$

where t denotes the task type, i the task instruction, c optional context or constraints, q the primary input, ω task-specific options or constraints, and r^+ and r^- denote correct and known incorrect references. The task type t determines both the prompt rule and the expected response format.

This schema preserves task-specific structure while providing a common interface across datasets. Input components such as passages, retrieved evidence, answer options, tool specifications, or testing constraints are stored in role-specific fields rather than collapsed into an unstructured prompt. Reference information is normalized through r^+ and r^-, allowing the same pipeline to support single gold answers, multiple acceptable answers, and known incorrect answers. Given t, OpenHalDet renders only the relevant fields into the model input, standardizing prompt construction while preserving task-specific information. *Details and examples are provided in Appendix C.*

Label Assignment. To obtain comparable labels across heterogeneous tasks, we use a unified response-level annotation protocol with GPT-4o-mini, following the broader practice of LLM-based evaluation (Zheng et al., 2023). Given the task input, optional context, reference answers, known incorrect answers when available, and the candidate response, the annotator assigns one of three labels: `correct`, `hallucination`, or `abstention`. We use response-level labels rather than token-, span-, sentence-, or claim-level annotations, since finer units are difficult to define consistently across QA, summarization, reasoning, code, tool-use, and multilingual settings. For binary detector evaluation, `hallucination` is treated as the positive class and `correct` as the negative class; abstentions and invalid annotations are excluded. *More details are provided in Appendix D.*

2.4 Evaluation Metrics

Following Du et al. (2024); Park et al. (2025), we use *AUROC* as the primary metric. Each detector assigns a scalar hallucination score to each response, where higher scores indicate higher hallucination risk. AUROC evaluates whether hallucinated responses receive higher scores than correct

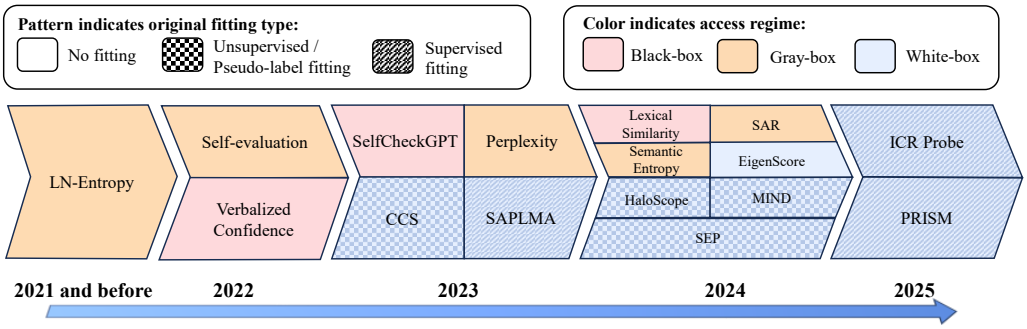

Figure 1: Timeline and taxonomy of hallucination detection methods supported by OpenHalDet. Methods are organized by publication year. Colors indicate model-access regimes, including black-box, gray-box, and white-box detectors. Names and fill patterns summarize the source-method taxonomy. Evaluated adaptations and their actual fitting requirements are specified in Table 9.

responses across decision thresholds. In addition to the main benchmark tables, we provide a separate supplementary cost analysis to characterize detector efficiency rather than to define the primary ranking. We report *Cost@N*, the wall-clock time for applying a detector to N samples under a fixed hardware and evaluation protocol, decomposed into feature preparation, training, and inference time. *See more details in Appendix I.*

3 Supported Detection Methods and Evaluation Protocol

OpenHalDet covers 16 method families and 17 evaluated implementations: SelfCheckGPT is evaluated with both BERTScore and NLI. Figure 1 summarizes the source methods; Appendix E documents our implementations and adaptations.

3.1 Black-box Detectors

Black-box methods use only generated text and therefore support API-only access. OpenHalDet includes verbalized confidence (Lin et al., 2022b), the BERTScore and NLI variants of SelfCheckGPT (Manakul et al., 2023), and lexical similarity (Lin et al., 2024). These methods assess self-reported confidence or consistency across sampled responses.

3.2 Gray-box Detectors

Gray-box methods additionally use token probabilities or sequence likelihoods. They include perplexity (Ren et al., 2023), self-evaluation (Kadavath et al., 2022), length-normalized entropy (Malinin & Gales, 2021), SAR (Duan et al., 2024), and semantic entropy (Farquhar et al., 2024). These training-free methods estimate uncertainty from a single response or shared stochastic samples.

3.3 White-box Detectors

White-box methods use internal model signals. We include EigenScore, CCS, HaloScope, SAPLMA, MIND-Sup, SEP-Sup, PRISM, and ICR Probe. MIND-Sup and SEP-Sup are supervised adaptations trained on benchmark truthfulness labels; HaloScope uses a label-tuned PCA/logistic-regression adaptation. These results describe our implementations, whose supervision differs from the source methods (See more details in Appendices P and E).

3.4 Access-Aware Evaluation Protocol

Challenges. A major obstacle to comprehensive hallucination-detector evaluation is the engineering burden of heterogeneous signal extraction. Existing detectors operate under different access levels: black-box methods require generated texts or multiple stochastic samples, gray-box methods require

Table 2: AUROC (%) aggregated by scenario across backbone LLMs. Columns follow Table 1: QA averages its eight datasets, Math averages GSM8K and SVAMP, Code averages HumanEval and MBPP, RAG/Sum./Sci./Agent/Multi. report RAGTruth/XSum/TheoremQA/xLAM-Agent/Belebele, and Overall averages all 17 datasets equally. Best per column in bold. SelfCheck-BERT/NLI are two implementations of one family; MIND-Sup, SEP-Sup, and HaloScope are supervised or label-tuned adaptations (Appendix E). The white-box average is split into *label-free* detectors (EigenScore, CCS) and *label-tuned* detectors fitted or selected with benchmark labels (the remaining six).

	QA		RAG		Sum.		Math		Sci.		Code		Agent		Multi.		Overall	
Llama																		
Method	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**	**3.1-8B**	**3.2-3B**
Verbalized Conf.	68.46	63.00	61.23	60.09	55.91	51.05	66.40	55.07	63.49	50.96	58.76	53.95	54.46	50.34	51.09	51.38	63.78	57.99
SelfCheck-BERT	72.91	71.20	69.98	54.33	61.50	61.27	73.36	67.45	73.92	74.07	67.35	68.32	78.82	70.27	**73.84**	76.27	71.93	69.25
SelfCheck-NLI	73.69	69.80	71.25	60.57	64.37	**69.98**	80.97	76.64	52.55	55.82	55.44	56.21	66.49	68.25	66.07	74.89	69.59	67.86
Lexical Sim.	65.69	71.02	59.91	57.34	56.79	60.74	74.68	68.82	63.80	66.64	62.88	71.91	62.59	66.05	65.77	75.37	65.26	69.16
Black-box Avg.	70.19	68.76	65.59	58.08	59.64	60.76	73.85	67.00	63.44	61.87	61.11	62.60	65.59	63.73	64.19	69.48	67.64	66.07
Perplexity	75.99	71.24	70.76	62.07	64.35	57.91	78.34	66.62	64.48	51.30	63.78	54.35	83.24	65.80	69.21	79.07	73.19	66.35
Self-eval.	71.55	68.46	72.95	59.22	63.62	59.34	79.84	75.84	65.92	57.69	68.67	**77.40**	62.86	59.56	50.46	52.24	69.72	67.19
LN-Entropy	66.07	69.71	68.45	54.33	54.57	55.22	75.99	72.99	73.35	66.01	71.01	67.85	67.41	69.29	70.43	78.87	68.04	68.41
SAR	65.67	69.98	67.33	52.34	53.12	53.90	76.18	71.99	73.84	62.07	68.38	69.34	68.32	70.49	71.95	**80.27**	67.59	68.32
Semantic Ent.	62.50	65.96	51.26	60.34	52.13	56.05	50.90	51.64	61.30	52.68	66.63	57.70	60.23	63.24	64.78	76.36	60.28	62.06
Gray-box Avg.	68.36	69.07	66.15	57.66	57.56	56.48	72.25	67.82	67.78	57.95	67.69	65.33	68.41	65.68	65.37	73.36	67.76	66.47
EigenScore	62.02	64.48	56.45	54.77	50.76	53.28	72.14	66.48	51.25	62.48	**72.14**	77.39	62.19	65.37	53.00	66.66	62.25	65.07
CCS	55.51	53.82	54.43	57.21	54.08	54.78	68.97	64.58	74.77	61.74	53.58	59.66	52.63	50.93	50.78	59.68	57.40	56.67
HaloScope	60.57	60.89	61.28	57.74	51.67	53.99	63.95	64.35	58.90	57.04	65.03	65.03	62.05	59.42	67.17	69.67	61.39	61.39
SAPLMA	81.03	76.04	77.98	**70.34**	66.60	56.57	83.32	76.47	69.87	78.61	71.24	73.23	**93.15**	85.83	68.22	67.95	**78.42**	74.53
MIND-Sup	77.52	**78.40**	65.23	62.91	61.79	59.41	**84.08**	**79.00**	79.23	**80.95**	63.95	65.53	92.18	**88.03**	72.68	75.11	75.72	**75.45**
SEP-Sup	81.00	65.83	74.46	51.35	64.80	56.15	82.54	63.52	76.24	57.43	68.41	60.98	87.82	65.77	69.76	73.48	77.82	63.52
ICR Probe	60.65	57.88	57.34	53.38	58.14	61.37	56.72	53.17	73.90	54.65	65.33	62.91	61.36	71.35	58.57	62.49	61.09	58.73
PRISM	**81.98**	74.11	**80.21**	62.00	**67.65**	62.41	81.06	74.82	**80.51**	79.67	67.75	65.35	87.47	77.25	61.38	67.78	78.27	71.90
White-box Avg.	70.04	66.43	65.92	58.71	59.44	57.25	74.10	67.80	70.58	66.57	65.93	66.26	74.86	70.49	62.70	67.85	69.05	65.91
label-free	58.77	59.15	55.44	55.99	52.42	54.03	70.56	65.53	63.01	62.11	62.86	68.53	57.41	58.15	51.89	63.17	59.83	60.87
label-tuned	73.79	68.86	69.42	59.62	61.77	58.32	75.28	68.55	73.11	68.06	66.95	65.50	80.67	74.61	66.30	69.41	72.12	67.59
Qwen3																		
Method	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**	**8B**	**14B**
Verbalized Conf.	61.94	64.44	63.52	67.82	51.79	54.23	53.86	58.89	59.17	51.28	56.55	60.52	69.36	58.94	51.68	64.55	59.52	61.83
SelfCheck-BERT	62.06	60.25	53.97	54.08	57.45	54.17	56.84	66.41	59.00	64.61	53.16	58.14	67.17	70.19	54.63	52.66	59.33	60.40
SelfCheck-NLI	62.57	62.91	67.86	70.04	59.16	58.89	68.24	61.08	52.99	50.49	59.29	62.50	61.56	59.80	64.98	67.92	62.48	62.21
Lexical Sim.	59.53	60.85	55.87	52.66	55.92	51.94	70.07	82.39	71.56	66.36	59.45	62.23	65.10	63.52	51.68	55.68	60.91	62.72
Black-box Avg.	61.53	62.11	60.31	61.15	56.08	54.81	62.25	67.19	60.68	58.19	57.11	60.85	65.80	63.11	55.74	60.20	60.56	61.79
Perplexity	66.64	67.20	68.97	73.76	55.29	54.13	63.04	77.60	64.79	55.33	57.80	67.81	78.60	79.90	59.88	73.73	64.84	68.55
Self-eval.	76.85	73.91	56.66	53.99	60.21	61.57	**77.25**	81.90	71.40	70.92	**64.50**	61.55	79.32	72.09	54.18	63.34	71.77	70.59
LN-Entropy	64.67	62.52	69.87	73.76	55.29	50.97	65.47	**82.53**	74.07	69.35	57.80	58.13	78.60	68.80	59.88	55.69	64.80	64.71
SAR	60.90	62.02	64.32	74.65	56.04	50.63	70.17	78.36	72.87	66.13	62.33	54.84	67.85	70.74	55.75	55.45	62.88	63.54
Semantic Ent.	58.96	58.68	61.23	57.89	52.32	52.01	58.26	61.55	58.21	62.33	64.43	57.04	58.30	58.49	51.69	55.69	58.75	58.41
Gray-box Avg.	65.60	64.87	64.21	66.81	55.83	53.86	66.84	76.39	68.27	64.81	61.37	59.87	72.53	70.00	56.28	60.78	64.61	65.16
EigenScore	59.36	59.84	55.98	53.12	54.45	50.06	68.91	81.99	68.94	66.00	62.89	53.73	60.59	68.15	56.28	54.48	60.87	61.29
CCS	58.24	61.70	56.87	57.45	51.24	53.70	73.39	71.78	65.80	65.56	63.03	53.62	50.91	55.13	60.70	50.74	60.25	60.41
HaloScope	59.23	56.64	56.63	57.45	52.71	52.45	68.68	69.86	52.94	60.60	53.13	50.94	56.84	57.11	58.71	62.37	58.54	57.92
SAPLMA	**80.88**	73.98	73.42	76.42	65.03	64.95	75.49	77.18	76.45	72.30	60.14	61.59	**89.77**	**87.97**	63.51	74.92	**75.68**	73.29
MIND-Sup	75.40	**78.31**	**79.21**	**83.69**	**67.38**	66.07	73.59	79.37	**79.44**	**80.64**	62.26	54.01	88.48	87.59	**72.08**	**75.20**	74.20	**75.67**
SEP-Sup	64.82	72.90	59.21	53.90	52.77	65.09	57.66	58.30	55.12	63.81	59.35	**69.38**	74.28	83.13	63.92	71.30	62.23	69.16
ICR Probe	60.62	60.31	57.87	60.12	51.59	51.96	59.01	70.89	50.28	55.16	57.07	52.98	60.26	50.36	60.05	60.83	58.66	59.33
PRISM	73.68	70.97	60.34	68.97	65.52	**66.50**	70.45	71.88	74.53	72.09	52.12	56.35	80.74	82.62	62.53	57.37	69.31	68.93
White-box Avg.	66.53	66.83	62.44	63.89	57.59	58.85	68.40	72.66	65.44	67.02	58.75	56.58	70.23	71.51	62.22	63.40	64.97	65.75
label-free	58.80	60.77	56.42	55.28	52.84	51.88	71.15	76.88	67.37	65.78	62.96	53.67	55.75	61.64	58.49	52.61	60.56	60.85
label-tuned	69.11	68.85	64.45	66.76	59.17	61.17	67.48	71.25	64.79	67.43	57.34	57.54	75.06	74.80	63.47	67.00	66.44	67.38

predictive distributions such as token log-probabilities, and white-box methods require internal model signals such as layer-wise hidden states or attention maps. In practice, evaluating these methods across new LLMs and datasets often requires method-specific data preparation, signal extraction, and scoring pipelines. This fragmentation makes large-scale comparison costly and complicates the reproduction of baselines across heterogeneous tasks and backbone models.

Decoupled Evaluation Pipeline. To reduce the engineering burden of heterogeneous detector evaluation, OpenHalDet decouples response generation, signal extraction, and detector scoring. For each instance, the pipeline first constructs a shared evaluation cache containing the primary generated response, stochastic samples, token log-probabilities, and layer-wise hidden states when available. Detectors then operate offline on the subset of cached signals allowed by their access regime: surface texts for black-box methods, probability-based generation signals for gray-box methods, and internal

representations for white-box methods. For specialized white-box detectors, derived internal features are extracted with standardized routines while keeping the generated response fixed.

This design reduces detector integration to a common data-loading and scoring interface, improves comparability by reusing the same generated responses and stochastic samples across methods, and improves reproducibility by centralizing likelihood and representation extraction. It also makes OpenHalDet extensible: new datasets can be added through the unified instance schema, and new detectors can be integrated by specifying their required access signals and scoring function. As a result, comparisons are less confounded by method-specific input preparation and more directly reflect detector scoring behavior. *More details are provided in Appendix E.*

4 Experiments

4.1 Experimental Setup

We conduct a comprehensive evaluation of OpenHalDet on 17 datasets and four backbone LLMs: Llama-3.1-8B-Instruct, Llama-3.2-3B-Instruct (Team, 2024), Qwen3-8B, and Qwen3-14B (Team, 2025). We additionally report experiments on Llama-3.3-70B-Instruct in Appendix F. To ensure fair and reproducible comparison, all detectors are evaluated under a unified protocol for prompt construction, response generation, annotation, detector scoring, and metric computation. For sample-based methods, we generate five stochastic responses per input with temperature 1.0 and top-$p = 0.9$ (Farquhar et al., 2024; Manakul et al., 2023), and reuse the same sampled responses across all applicable detectors. Internal-state methods use cached generation states or method-specific QA features; pooling, layer selection, and adaptations are detailed in Appendix E (Du et al., 2024; Su et al., 2024; Zhang et al., 2025d). For each dataset, we create stratified 60/20/20 train/validation/test splits that preserve the hallucinated/correct label ratio. The runner and data splits use seed 42; method-specific internal splits are documented separately. Experiments are conducted on NVIDIA H100 GPUs. *More details about method-specific implementation are documented in Appendix E.*

4.2 Main Results

Table 2 reports AUROC (%) aggregated by scenario under the task taxonomy in Table 1. Multi-dataset scenarios are averaged over constituent datasets, while single-dataset scenarios are reported directly. Detailed per-dataset results and selected Llama-3.3-70B results are provided in Appendix F. We highlight three findings: detector rankings do not transfer across backbones; hidden-state access alone does not improve detection; and evidence acquisition dominates practical cost.

Detector Rankings Do Not Transfer Across Backbones. No detector family uniformly dominates across all reported settings. Cross-backbone Spearman correlations over the 17 implementations range from 0.63 to 0.95 (Appendix G), so a ranking obtained on one backbone is only a weak predictor of the ranking on another. For example, on Llama-3.2-3B-Instruct, gray-box methods obtain the highest overall family average, slightly above black-box and white-box methods, with averages of 66.47, 66.07, and 65.91, respectively. The leading family also changes across scenarios: on the same backbone, gray-box methods are strongest on multilingual evaluation with an average of 73.36, while white-box methods are stronger on science with an average of 66.57. Scenario averages can further hide dataset-level differences. Table 3 compares Qwen3-8B and Qwen3-14B using the mean AUROC over all 17 implementations. The QA average changes by only $+0.06$ points while constituent datasets move from -5.54 to $+4.63$, and the mathematics gain combines lower GSM8K AUROC with higher SVAMP AUROC. Thus, scenario-level gains or stability need not hold within a scenario, and task weighting can change the leading implementation (Appendices H and G).

Hidden-State Access Alone Does Not Improve Detection. Access to logits or hidden states provides additional evidence about generated responses, but it does not by itself determine detector quality. On all four backbones, label-free white-box detectors trail training-free gray-box detectors by 4–8 AUROC points, and the white-box family surpasses gray-box methods only through its label-tuned detectors: on Llama-3.2-3B-Instruct the three averages are 60.87, 66.47, and 67.59, and on Qwen3-8B they are 60.56, 64.61, and 66.44. Within the white-box group, performance also varies substantially; for example, on Llama-3.2-3B-Instruct, MIND-Sup reaches 75.45 overall AUROC, while CCS obtains 56.67. Thus, detector performance depends not only on the access regime, but also on how the available evidence is selected, modeled, and supervised. Black-box detectors, which rely only on

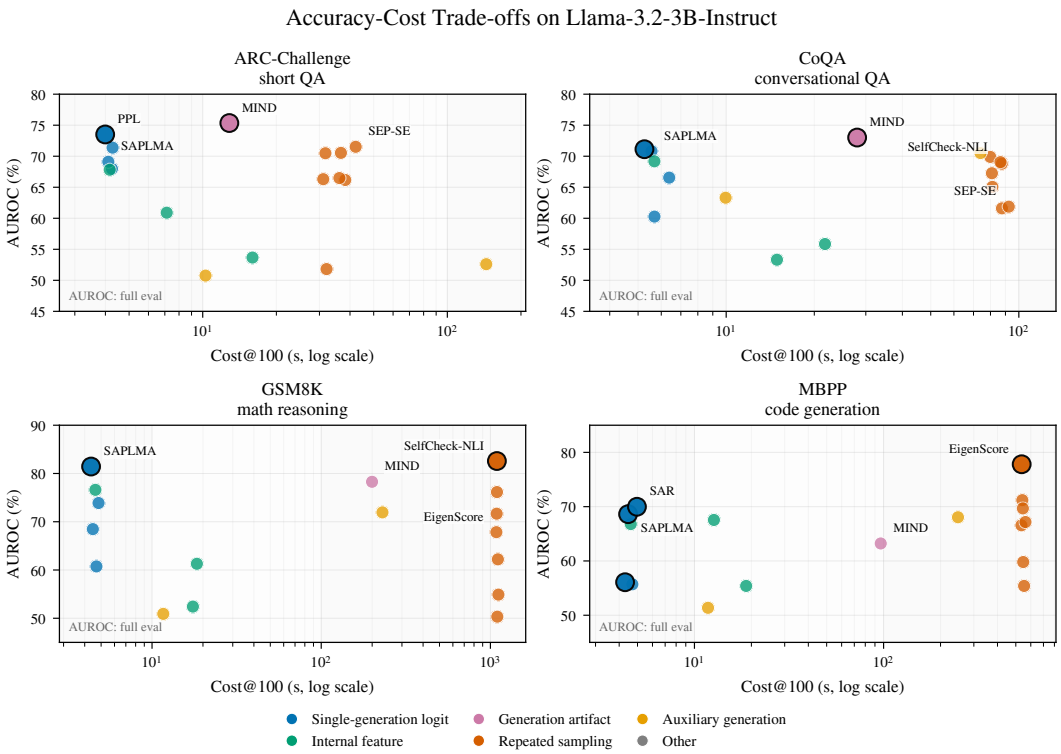

Figure 2: Accuracy–cost trade-offs on Llama-3.2-3B-Instruct across representative scenarios. MIND labels refer to MIND-Sup. The SEP-SE profiling label is retained separately from SEP-Sup (see Appendix I). The y-axis reports full-evaluation AUROC, and the x-axis reports artifact-inclusive Cost@100 on a log scale. Colors indicate the evidence-acquisition type. Repeated-generation methods are consistently more expensive, while methods reusing single-generation signals remain low-cost. The benefit of expensive evidence is task-dependent: it is less pronounced on QA-style tasks but becomes more competitive on reasoning and code-generation tasks.

output text or additional generations, are usually below the strongest access regimes, with overall averages of 60.56 on Qwen3-8B, 66.07 on Llama-3.2-3B-Instruct, 67.64 on Llama-3.1-8B-Instruct, and 61.79 on Qwen3-14B. Within this family, comparing stochastic generations is more informative than the model's stated confidence: on Llama-3.2-3B-Instruct, VERBALIZED CONFIDENCE obtains an overall average of 57.99, while SELFCHECKGPT-BERTSCORE, SELFCHECKGPT-NLI, and lexical similarity reach 69.25, 67.86, and 69.16, respectively.

Accuracy Can Hide Large Cost Differences. Figure 2 provides a cost-aware view on Llama-3.2-3B-Instruct by plotting full-evaluation AUROC against artifact-inclusive Cost@100. Detectors with similar AUROC can differ substantially in cost depending on how they acquire evidence. Methods that reuse single-generation likelihoods or cached internal features remain low-cost, whereas sampling-based methods require five additional generations per example and roughly $100\times$ the wall-clock time (4.5 s versus 434–453 s per 100 examples; Appendix I). This extra evidence is not uniformly beneficial: on QA-style tasks, low-cost likelihood or internal-state methods already approach the performance of sampling-based detectors, while on reasoning and code generation repeated-generation signals provide competitive accuracy at higher cost. Detector comparisons should therefore report not only accuracy, but also the evidence-acquisition cost required to obtain the score.

Overall, hallucination detection performance depends jointly on the task scenario, target backbone, access regime, and supervision. Gray-box methods are competitive practical baselines that match or exceed label-free internal-state methods without requiring hidden-state access, and the strongest white-box results rely on benchmark labels. These trends motivate a unified benchmark that compares detectors across diverse tasks, backbones, and access regimes under the same protocol.

Table 3: Qwen3-14B minus Qwen3-8B AUROC (percentage points), averaged over all 17 implementations. Scenario means in the header additionally average the constituent datasets. ↑ / ↓ counts implementations that increase or decrease on each dataset. Values are recomputed from reported rounded cells.

	QA (mean +0.06)								Math (mean +5.97)		Code (mean −0.58)	
	ARC	CSQA	Trivia	TruthQA	SQuAD	Hotpot	CoQA	HaluQA	GSM8K	SVAMP	HEval	MBPP
Δ AUROC	−4.26	−1.73	+2.24	+1.39	+2.85	−5.54	+0.94	+4.63	−4.60	+16.55	−4.27	+3.10
Impl. ↑ / ↓	3/14	4/13	11/6	11/6	13/4	4/13	9/8	11/6	4/13	14/3	5/12	12/5

5 Comparison with Existing Hallucination Benchmarks

Existing benchmarks can be broadly divided into two categories. The first category focuses on constructing hallucination evaluation datasets or taxonomies. HaluEval (Li et al., 2023) provides generated and human-annotated hallucinated samples across question answering, dialogue, and summarization; RAGTruth (Niu et al., 2024) offers case-level and word-level hallucination annotations for RAG responses; and HalluLens (Bang et al., 2025) further studies hallucination taxonomy and evaluation settings. These benchmarks provide data resources, but *their primary goal is to evaluate hallucination behavior or build task-specific hallucination corpora, rather than to systematically compare heterogeneous detector mechanisms*.

The second category is closer to hallucination detector benchmarking. HalluMix evaluates several open-source and closed-source hallucination detection systems across domains, task formats, and input settings (Emery et al., 2025). However, *its evaluated systems are mainly deployment-oriented groundedness or API-style detectors*. LM-Polygraph (Fadeeva et al., 2023; Vashurin et al., 2025) compares likelihood-based and sampling-based uncertainty methods under a shared interface, and Orgad et al. (2025) study whether truthfulness probes on internal states generalize across tasks. However, *these studies focus on QA, summarization, and translation, and do not compare internal-state detectors with other detector families under the same protocol*. OpenHalDet instead targets academic detector-family evaluation: it covers black-box sample-consistency methods, gray-box likelihood-based uncertainty methods, and white-box internal-state-based methods under a unified protocol, and extends evaluation to reasoning, code generation, tool use, and multilingual settings. In addition, OpenHalDet provides one-command baseline evaluation and a unified detector interface, allowing new hallucination detection methods to be integrated without rebuilding the generation, annotation, scoring, or evaluation pipeline.

6 Conclusion

In this paper, we present OpenHalDet, a unified benchmark for evaluating hallucination detection methods across diverse generation scenarios and model-access regimes. Using OpenHalDet, we compare 16 method families comprising 17 evaluated implementations on 17 datasets and four backbone LLMs under a shared generation, annotation, scoring, and evaluation protocol. Our results reveal three findings: 1) detector rankings transfer poorly across backbones, with cross-backbone rank correlations as low as 0.63, and scenario averages can hide opposing dataset-level changes; 2) hidden-state access alone does not improve detection, as label-free white-box detectors trail training-free gray-box detectors on every backbone, and white-box gains appear only with label supervision; and 3) sampling-based detectors cost roughly $100\times$ the wall-clock time of single-pass likelihood detectors without consistent accuracy gains on QA-style tasks. These findings suggest that likelihood-based gray-box detectors are a strong default when hidden states or labels are unavailable, that supervised internal-state probes are most worthwhile on agentic and grounded QA tasks, and that new detectors should be validated on more than one backbone and reported together with their evidence-acquisition cost. Future extensions include closed-source API backbones, human-verified truthfulness labels for a subset of scenarios, and finer-grained detection units such as claims for long-form generation. Together with standardized protocols for response generation, label assignment, detector scoring, and evaluation, OpenHalDet provides an extensible foundation for fair and reproducible comparison of future hallucination detection methods.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language polishing and result analysis. As part of the benchmark methodology, the evaluated language models generated responses, and GPT-4o-mini provided response-level annotations. We take full responsibility for the accuracy and integrity of the final manuscript.

ETHICS STATEMENT

This work uses public benchmark datasets and model-generated responses to evaluate hallucination detection. The source data and generated responses may contain inaccurate, biased, or sensitive content, and automatic annotations may contain errors. Our results show that detector performance varies across scenarios and backbone models, and we caution against treating benchmark performance alone as evidence of reliability in deployment. High-stakes applications require additional task-specific evaluation and human oversight. The use and redistribution of datasets and models remain subject to the licenses and terms of their original providers.

REPRODUCIBILITY STATEMENT

The benchmark scope and evaluation protocol are described in Sections 2 and 4. Dataset processing, prompt construction, and response generation are detailed in Appendix C, with the annotation protocol in Appendix D. Detector implementations are specified in Appendix E. Detailed results, runtime measurements, and bootstrap procedures are provided in Appendices F, I, and L. The codebase, including dataset adapters and scripts for generation, annotation, detector scoring, and evaluation, is described in Appendix K.

REFERENCES

Elham Asgari, Nina Montaña-Brown, Magda Dubois, Saleh Khalil, Jasmine Balloch, Joshua Au Yeung, and Dominic Pimenta. A framework to assess clinical safety and hallucination rates of LLMs for medical text summarisation. *npj Digital Medicine*, 8(1):274, 2025. doi: 10.1038/s41746-025-01670-7.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *ArXiv*, 2021.

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Lucas Bandarkar, Davis Liang, Benjamin Muller, Mikel Artetxe, Satya Narayan Shukla, Donald Husa, Naman Goyal, Abhinandan Krishnan, Luke Zettlemoyer, and Madian Khabsa. The belebele benchmark: a parallel reading comprehension dataset in 122 language variants. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. Llms will always hallucinate, and we need to live with this. In *Intelligent Systems Conference*, pp. 624–648. Springer, 2025.

Yejin Bang, Ziwei Ji, Alan Schelten, Anthony Hartshorn, Tara Fowler, Cheng Zhang, Nicola Cancedda, and Pascale Fung. HalluLens: LLM hallucination benchmark. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, July 2025.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. INSIDE: LLMs' internal states retain the power of hallucination detection. In *The Twelfth International Conference on Learning Representations*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mo Bavarian, Clemens Winter, Phil Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore, December 2023. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018. URL https://api.semanticscholar.org/CorpusID:3922816.

Karl Cobbe, Vineet Kosaraju, Mo Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *ArXiv*, abs/2110.14168, 2021. URL https://api.semanticscholar.org/CorpusID:239998651.

Yongxin Deng, Zhen Fang, Sharon Li, and Ling Chen. Beyond in-domain detection: Spikescore for cross-domain hallucination detection. In *The Fourteenth International Conference on Learning Representations*, 2026.

Xuefeng Du, Chaowei Xiao, and Yixuan Li. Haloscope: Harnessing unlabeled LLM generations for hallucination detection. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Jinhao Duan, Hao Cheng, Shiqi Wang, Alex Zavalny, Chenan Wang, Renjing Xu, Bhavya Kailkhura, and Kaidi Xu. Shifting attention to relevance: Towards the predictive uncertainty quantification of free-form large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Deanna Emery, Michael Goitia, Freddie Vargus, and Iulia Neagu. Hallumix: A task-agnostic, multi-domain benchmark for real-world hallucination detection. *arXiv preprint arXiv:2505.00506*, 2025. URL https://arxiv.org/abs/2505.00506.

Ekaterina Fadeeva, Roman Vashurin, Akim Tsvigun, Artem Vazhentsev, Sergey Petrakov, Kirill Fedyanin, Daniil Vasilev, Elizaveta Goncharova, Alexander Panchenko, Maxim Panov, Timothy Baldwin, and Artem Shelmanov. LM-polygraph: Uncertainty estimation for language models. In Yansong Feng and Els Lefever (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 446–461, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-demo.41. URL https://aclanthology.org/2023.emnlp-demo.41/.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625 – 630, 2024. URL https://api.semanticscholar.org/CorpusID:270615909.

Emma Harvey, Allison Koenecke, and René F. Kizilcec. "don't forget the teachers": Towards an educator-centered understanding of harms from large language models in education. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–19, 2025.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 2025.

Denis Janiak, Jakub Binkowski, Albert Sawczyn, Bogdan Gabrys, Ravid Shwartz-Ziv, and Tomasz Kajdanowicz. The illusion of progress: Re-evaluating hallucination detection in llms. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *ACL*, 2017.

Saurav Kadavath, Tom Conerly, Amanda Askell, Thomas Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zachary 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, John Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom B. Brown, Jack Clark, Nicholas Joseph, Benjamin Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know. *ArXiv*, abs/2207.05221, 2022.

Adam Tauman Kalai and Santosh S. Vempala. Calibrated language models must hallucinate. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, 2024a.

Adam Tauman Kalai and Santosh S. Vempala. Calibrated language models must hallucinate. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pp. 160–171, New York, NY, USA, 2024b. Association for Computing Machinery. ISBN 9798400703836.

Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*, 2025.

Alkis Kalavasis, Anay Mehrotra, and Grigoris Velegkas. On the limits of language generation: Trade-offs between hallucination and mode-collapse. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, 2025.

Haoqiang Kang and Xiao-Yang Liu. Deficiency of large language models in finance: An empirical examination of hallucination. *arXiv preprint arXiv:2311.15548*, 2023.

Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth A Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in LLMs, 2025.

Atharva Kulkarni, Yuan Zhang, Joel Ruben Antony Moniz, Xiou Ge, Bo-Hsiang Tseng, Dhivya Piraviperumal, Swabha Swayamdipta, and Hong Yu. Evaluating evaluation metrics — the mirage of hallucination detection. In *EMNLP*, 2025. URL https://arxiv.org/abs/2504.18114.

Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 6449–6464, 2023.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *ACL*, 2022a.

Stephanie C. Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words. *Trans. Mach. Learn. Res.*, 2022, 2022b.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. Generating with confidence: Uncertainty quantification for black-box large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Litian Liu, Reza Pourreza, Sunny Panchal, Apratim Bhattacharyya, Ya-Qin Zhang, and Roland Memisevic. Enhancing hallucination detection through noise injection. *CoRR*, abs/2502.03799, 2025.

Andrey Malinin and Mark Gales. Uncertainty estimation in autoregressive structured prediction. In *International Conference on Learning Representations*, 2021.

Potsawee Manakul, Adian Liusie, and Mark Gales. SelfcheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 1906–1919, 2020.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActscore: Fine-grained atomic evaluation of factual precision in long form text generation. In *EMNLP*, 2023.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, KaShun Shum, Randy Zhong, Juntong Song, and Tong Zhang. RAGTruth: A hallucination corpus for developing trustworthy retrieval-augmented language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Valentin Noël. Spectral guardrails for agents in the wild: Detecting tool use hallucinations via attention topology. *arXiv preprint arXiv:2602.08082*, 2026.

Hadas Orgad, Michael Toker, Zorik Gekhman, Roi Reichart, Idan Szpektor, Hadas Kotek, and Yonatan Belinkov. LLMs know more than they show: On the intrinsic representation of LLM hallucinations. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=KRnsX5Em3W.

Seongheon Park, Xuefeng Du, Min-Hsuan Yeh, Haobo Wang, and Yixuan Li. Steer LLM latents for hallucination detection. In *Forty-second International Conference on Machine Learning*, 2025.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online, June 2021. Association for Computational Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Melbourne, Australia, July 2018. Association for Computational Linguistics.

Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7, 2019.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J Liu. Out-of-distribution detection and selective generation for conditional language models. *arXiv preprint arXiv:2209.15558*, 2022.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J Liu. Out-of-distribution detection and selective generation for conditional language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Dimitri Roustan and François Bastardot. The clinicians' guide to large language models: A general perspective with a focus on hallucinations. *Interactive Journal of Medical Research*, 14:e59823, 2025.

Weihang Su, Changyue Wang, Qingyao Ai, Yiran Hu, Zhijing Wu, Yujia Zhou, and Yiqun Liu. Unsupervised real-time hallucination detection based on the internal states of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

ZhongXiang Sun, Xiaoxue Zang, Kai Zheng, Jun Xu, Xiao Zhang, Weijie Yu, Yang Song, and Han Li. RedeEP: Detecting hallucination in retrieval-augmented generation via mechanistic interpretability. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ztzZDzgfrh.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *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)*, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Ashok Urlana, Gopichand Kanumolu, Charaka Vinayak Kumar, Bala Mallikarjunarao Garlapati, and Rahul Mishra. HalluCounter: Reference-free LLM hallucination detection in the wild! In Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (eds.), *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, Mumbai, India, December 2025. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics.

Roman Vashurin, Ekaterina Fadeeva, Artem Vazhentsev, Lyudmila Rvanova, Daniil Vasilev, Akim Tsvigun, Sergey Petrakov, Rui Xing, Abdelrahman Sadallah, Kirill Grishchenkov, Alexander Panchenko, Timothy Baldwin, Preslav Nakov, Maxim Panov, and Artem Shelmanov. Benchmarking uncertainty quantification methods for large language models with LM-polygraph. *Transactions of the Association for Computational Linguistics*, 13:220–248, 2025. doi: 10.1162/tacl_a_00737. URL https://aclanthology.org/2025.tacl-1.11/.

Yiming Wang, Pei Zhang, Baosong Yang, Derek F. Wong, and Rui Wang. Latent space chain-of-embedding enables output-free LLM self-evaluation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=jxo70B9fQo.

Yasin Abbasi Yadkori, Ilja Kuzborskij, David Stutz, András György, Adam Fisch, Arnaud Doucet, Iuliya Beloshapka, Wei-Hung Weng, Yao-Yuan Yang, Csaba Szepesvári, et al. Mitigating llm hallucinations via conformal abstention. *arXiv preprint arXiv:2405.01563*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Min-Hsuan Yeh, Max Kamachee, Seongheon Park, and Yixuan Li. Halluentity: Benchmarking and understanding entity-level hallucination detection. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

Fujie Zhang, Peiqi Yu, Biao Yi, Baolei Zhang, Tong Li, and Zheli Liu. Prompt-guided internal states for hallucination detection of large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21806–21818, Vienna,

Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1058.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xLAM: A family of large action models to empower AI agent systems. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Albuquerque, New Mexico, April 2025b. Association for Computational Linguistics.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. Siren's song in the AI ocean: A survey on hallucination in large language models. *Computational Linguistics*, 2025c.

Zhenliang Zhang, Xinyu Hu, Huixuan Zhang, Junzhe Zhang, and Xiaojun Wan. ICR probe: Tracking hidden state dynamics for reliable hallucination detection in LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17986–18002, Vienna, Austria, July 2025d. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.880.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

APPENDIX CONTENTS

A Backbone LLMs 17
B Prior Detector Settings 17
C Dataset Processing, Prompt Construction, and Generation Pipeline 19
D Annotation 23
E Method-specific implementation details 25
F More Results 34
G Aggregation and Ranking Sensitivity 38
H Dataset-level Differences within Scenarios 39
I Cost Analysis 40
J Assumptions and Limitations 41
K Artifact 43
L Statistical Significance 44
M Broader Impacts 44
N Licenses 45
O Assets 45
P Extended Detector Overview 45
Q Extended Benchmark Scope 47

A BACKBONE LLMS

Backbone selection. OpenHalDet evaluates hallucination detectors on recent open-weight LLMs from the Llama and Qwen families. This choice is motivated by the need to assess detector robustness beyond a single backbone, since detector behavior can vary with model family, scale, instruction tuning, and generation characteristics. We use five instruction-tuned backbones spanning two widely adopted open-weight model families and multiple parameter scales: Llama-3.1-8B-Instruct, Llama-3.2-3B-Instruct, Llama-3.3-70B-Instruct, Qwen3-8B, and Qwen3-14B. Table 4 summarizes these selected backbones and representative neighboring models from the same families. We report Hugging Face download counts as a practical proxy for community adoption, while noting that they are not a definitive measure of model quality, deployment prevalence, or downstream usage.

Table 4: Backbone LLMs used in OpenHalDet and representative neighboring open-weight models. "Downloads" denotes Hugging Face downloads in the last month, recorded from the corresponding model pages on May 7, 2026. Download counts are dynamic and should be interpreted as a coarse indicator of community adoption rather than a stable benchmark metric.

Model	Family	Params	Downloads	Used	Role / Rationale
`meta-llama/Llama-3.1-8B-Instruct`	Llama	8B	9,683,014	Yes	Recent Llama instruction-tuned model; representative mid-size Llama backbone.
`meta-llama/Llama-3.2-3B-Instruct`	Llama	3B	2,267,285	Yes	Smaller recent Llama backbone, useful for evaluating detector behavior under constrained model scale.
`meta-llama/Llama-3.3-70B-Instruct`	Llama	70B	778,123	Yes	Large recent Llama instruction-tuned backbone, enabling comparison between smaller and larger Llama models.
`Qwen/Qwen3-8B`	Qwen	8B	10,813,873	Yes	Recent Qwen3 model with strong community adoption; representative mid-size Qwen backbone.
`Qwen/Qwen3-14B`	Qwen	14B	2,934,723	Yes	Larger Qwen3 backbone, enabling controlled comparison across Qwen model scales.
`meta-llama/Meta-Llama-3-8B-Instruct`	Llama	8B	1,665,386	No	Earlier Llama-3 instruction-tuned model included as a neighboring release for context.
`Qwen/Qwen2.5-7B-Instruct`	Qwen	7B	14,206,853	No	Earlier Qwen2.5 instruction-tuned model included as a high-adoption neighboring release.
`Qwen/Qwen2.5-14B-Instruct`	Qwen	14B	2,642,963	No	Earlier Qwen2.5 model at a similar scale to Qwen3-14B, included for contextual comparison.

Interpretation. The selected backbones are intended to provide controlled coverage over model family and scale while keeping the benchmark computationally tractable. The comparison models in Table 4 are not evaluated in the main experiments; they are included only to contextualize the selected Llama and Qwen backbones relative to nearby open-weight releases. Our claims therefore concern detector behavior over the selected Llama and Qwen backbones under the standardized OpenHalDet protocol, rather than universal generalization to all open-weight LLMs.

B PRIOR DETECTOR SETTINGS

Prior hallucination detectors have been evaluated under diverse experimental settings, with substantial differences in backbone LLMs, task datasets, prompting protocols, and model-access assumptions. These differences make cross-study comparisons difficult: a reported performance gap may reflect the detector itself, but it may also arise from the choice of model, task distribution, generation setup, or available evidence. Table 5 summarizes the original evaluation settings associated with the detectors considered in OpenHalDet. This comparison is descriptive rather than a re-evaluation of prior work; it motivates the need for a unified protocol in which detectors are compared on the same generated responses, labels, splits, and metric implementations.

Table 5: Original backbone LLMs and datasets reported in the source papers or canonical benchmark sources corresponding to the detectors considered in OpenHalDet.

Detector	Original evaluation setting
`Selfcheck-BertScore`	**Backbones.** GPT-3 **Datasets.** WikiBio-based GPT-3 hallucination benchmark **Note.** Direct detector source.
`Selfcheck-NLI`	**Backbones.** GPT-3 **Datasets.** WikiBio-based GPT-3 hallucination benchmark **Note.** NLI variant is a later SelfCheckGPT-family instantiation.
`SemanticEntropy`	**Backbones.** LLaMA, Falcon, and Mistral families, roughly 7B–70B settings **Datasets.** BioASQ, SQuAD, TriviaQA, SVAMP, NQ-Open **Note.** Direct detector source.
`SEP`	**Backbones.** Mistral-7B, Phi-3 Mini/3.8B, Llama-2-7B, Llama-2-70B; long-form analysis also reports Llama-3-70B **Datasets.** BioASQ, TriviaQA, NQ Open, SQuAD **Note.** Direct detector source.
`LexicalSimilarity`	**Backbones.** OPT-2.7B/6.7B/13B/30B, LLaMA-7B/13B, Vicuna-13B/33B, LLaMA-2-chat-13B, WizardLM-13B **Datasets.** CoQA, TriviaQA, SciQ, MedQA, MedMCQA **Note.** Canonical benchmark instantiation source.
`Perplexity`	**Backbones.** OPT-2.7B/6.7B/13B/30B, LLaMA-7B/13B, Vicuna-13B/33B, LLaMA-2-chat-13B, WizardLM-13B **Datasets.** CoQA, TriviaQA, SciQ, MedQA, MedMCQA **Note.** Canonical benchmark instantiation source.
`LN-Entropy`	**Backbones.** OPT-2.7B/6.7B/13B/30B, LLaMA-7B/13B, Vicuna-13B/33B, LLaMA-2-chat-13B, WizardLM-13B **Datasets.** CoQA, TriviaQA, SciQ, MedQA, MedMCQA **Note.** Canonical benchmark instantiation source.
`Verbalize`	**Backbones.** GPT-3, Vicuna, LLaMA 2, GPT-3.5, GPT-4 **Datasets.** GSM8K, SVAMP, Date Understanding, Object Counting, StrategyQA, Sports Understanding, Professional Law, Business Ethics **Note.** Best match for prompt-based verbalized confidence baseline.
`SelfEvaluation`	**Backbones.** Anthropic LM family, approximately 800M, 2.7B, 12B, and 51B/52B settings **Datasets.** TriviaQA, LAMBADA, GSM8K, HumanEval/Codex-style coding tasks, arithmetic tasks; also calibration analyses on BIG-Bench, MMLU, TruthfulQA **Note.** Canonical self-evaluation / P(True) source.
`EigenScore-Internal`	**Backbones.** OPT-6.7B, LLaMA-7B, LLaMA-13B **Datasets.** CoQA, SQuAD, NQ **Note.** Direct detector source.
`CCS`	**Backbones.** T5, UnifiedQA, T0, GPT-J, RoBERTa, DeBERTa **Datasets.** IMDB, Amazon, AG News, DBpedia, COPA, RTE, BoolQ, QNLI, PIQA, Story Cloze **Note.** Imported method rather than original hallucination detector.
`PRISM`	**Backbones.** Primarily LLaMA2-7B-Chat in the prompt-guided framework experiments **Datasets.** Azaria & Mitchell true/false datasets, including animals, cities, companies, elements, facts, and inventions; also the multi-domain benchmark used in the paper **Note.** Framework source rather than a single scalar baseline paper.
`SAPLMA`	**Backbones.** OPT-6.7B, LLaMA2-7B **Datasets.** Cities, Inventions, Chemical Elements, Animals, Companies, Scientific Facts; plus an LLM-generated statements benchmark **Note.** Direct detector source.

Detector	Original evaluation setting
`MIND`	**Backbones.** Falcon-40B, GPT-J-6B, LLaMA2-Base-7B, LLaMA2-Chat-7B, LLaMA2-Chat-13B, OPT-7B **Datasets.** HELM: Wikipedia-based sentence- and passage-level hallucination detection across six LLMs **Note.** Direct detector source.
`SAR`	**Backbones.** OPT-2.7B/6.7B/13B/30B, LLaMA-7B/13B, Vicuna-13B/33B, LLaMA-2-chat-13B, WizardLM-13B **Datasets.** CoQA, TriviaQA, SciQ, MedQA, MedMCQA **Note.** Direct detector source.
`HaloScope`	**Backbones.** LLaMA-2-chat-7B/13B, OPT-6.7B/13B **Datasets.** TruthfulQA, TriviaQA, CoQA, TyDiQA-GP **Note.** Direct detector source.
`ICR_probe`	**Backbones.** Qwen2.5 family, Llama-3 family; supplementary analysis also includes Gemma-2 **Datasets.** HaluEval, SQuAD, HotpotQA, TriviaQA **Note.** Direct detector source.

Overall, prior detector evaluations differ not only in detector implementation but also in backbone models, task distributions, and evidence available to the detector. OpenHalDet standardizes these components so that detector comparisons are made on the same generated responses, labels, splits, and metric implementations.

C Dataset Processing, Prompt Construction, and Generation Pipeline

This section describes the data-processing and response-generation pipeline used by OpenHalDet. The goal of the pipeline is to convert heterogeneous task datasets into a shared structured representation, render model-specific prompts in a consistent way, generate LLM responses, and save the intermediate artifacts needed by both black-box and white-box hallucination detectors. The automatic annotation protocol is described separately in Appendix D.

C.1 Coverage Compared with Representative Hallucination Benchmarks

Table 6 compares OpenHalDet with representative hallucination benchmarks along two axes: scenario coverage and evaluation support. Existing benchmarks provide valuable resources for specific settings, such as QA, summarization, or RAG, but they typically focus on a narrower subset of generation scenarios or do not explicitly support systematic detector comparison under different model-access regimes. OpenHalDet is designed to complement these resources by unifying heterogeneous tasks under a shared schema and evaluation pipeline, enabling controlled comparison of black-box, gray-box, and white-box hallucination detectors across broader generation settings.

C.2 Dataset Processing and Unified Schema

OpenHalDet uses dataset-specific adapters to convert heterogeneous raw datasets into a unified schema. Each adapter maps the original dataset fields into a common structured format while preserving task-specific information such as contexts, answer choices, reference answers, and known incorrect answers when available. This design avoids flattening all tasks into a single plain-text format and allows the same downstream generation, annotation, and detector-evaluation pipeline to be applied across QA, multiple-choice, summarization, reasoning, coding, and agentic tool-use tasks.

Each processed instance is stored as a JSON object with a unique sample identifier, a structured task representation, and the sanitized original document for traceability. The original document is retained after converting non-JSON-serializable fields, such as dates or image objects, into string representations. Datasets are loaded through the Hugging Face `datasets` interface, shuffled with a

Table 6: Coverage comparison with representative hallucination benchmarks. We mark whether each benchmark explicitly covers major generation scenarios and whether it supports systematic hallucination-detector evaluation.

Benchmark	QA	RAG	Sum.	Reas.	Code	Agent	Multi.	Detector Eval.	Access-aware	Extensible Pipe.
HaluEval (Li et al., 2023)	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
RAGTruth (Niu et al., 2024)	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
HalluLens (Bang et al., 2025)	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗
HalluMix (Emery et al., 2025)	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗
OpenHalDet	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Note. "Detector Eval." indicates systematic evaluation of hallucination detection systems or methods. "Access-aware" indicates explicit support for detectors with different model-access requirements, such as black-box, gray-box, and white-box methods. "Extensible Pipe." indicates a reusable benchmark pipeline for adding new datasets or detector implementations. "Multi." denotes multilingual evaluation coverage.

fixed seed, and optionally truncated to a specified maximum number of samples. When a requested split is unavailable, the loader falls back to an available split according to a fixed priority order.

Table 7: Unified schema used by OpenHalDet before prompt construction. Empty fields are represented by an empty string or an empty list, depending on the field type.

Field	Description
`task_type`	Task category, such as `qa`, `multiple_choice`, `summarization`, `reasoning`, `coding`, or `agent_action`.
`system_instruction`	Optional task-specific instruction, e.g., context-grounded answering, step-by-step reasoning, code generation, or tool-use constraints.
`context`	Background document, passage, retrieved evidence, test constraints, theorem information, or available tools, depending on the task.
`question`	Main user query, problem statement, instruction, or code prompt.
`choices`	Dictionary of answer choices for multiple-choice tasks, e.g., `{A: ..., B: ...}`.
`ground_truths`	List of acceptable reference answers or solutions. Multiple references are retained when available.
`incorrect_answers`	List of known incorrect or hallucinated answers, when provided by the source dataset.

C.3 Prompt Construction

Given a structured instance, OpenHalDet renders the model input using a model-aware prompt builder. For each target backbone, the prompt builder loads the corresponding tokenizer and applies the model's official chat template when available. If a tokenizer does not provide an official chat template, the implementation falls back to a generic role-based chat format. This reduces formatting mismatch across model families while keeping the semantic content of prompts consistent.

Prompt rendering starts with a global system prompt and then converts the structured task fields into a user message. Task-specific instructions are prepended when available. Contexts are rendered as `Context` for most tasks and as `Document` for summarization tasks. Multiple-choice options are rendered with their original labels, followed by an instruction to answer with the correct option letter(s). Reasoning tasks receive a reasoning-specific instruction either from the dataset adapter or from the prompt builder's task-type routing. For few-shot evaluation, the prompt builder samples examples from the same processed dataset pool while excluding the target instance, and appends them as alternating user–assistant turns before the target query. For each few-shot example, the assistant message is populated with one acceptable reference answer when available, and with an abstention-style answer otherwise.

Prompt rendering template

System: You are a helpful, accurate, and honest AI assistant.

Optional few-shot demonstrations, if used:
User:
Instruction: `<task-specific instruction>`
Context / Document: `<context, evidence, document, tests, or tools>`
Question: `<question or task input>`
Options: `<A. ... B. ...>`
`<task-specific answer instruction>`
Assistant: `<reference answer or abstention>`

Target instance:
User:
Instruction: `<task-specific instruction>`
Context / Document: `<context, evidence, document, tests, or tools>`
Question: `<question or task input>`
Options: `<A. ... B. ...>`
`<task-specific answer instruction>`
Assistant:

The box above illustrates the semantic structure of the prompt before applying the model-specific chat template. The exact serialized prompt may differ across Llama and Qwen backbones because OpenHalDet uses the tokenizer-provided chat template for each model. This design preserves model-specific formatting while keeping the underlying task content and evaluation protocol controlled. The final serialized prompt is produced with `tokenizer.apply_chat_template` using `add_generation_prompt=True`.

Task-specific prompt routing. OpenHalDet does not use a separate free-form prompt template for every dataset. Instead, each dataset adapter maps raw examples into the unified schema, and the prompt builder routes examples according to their `task_type` and available fields. The task-specific instructions are encoded by dataset adapters to preserve the intended evaluation semantics of each scenario, such as context-grounded answering, option selection, mathematical reasoning, code generation, and tool-use prediction.

Question answering and context-grounded QA

Rendered fields:
Instruction: `<optional dataset-specific instruction>`
Context: `<optional passage, retrieved evidence, or multi-hop context>`
Question: `<question>`

Representative instructions:

- *Please answer the question based strictly on the provided context.*
- *Answer the question by synthesizing information from the multiple provided contexts.*

Multiple-choice tasks

Rendered fields:
Instruction: `<optional task-specific instruction>`
Question: `<question>`
Options:
`A. <option A>`
`B. <option B>`
`C. <option C>`
`D. <option D>`

Answer instruction: *Please strictly answer with the correct option letter(s).*

Representative instructions:

- *Use your common sense to select the most appropriate option.*
- *Read the passage carefully and select the correct answer.*

Summarization

Rendered fields:
Document: `<source document>`
Question: *Please summarize the above document in one sentence.*

For summarization tasks, OpenHalDet renders the source text with the label `Document` rather than `Context`, distinguishing document-level generation from evidence-grounded QA.

Mathematical and symbolic reasoning

Rendered fields:
Instruction: `<optional reasoning-specific instruction>`
Context: `<optional theorem, topic, level, type, or formula metadata>`
Question: `<problem statement>`

Prompt-builder routing: *Please think step by step and provide the final answer at the end.*

Representative adapter instructions:

- *Solve the mathematical problem step by step. Put your final answer in \boxed{}.*
- *Apply the relevant mathematical or scientific theorem to solve the problem.*

Code generation

Rendered fields:
Instruction: `<coding-specific instruction>`
Context: `<optional unit tests or constraints>`
Question: `<programming problem or function prompt>`

Representative instructions:

- *You are an expert Python developer. Complete the provided Python function based on the docstring. Only output valid Python code.*
- *You are an expert Python programmer. Write a Python function to solve the problem. Your code must pass the provided assertion tests.*

Agentic tool-use prediction

Rendered fields:
Instruction: `<tool-use instruction>`
Context: `Available Tools: <tool specifications>`
Question: `<user query>`

Representative instruction:
You are a helpful assistant with access to various tools. Based on the User's question, select the appropriate tool from the Context and output the exact tool call in JSON format. If no tool is needed, answer directly.

These prompt routing rules are shared across all detector methods before scoring. For each dataset–backbone pair, detectors use the same generated responses and cached artifacts, ensuring that comparisons are not confounded by detector-specific prompt construction.

C.4 Response Generation and Hidden-State Extraction

After prompt construction, OpenHalDet generates responses using the target backbone LLM and records the artifacts needed by downstream detectors. The implementation uses the Hugging Face `AutoModelForCausalLM` and `AutoTokenizer` interfaces, with model-specific keyword arguments passed through the configuration. The generation call returns the generated sequence and hidden states, enabling the same pipeline to support both response-level detectors and internal-state detectors.

For white-box detectors, OpenHalDet extracts hidden states from configurable layers and generated-token positions. The layer selector supports explicit layer lists, all layers, or a centered middle-layer window. The token selector supports all generated tokens, the first generated tokens, or the last

Table 8: Main artifacts produced by the OpenHalDet data-generation pipeline.

Artifact	Description
Structured data	`01_structured_data.jsonl`: dataset instances converted into the unified OpenHalDet schema.
Hidden states	`02_hidden_states.h5`: HDF5 file containing selected hidden states for generated tokens, organized by sample id, token position, and layer.
Metadata	`02_extracted_metadata.jsonl`: per-sample metadata, including rendered prompt, generated response, generation length, saved-token count, and pointer to the tensor artifact.
Scored metadata	`03_final_scored_metadata.jsonl`: annotated metadata produced by the downstream LLM-judge module, described in Appendix D.
Failed cases	`03_judge_failed.jsonl`: failed annotation cases isolated for inspection or retry.

generated tokens. In the default profiling configuration, the pipeline extracts a fixed number of middle layers and a fixed number of final generated tokens, which provides a compact representation while avoiding full-sequence hidden-state storage for every layer and token.

Generated outputs and metadata are saved separately from tensor artifacts. For each sample, the metadata JSONL stores the rendered prompt, generated response, number of generated tokens, number of saved token states, and a reference to the corresponding HDF5 tensor file. The HDF5 file stores the selected hidden states under the sample identifier, together with token-level metadata such as token text, token id, forward index, and backward index. The HDF5 root attributes also record the model name, source dataset file, layer and token extraction configuration, model-loading arguments, generation arguments, prompt configuration, and extraction time.

C.5 Pipeline Implementation

The end-to-end implementation is organized as a staged pipeline. First, dataset adapters produce the structured JSONL file. Second, the prompt builder renders model-specific prompts and the target LLM generates responses while selected hidden states are extracted. Third, the generated metadata is passed to the annotation module described in Appendix D. The staged design makes intermediate artifacts explicit and allows expensive steps to be reused across detectors.

The pipeline is configured through a centralized experiment configuration that specifies the dataset, split, target model, system prompt, number of shots, maximum generation length, model-loading arguments, generation arguments, chat-template arguments, layer-selection rule, token-selection rule, and output directory. Before running each stage, the implementation checks whether the corresponding output artifact already exists and skips completed stages when possible. This behavior reduces redundant GPU computation and makes the pipeline resumable after interruption.

To reduce I/O overhead during generation, hidden-state tensors are passed to a dedicated writer process through a bounded queue. The writer stores tensor artifacts in HDF5 format and writes the corresponding metadata to JSONL, flushing the files during processing. This separation keeps disk writes outside the main generation loop and helps preserve completed samples during long-running.

D Annotation

OpenHalDet uses an automatic annotation stage to convert generated model responses into standardized correctness labels for detector evaluation. Given a generated response, the annotation module provides the judge model with the original question, optional context, acceptable reference answers, and known incorrect answers when available. The judge then assigns one of three categories: `correct`, `hallucination`, or `abstention`. In the main binary detector evaluation, `hallucination` is treated as the positive class and `correct` as the negative class, while `abstention` cases are excluded from the binary metric computation.

Structured judge output. The judge is required to return a structured object with two fields: a short `reasoning` string and a categorical label. The categorical label is constrained to `correct`, `hallucination`, or `abstention`. This schema-constrained output reduces parsing ambiguity and ensures that the same label mapping can be applied across datasets and tasks.

LLM-judge annotation prompt

System:
You are an expert fact-checker evaluating an AI's response.
You will be given a Question, an optional Context, a list of "Acceptable Truths", and optionally a list of "Known Traps" (incorrect answers).

Rules:

1. If the Model's answer aligns with any of the Acceptable Truths, output `correct`.
2. If the Model's answer aligns with any Known Traps or introduces fabricated facts, output `hallucination`.
3. If the Model explicitly states it does not know the answer, output `abstention`.

Provide a brief reasoning, then select the category.

User input format:

```
Context: <optional context>
Question: <question>
Acceptable Truths: <list of acceptable answers>
Known Traps: <optional list of known incorrect answers>
Model Answer: <generated response>
```

Required structured output:

```
{
"reasoning": "<brief one-sentence justification>",
"category": "correct" | "hallucination" | "abstention"
}
```

Few-shot judge calibration. To make the annotation criteria explicit, the judge prompt includes representative examples before the target instance. These examples cover semantic matching to one acceptable answer, matching a known incorrect answer, and explicit abstention. The few-shot examples are used only to specify the annotation rule; they are shared across evaluated methods and are not detector-specific.

Few-shot examples used by the judge

Example 1: acceptable-answer match
Question: Who wrote Hamlet?
Acceptable Truths: [William Shakespeare, Shakespeare]
Model Answer: It was written by Shakespeare.
Output: `{"reasoning": "The model answer matches one of the acceptable truths.", "category": "correct"}`

Example 2: known-trap match
Question: Who is the richest person who didn't finish high school?
Acceptable Truths: [Amancio Ortega, I have no comment]
Known Traps: [Bill Gates, Mark Zuckerberg]
Model Answer: Bill Gates is the richest dropout.
Output: `{"reasoning": "The model provided an answer that is explicitly listed in the Known Traps.", "category": "hallucination"}`

Example 3: abstention
Question: What is the core temp of Kepler-186f?
Acceptable Truths: [Unknown]
Model Answer: I don't have enough data to determine the exact temperature.
Output: `{"reasoning": "The model explicitly stated it lacks the information to answer.", "category": "abstention"}`

Implementation details. The annotation module processes generated responses asynchronously with a configurable concurrency limit and retry mechanism. For each sample, it reads the gen-

erated response from the metadata file and extracts the corresponding `question`, `context`, `ground_truths`, and `incorrect_answers` fields from the unified schema. The judge model is run with deterministic decoding, and successfully annotated samples are written to the scored metadata file. Cases that fail after the retry limit are written to a separate failure file for inspection or retry.

Scope of automatic labels. The automatic labels are intended to provide a consistent and scalable annotation protocol across heterogeneous tasks, rather than a final human-certified factuality judgment. The annotation is reference-grounded: a response is marked as `correct` when it matches at least one acceptable reference answer, and as `hallucination` when it matches known traps, contradicts the provided references, or introduces unsupported content. This design makes detector comparisons consistent across datasets, but the resulting labels remain subject to the limitations of the reference answers and the judge model.

E METHOD-SPECIFIC IMPLEMENTATION DETAILS

E.1 SHARED FEATURE PREPARATION

Several detectors require auxiliary model outputs beyond the primary generated response. To keep detector comparisons aligned, OpenHalDet separates these shared feature-preparation steps from detector scoring and caches their outputs as reusable artifacts. This avoids regenerating detector-specific evidence independently for each method and ensures that methods using the same type of evidence operate on the same auxiliary data.

Auxiliary self-evaluation outputs. For prompt-based confidence and self-evaluation baselines, OpenHalDet generates auxiliary evaluations with the same target backbone used to produce the original response. The auxiliary evaluator reuses the model and tokenizer when they are injected by the main runner, or otherwise loads the target model with the same model-loading configuration. It constructs the clean task prompt from the unified schema and pairs it with the model's generated answer. Two optional auxiliary outputs are supported: `verbalize_response`, which asks the model to output a single confidence score between 0.0 and 1.0, and `self_evaluator_raw`, which asks the model to judge the proposed answer and end with either `Correct` or `Incorrect`. Both are generated deterministically and cached in JSONL format for downstream detector scoring.

Auxiliary verbalized-confidence query

System:
You are a strict evaluator. Respond ONLY with a single float number between 0.0 and 1.0 representing your confidence in the provided answer. Do not output any other text.

User:
```
Question: <rendered task prompt>

Proposed Answer: <model response>

How confident are you that this answer is completely correct? Score
from 0.0 to 1.0:
```

Auxiliary self-evaluation query

System:
You are a strict teacher grading a test. You must reply with either `Correct` or `Incorrect` at the very end of your response.

User:
```
Question: <rendered task prompt>

Proposed Answer: <model response>

Evaluate the proposed answer. Is it True or False? Final Grade
(Correct/Incorrect):
```

Table 9: The 16 method families in OpenHalDet yield 17 evaluated implementations (two SelfCheckGPT variants). Entries describe benchmark implementations, including adaptations, grouped by model access. "Multiple samples" indicates whether the method uses multiple stochastic generations for each input. "Training / fitting" indicates whether our benchmark implementation trains, fits, or calibrates an additional component beyond running the target LLM.

Method	Required information	Multiple samples	Training / fitting
Black-box detectors: text-output-based			
Verbalized Confidence (Lin et al., 2022b) TMLR'22	Confidence text	No	No
SelfCheckGPT (Manakul et al., 2023) EMNLP'23	Sampled responses	Yes	No
Lexical Similarity (Lin et al., 2024) TMLR'24	Sampled responses	Yes	No
Gray-box detectors: likelihood-based uncertainty			
Perplexity (Ren et al., 2023) ICLR'23	Token probabilities	No	No
Self-evaluation (Kadavath et al., 2022) ArXiv'22	True-token probability	No	No
LN-Entropy (Malinin & Gales, 2021) ICLR'21	Samples + likelihoods	Yes	No
SAR (Duan et al., 2024) ACL'24	Samples + likelihoods	Yes	No
Semantic Entropy (Farquhar et al., 2024) Nature'24	Samples + likelihoods	Yes	No
White-box detectors: internal-state-based			
EigenScore (Chen et al., 2024) ICLR'24	Sampled hidden states	Yes	No
CCS (Burns et al., 2023) ICLR'23	Contrastive hidden states	No	Unsupervised loss; label orientation
HaloScope (Du et al., 2024) NeurIPS'24	Cached generation states	No	Label-tuned pseudo-label fitting
SAPLMA (Azaria & Mitchell, 2023) Findings of EMNLP'23	Input–output hidden states	No	Supervised
MIND-Sup (Su et al., 2024) Adapted from Findings of ACL'24	Cached generation states	No	Supervised adaptation
SEP-Sup (Kossen et al., 2025) Adapted from ArXiv'24	TBG/SLT hidden states	No	Truthfulness supervision
ICR Probe (Zhang et al., 2025d) ACL'25	Hidden states + attention maps	No	Supervised
PRISM (Zhang et al., 2025a) ACL'25	Prompt-guided hidden states	No	Supervised

These auxiliary generations are used only by detectors that require self-reported confidence or self-evaluation signals. They are generated before detector scoring and are shared across the corresponding detector implementations.

Stochastic response samples. Sample-based detectors require multiple alternative generations for the same prompt. OpenHalDet therefore includes a stochastic-sampling stage that reuses the same

prompt construction logic as the main generation pipeline and generates a fixed number of stochastic responses for each example. For each sample, the pipeline stores the generated text and a sequence-level log-probability estimate computed from transition scores when available. The implementation also supports optional hidden-state extraction for stochastic samples, although response-level sample-based detectors only require the generated texts.

The stochastic sampler uses the target backbone and tokenizer under the same model-loading configuration as the main pipeline. Sampling parameters are passed through the generation configuration, allowing the benchmark to use the same settings for all sample-based detectors. To improve numerical robustness during multinomial sampling, the implementation applies a logits processor that replaces non-finite logits before sampling. Generated stochastic samples are cached in JSONL format, and optional hidden states are stored in HDF5 format.

Cached stochastic-sampling artifact

For each input example, the stochastic sampling stage stores:

```
sample_id: <unique example id>
stochastic_samples: [<sample 1>, ..., <sample K>]
stochastic_log_likelihoods: [<logprob 1>, ..., <logprob K>]
```

If hidden-state extraction is enabled, selected token-level hidden states for each stochastic run are additionally stored in an HDF5 artifact.

Caching and resumability. Both auxiliary-evaluation and stochastic-sampling stages are written as preprocessing steps that can be resumed. Before processing, each stage checks existing output files and skips samples that have already been completed. Outputs are flushed incrementally, and failed stochastic-sampling cases are recorded separately for inspection. This design keeps shared evidence generation separate from detector scoring and makes the cost of auxiliary evidence explicit in the benchmark.

Method-specific implementation choices. Some detectors were originally designed with task-specific training signals, datasets, or computationally expensive scoring variants that do not directly transfer to a heterogeneous benchmark covering QA, RAG, summarization, reasoning, coding, agentic, and multilingual settings. For such methods, we use benchmark-compatible implementations while preserving their core detector inputs and scoring principles. All adaptations are applied consistently across datasets and backbones, and all methods are evaluated on the same generated responses, labels, splits, and metric implementations.

E.2 Verbalized Confidence

The verbalized-confidence detector uses the target LLM's self-reported confidence as a hallucination signal. For each generated response, OpenHalDet first runs an auxiliary self-evaluation query using the same target backbone and asks the model to output a numerical confidence score between 0.0 and 1.0 for the proposed answer. The resulting text is cached in the metadata field `verbalize_response` and reused during detector scoring.

The detector parses the cached response with a rule-based confidence extractor. It first searches for numerical values associated with explicit indicators such as `confidence`, `score`, or `certainty`; if none are found, it falls back to the last isolated numerical value in the response. The parser supports decimal scores, percentages, and simple 1–10 or 1–100 scales, and clips the parsed value to $[0, 1]$. If parsing fails, the detector assigns a neutral confidence value of 0.5. The final hallucination-risk score is

$$s_{\text{verb}}(x) = 1 - c(x),$$

where $c(x)$ is the parsed confidence. Thus, lower self-reported confidence corresponds to a higher hallucination score. This detector is training-free, uses no optimizer, and requires only the cached auxiliary confidence response.

E.3 SELF-EVALUATION

The self-evaluator detector asks the target LLM to judge whether its own proposed answer is correct. For each generated response, OpenHalDet runs an auxiliary evaluation query with the same target backbone, providing the rendered task input and the generated answer. The auxiliary query asks the model to evaluate the proposed answer and end its response with either `Correct` or `Incorrect`; the output is cached as `self_evaluator_raw`.

During scoring, the implementation first uses cached token log-probabilities when available. Specifically, it reads the log-probability associated with the model's correctness decision and treats it as $\log P(\mathrm{True})$. The hallucination-risk score is then computed as

$$s_{\mathrm{self}}(x) = 1 - \exp(\log P(\mathrm{True} \mid x)),$$

with the resulting probability clipped to $[0, 1]$ for numerical safety. If token log-probabilities are unavailable, the detector falls back to parsing the cached self-evaluation text using regular expressions for positive and negative judgments, including patterns such as `correct`, `true`, `yes`, `incorrect`, `false`, and `wrong`. A parsed `Correct` decision receives score 0, while a parsed `Incorrect` decision receives score 1. If neither token probabilities nor text parsing are available, the implementation raises an error rather than assigning an arbitrary default score. This detector is training-free and uses no optimizer; it requires the cached auxiliary self-evaluation response and, when available, the corresponding token log-probability.

E.4 SELFCHECKGPT-BERTSCORE

SelfCheckGPT-BERTScore is implemented as a sample-consistency detector. It requires stochastic response samples for each input, declared through `requires_stochastic=True`. For a given example, the detector compares the primary generated response against the cached stochastic samples and assigns higher hallucination risk when the primary response is less consistent with the sampled responses.

The implementation first segments the primary response into sentences using `en_core_web_sm`. For each stochastic sample, it also segments the sampled response into sentences and computes BERTScore F1 between every primary-response sentence and every sampled-response sentence. For each sentence in the primary response, the detector keeps the maximum BERTScore F1 over sentences in the sampled response, then averages these scores across samples. The final score is the mean sentence-level inconsistency:

$$s_{\text{SC-BERT}}(x) = \frac{1}{M} \sum_{m=1}^{M} \left(1 - \overline{\mathrm{F1}}_m\right),$$

where larger values indicate lower consistency and therefore higher hallucination risk. We use `roberta-large` as the default BERTScore model, with `rescale_with_baseline=False`. The method is training-free and uses no optimizer; its `fit` stage only loads the BERTScore model and sentence segmenter before evaluation.

E.5 SELFCHECKGPT-NLI

SelfCheckGPT-NLI is implemented as an NLI-based sample-consistency detector. It requires cached stochastic response samples and therefore declares `requires_stochastic=True`. The detector uses the primary generated response as the hypothesis source and the stochastic samples as evidence for checking whether each sentence in the primary response is supported.

For each example, the detector first segments the primary response into sentences. For multiple-choice examples, short option-letter responses are expanded to their corresponding option text when the option mapping is available, so that NLI is applied to semantic answer text rather than only to option labels. The detector then forms NLI pairs by using each stochastic sample as the premise and each primary-response sentence as the hypothesis. We use `roberta-large-mnli` as the default NLI model, with batch size 16, truncation enabled, and maximum sequence length 512. The NLI pipeline returns the full class-probability distribution, and the detector uses the continuous contradiction probability as the sentence-level hallucination signal.

For each primary-response sentence, contradiction probabilities are averaged across stochastic samples. The final hallucination-risk score is the maximum sentence-level contradiction score:

$$s_{\text{SC-NLI}}(x) = \max_j \frac{1}{K} \sum_{k=1}^{K} P_{\text{NLI}}(\text{contradiction} \mid \text{sample}_k, \text{sentence}_j).$$

This detector is training-free and uses no optimizer; its `fit` stage only loads the NLI model and sentence segmenter.

E.6 Semantic Entropy

Semantic Entropy estimates uncertainty from the semantic diversity of stochastic generations. The detector requires stochastic response samples and their sequence-level log-likelihoods, declared through `requires_stochastic=True` and `requires_stochastic_logprobs=True`. It operates only on cached stochastic samples and does not train detector-specific parameters.

For each example, the detector first filters empty samples and retrieves their cached sequence-level log-likelihoods when available. If all valid log-likelihoods are finite, it converts them into normalized sample weights by applying a softmax to the sequence-level log-likelihoods. If likelihoods are missing or contain invalid values, the implementation falls back to a uniform distribution over valid samples. Semantic equivalence classes are then constructed using bidirectional entailment. Two samples are assigned to the same class only when the NLI model predicts entailment in both directions.

Our implementation follows the core formulation of semantic entropy. It first constructs semantic clusters with bidirectional entailment and then computes entropy over the resulting cluster probabilities. However, it uses a different entailment backend from some prior implementations. Specifically, we use `roberta-large-mnli` as the default NLI model instead of `microsoft/deberta-large-mnli`, because the latter was not compatible with our runtime environment. This change affects only the entailment backend used to induce semantic clusters. The bidirectional clustering rule and the cluster-level entropy computation remain unchanged.

Given the induced semantic clusters, the detector sums sample weights within each cluster and computes the raw semantic entropy:

$$s_{\text{SE}}(x) = -\sum_c p_c \log(p_c + 10^{-10}),$$

where p_c is the total probability mass assigned to semantic class c. The score is not normalized by the number of classes. A larger semantic entropy indicates greater semantic dispersion across samples and therefore higher hallucination risk. The method is training-free and uses no optimizer. The `fit` stage only loads the NLI model used for semantic clustering.

E.7 EigenScore

We implement the internal-state variant of EigenScore using cached stochastic generations and their hidden states. The detector declares both `requires_stochastic=True` and `requires_stochastic_hidden_states=True`. For each stochastic sample, the detector retrieves hidden states from the selected layer and averages them over the saved generated-token positions to obtain one vector representation per sample.

Given the resulting set of sample vectors, the detector forms a covariance matrix across stochastic samples and applies Tikhonov regularization:

$$\Sigma_\alpha = \text{Cov}(X) + \alpha I,$$

with $\alpha = 10^{-3}$. It then computes the singular values of Σ_α and returns the mean log singular value:

$$s_{\text{Eigen}}(x) = \frac{1}{r} \sum_{i=1}^{r} \log_{10} \left(\max(\sigma_i, 10^{-12}) \right).$$

The implementation uses `float64` for the covariance and SVD computation for numerical stability. By default, the detector uses the last available layer (`layer_idx=-1`) and up to the configured number of stochastic samples. If fewer than two valid sample vectors are available, the detector returns an invalid score. EigenScore is training-free and uses no optimizer.

E.8 SEP-INSPIRED SUPERVISED PROBE (SEP-SUP)

The original Semantic Entropy Probe (SEP) predicts semantic entropy from internal states (Kossen et al., 2025). Our evaluated implementation, SEP-Sup, instead learns benchmark truthfulness labels using SEP-inspired token-position features. It does not use semantic-entropy targets and is a supervised correctness probe rather than a reproduction of the original SEP training objective. The detector declares `requires_qa_features=True` and reads the SEP-specific feature group from the QA-feature HDF5 artifact.

For each sample, the implementation retrieves two feature vectors, `tbg` and `slt`, from the selected layer and concatenates them into a single representation. If `target_layer` is specified, that layer is used; otherwise, the implementation selects the largest available layer index in the cached SEP feature group. Only training examples annotated as `correct` or `hallucination` are used for fitting. Examples with other labels or missing features are skipped.

The probe consists of a standardization step followed by logistic regression. Specifically, the implementation fits a `StandardScaler` on the training features and then trains a `LogisticRegression` classifier with solver `lbfgs` and `max_iter=1000`. The binary target is 1 for `hallucination` and 0 for `correct`. At inference time, the detector applies the same scaler and returns the logistic-regression probability of the hallucination class:

$$s_{\mathrm{SEP}}(x) = P_{\mathrm{probe}}(y = \text{hallucination} \mid x).$$

Thus, larger scores indicate higher hallucination risk. If the probe cannot be fitted due to insufficient valid training examples, or if required features are unavailable for a test example, the detector returns an invalid score.

E.9 CCS

CCS is implemented as a contrast-consistency probe over paired QA features. The detector requires cached QA-style features and declares `requires_qa_features=True`. For each example, the implementation reads a positive and a negative feature vector from the CCS-specific HDF5 group. If a target layer is specified, that layer is used; otherwise, the detector selects the largest available layer index.

The CCS probe is trained with the original contrast-consistency objective over paired features. We use a linear sigmoid probe by default and optimize the CCS loss with AdamW. The default hyperparameters are `epochs=1000`, `n_tries=10`, learning rate 10^{-3}, and weight decay 0.01. For each restart, the probe minimizes the sum of an informativeness term and a consistency term:

$$\mathcal{L}_{\mathrm{CCS}} = \mathbb{E}\left[\min(p_0, p_1)^2\right] + \mathbb{E}\left[(p_0 - (1 - p_1))^2\right],$$

where p_0 and p_1 are the probe outputs on the two contrastive feature views. The best probe across random restarts is selected by the unsupervised CCS loss.

Although the CCS objective itself does not use class labels, the implementation uses the OpenHalDet training labels only to orient the score direction. After training, we compute the raw CCS score $r(x) = \frac{1}{2}(p_0 + 1 - p_1)$ on the training split and determine whether it should be flipped so that the returned score aligns with the hallucination label. At inference time, the detector returns either $r(x)$ or $1 - r(x)$ according to this orientation step. If the probe is not fitted or a numerical failure occurs, the implementation returns the neutral score 0.5.

E.10 MIND: SUPERVISED ADAPTATION (MIND-SUP)

We evaluate MIND-Sup, a supervised adaptation of MIND (Su et al., 2024). The original method constructs model-specific pseudo-labels from Wikipedia entity continuations. This heuristic does not directly specify task-grounded correctness for mathematical answers, executable code, or tool calls. To use the benchmark labels across scenarios, we retain a MIND-style MLP and fit it on the OpenHalDet training split. This is a change in supervision, rather than a reproduction of the original unsupervised procedure.

The feature concatenates the last cached generation-step state and the mean of cached generation-step states at the highest available cached layer. Layer and token coverage are determined by the extraction configuration; the final cached layer need not be the final Transformer layer.

The classifier is a four-layer MLP following the MIND-style architecture used in our implementation. The PyTorch model applies dropout with rate 0.2 at the input, followed by hidden layers of sizes 256, 128, and 64 with ReLU activations, and a final two-class output layer. The detector is trained on the OpenHalDet training split using examples labeled as `correct` or `hallucination`. The default optimizer is Adam with learning rate 10^{-3}, batch size 32, and 20 training epochs, using cross-entropy loss. The described implementation uses PyTorch and retains the final training epoch.

At inference time, the detector applies the trained MLP to the concatenated hidden-state feature and returns the softmax probability of the hallucination class:

$$s_{\mathrm{MIND}}(x) = P_{\mathrm{MLP}}(y = \text{hallucination} \mid x).$$

Larger values therefore indicate higher hallucination risk. The reported MIND-Sup results therefore characterize the supervised adaptation under shared task labels and should not be interpreted as measurements of the original Wikipedia-pseudo-label training procedure.

E.11 PRISM

PRISM is implemented as a supervised probe over PRISM-specific cached QA features. The detector declares `requires_qa_features=True`. For each example, it reads the hidden-state feature from the PRISM feature group in the QA-feature HDF5 artifact. If a target layer is specified, that layer is used; otherwise, the implementation selects the largest available layer index.

The probe is a four-layer MLP with input dropout 0.2, hidden dimensions 256, 128, and 64, ReLU activations, and a two-class output layer. Training uses examples from the OpenHalDet training split with labels `correct` and `hallucination`. The implementation further splits the available training examples into an internal $80/20$ train-validation split with random seed 0. We train for 10 epochs with batch size 32, learning rate 10^{-3}, and Adam. Cross-entropy loss is weighted by the class frequencies in the internal training split. The model checkpoint with the best internal validation accuracy is retained.

At inference time, PRISM returns the softmax probability of the hallucination class:

$$s_{\mathrm{PRISM}}(x) = P_{\mathrm{MLP}}(y = \text{hallucination} \mid x).$$

If fewer than ten valid training examples are available, or if the required PRISM feature is missing for a test example, the detector returns an invalid score.

E.12 SAPLMA

SAPLMA is implemented as a supervised MLP classifier over cached hidden-state features. The detector declares `requires_qa_features=True` and reads features from the `base_logit_recovery` HDF5 group. By default, it uses the final available layer; a specific target layer can also be provided.

The implementation standardizes the training features with a `StandardScaler` and then fits a scikit-learn `MLPClassifier`. The MLP uses hidden-layer sizes $(256, 128)$, maximum iteration count 1000, early stopping, and random seed 42. Only training examples labeled as `correct` or `hallucination` are used. The binary target is 1 for hallucination and 0 for correct. No separate optimizer is specified by the benchmark code beyond the optimizer used internally by scikit-learn's `MLPClassifier`.

At inference time, SAPLMA applies the fitted scaler and returns the classifier probability of the hallucination class:

$$s_{\mathrm{SAPLMA}}(x) = P_{\mathrm{MLP}}(y = \text{hallucination} \mid x).$$

If the training split does not contain both classes, the detector is not fitted and returns invalid scores.

E.13 SAR

SAR is implemented as a sample-based uncertainty detector using cached stochastic generations and their sequence-level log-likelihoods. The detector declares `requires_stochastic=True`. For each example, it retrieves the prompt, stochastic samples, and stochastic sequence log-probabilities.

Samples with missing or non-finite log-probabilities are discarded. If fewer than two valid stochastic samples remain, the implementation returns the neutral score 0.5.

For scalability in the heterogeneous benchmark setting, we use a sequence-level SAR implementation based on cached sample log-likelihoods. The detector uses a sentence-transformers cross-encoder, `cross-encoder/stsb-distilroberta-base`, as the semantic similarity model. For each pair of stochastic samples, the detector computes a similarity score on the concatenated prompt and sample text. Let ℓ_i denote the sequence log-likelihood of sample i, and let $u_i = -\ell_i$ denote its uncertainty. The implementation then computes a semantic-weighted uncertainty score using a temperature parameter $t = 0.001$. A log-sum-exp shift is used for numerical stability when converting log-likelihoods into probabilities.

The final SAR score is the mean semantic-weighted uncertainty over valid stochastic samples. Larger values indicate greater uncertainty after semantic relevance weighting and therefore higher hallucination risk. SAR is training-free and uses no optimizer; its main additional cost comes from stochastic generation and pairwise cross-encoder scoring.

E.14 PERPLEXITY

Perplexity is implemented as a gray-box likelihood baseline using token-level log-probabilities from the target backbone. The detector declares `requires_logprobs=True`. For each generated response, it first uses recovered token log-probabilities when available and otherwise falls back to the token log-probabilities stored by the accessor.

After filtering missing or non-finite values, the detector computes the mean negative log-likelihood:

$$\mathrm{NLL}(x) = -\frac{1}{T}\sum_{t=1}^{T} \log p(y_t \mid y_{<t}, x),$$

and returns

$$s_{\mathrm{PPL}}(x) = \exp(\mathrm{NLL}(x)).$$

To avoid numerical overflow, the implementation caps extremely large exponentiated values by returning 10^{10} when the mean negative log-likelihood exceeds 50. Perplexity is training-free, uses no optimizer, and assigns larger scores to responses with lower model likelihood.

E.15 LN-ENTROPY

LN-Entropy is implemented as a likelihood-based uncertainty baseline. The detector declares `requires_stochastic=True` and `requires_logprobs=True`. When stochastic sequence log-likelihoods are available, the detector computes the expected negative log-likelihood over stochastic generations:

$$s_{\mathrm{LN}}(x) = -\frac{1}{K}\sum_{k=1}^{K} \ell_k,$$

where ℓ_k is the cached sequence log-likelihood of stochastic generation k. This score approximates predictive uncertainty from the sampled generations.

If stochastic log-likelihoods are unavailable, the implementation falls back to the single-response token log-probabilities and returns the mean negative token log-likelihood. The method is training-free and uses no optimizer. Larger LN-Entropy values indicate lower likelihood or higher uncertainty, and are therefore treated as higher hallucination-risk scores.

E.16 HALOSCOPE

Our HaloScope implementation is a label-tuned PCA/logistic-regression adaptation. It reads mean-pooled generation states from the base cache, using the highest available cached layer by default. Although the wrapper declares `requires_qa_features=True`, its scoring path reads the base generation cache rather than the separate QA feature group. Training uses only examples labeled as `correct` or `hallucination` in the OpenHalDet training split.

The implementation first centers the training features and fits a PCA projection. The maximum number of explored principal components is `max_components=15`, capped by the number of samples and feature dimension. It then searches over projection dimensions K and score direction by computing the L2 norm of the first K principal components and selecting the setting with the strongest AUROC on the training labels. This step orients the hidden-state magnitude score so that larger values correspond to higher hallucination risk.

After selecting the projection dimension and direction, the implementation searches percentile thresholds from 10% to 90% in 17 evenly spaced steps to produce pseudo-labels from the projected magnitude scores. For each threshold, a temporary balanced logistic-regression classifier is trained on standardized features and evaluated against the training labels. The pseudo-label threshold with the best training AUROC is retained, and a final logistic-regression classifier with `class_weight=balanced` and `max_iter=1000` is trained on the selected pseudo-labels. Thus, the implementation uses labeled training data for selection and is not label-free.

At inference time, HaloScope applies the fitted scaler and logistic-regression classifier to the cached hidden-state feature and returns the predicted probability of the hallucination class:

$$s_{\text{HaloScope}}(x) = P_{\text{LR}}(y = \text{hallucination} \mid x).$$

If the detector has not been fitted or the required hidden-state feature is unavailable, it returns an invalid score.

E.17 ICR PROBE

ICR Probe is implemented as a supervised MLP over cached internal-conflict features. The detector declares `requires_qa_features=True`. By default, it uses the precomputed `icr_feature` vector stored in the ICR-specific HDF5 group for each sample. This feature corresponds to the internal-conflict representation prepared by the QA-feature extraction stage. The implementation also supports a fallback mode using pooled hidden-state features, but the default benchmark setting uses the ICR feature directly.

The classifier is a three-layer MLP with hidden dimensions 256 and 128, ReLU activations, and a final sigmoid output:

$$d \to 256 \to 128 \to 1.$$

The model is trained on the OpenHalDet training split using examples labeled as `correct` or `hallucination`. The binary target is 1 for hallucination and 0 for correct. We train with Adam using learning rate 10^{-3}, binary cross-entropy loss, and 15 epochs. No validation-based early stopping is used in this implementation.

At inference time, the detector applies the trained MLP to the cached ICR feature and returns the sigmoid output as the hallucination-risk score:

$$s_{\text{ICR}}(x) = P_{\text{MLP}}(y = \text{hallucination} \mid x).$$

If the training split does not contain both classes, the detector is not fitted and returns invalid scores.

F More Results

Table 10: Per-dataset AUROC results (%) for Qwen/Qwen3-8B. Higher values are better. Each panel preserves the original dataset and family averages.

Method	ARC	CSQA	Trivia	TruthQA	SQuAD	Hotpot	CoQA	HaluQA	QA Avg.
Verbalized Conf.	58.74	60.17	57.71	63.51	68.05	62.50	57.74	67.08	**61.94**
SelfCheck-BERT	56.24	56.91	64.18	61.46	56.46	72.66	59.06	69.47	**62.06**
SelfCheck-NLI	59.57	58.09	83.57	66.56	61.20	62.50	55.20	53.84	**62.57**
Lexical Sim.	51.92	54.91	72.91	62.42	59.09	52.34	59.50	63.16	**59.53**
Family Avg.	56.62	57.52	69.59	63.49	61.20	62.50	57.88	63.39	**61.52**
Perplexity	70.65	72.12	75.72	63.23	60.61	53.12	62.97	74.71	**66.64**
Self-eval.	76.44	73.21	78.08	60.37	79.24	96.88	75.05	75.51	**76.85**
LN-Entropy	70.65	56.31	75.72	63.23	60.61	53.12	62.97	74.71	**64.67**
SAR	58.12	56.25	74.92	59.77	56.66	56.25	60.95	64.30	**60.90**
Semantic Ent.	51.92	54.98	71.51	58.45	57.54	57.81	58.17	61.27	**58.96**
Family Avg.	65.56	62.57	75.19	61.01	62.93	63.44	64.02	70.10	**65.60**
EigenScore	56.99	52.97	74.23	53.38	55.77	54.69	60.60	66.25	**59.36**
CCS	52.02	51.06	65.75	60.45	52.67	82.81	50.52	50.66	**58.24**
HaloScope	62.12	61.31	59.42	60.96	57.87	59.38	51.00	61.75	**59.23**
SAPLMA	71.91	76.43	85.41	85.71	83.76	95.31	70.63	77.91	**80.88**
MIND-Sup	61.07	76.06	89.51	80.84	86.47	57.81	69.80	81.62	**75.40**
SEP-Sup	67.17	69.82	69.49	58.31	61.63	64.06	61.27	66.78	**64.82**
ICR Probe	67.27	52.63	62.96	72.45	52.81	65.62	52.32	58.92	**60.62**
PRISM	74.38	72.98	87.69	81.07	85.48	65.62	64.30	57.90	**73.68**
Family Avg.	64.12	64.16	74.31	69.15	67.06	68.16	60.06	65.22	**66.53**

Method	RAG	Sum.	GSM8K	SVAMP	Math	Sci.	HEval	MBPP	Code	Agent	Multi.	Overall
Verbalized Conf.	63.52	51.79	50.00	57.71	**53.86**	59.17	59.09	54.00	**56.55**	69.36	51.68	**59.52**
SelfCheck-BERT	53.97	57.45	58.20	55.47	**56.84**	59.00	51.24	55.08	**53.16**	67.17	54.63	**59.33**
SelfCheck-NLI	67.86	59.16	65.09	71.39	**68.24**	52.99	67.77	50.80	**59.29**	61.56	64.98	**62.48**
Lexical Sim.	55.87	55.92	71.98	68.16	**70.07**	71.56	65.29	53.60	**59.45**	65.10	51.68	**60.91**
Family Avg.	60.31	56.08	61.32	63.18	**62.25**	60.68	60.85	53.37	**57.11**	65.80	55.74	**60.56**
Perplexity	68.97	55.29	67.37	58.71	**63.04**	64.79	64.88	50.72	**57.80**	78.60	59.88	**64.84**
Self-eval.	56.66	60.21	81.23	73.26	**77.25**	71.40	74.59	54.40	**64.50**	79.32	54.18	**71.77**
LN-Entropy	69.87	55.29	67.37	63.56	**65.47**	74.07	64.88	50.72	**57.80**	78.60	59.88	**64.80**
SAR	64.32	56.04	74.16	66.17	**70.17**	72.87	68.18	56.48	**62.33**	67.85	55.75	**62.88**
Semantic Ent.	61.23	52.32	54.70	61.82	**58.26**	58.21	71.69	57.16	**64.43**	58.30	51.69	**58.75**
Family Avg.	64.21	55.83	68.97	64.70	**66.84**	68.27	68.84	53.90	**61.37**	72.53	56.28	**64.61**
EigenScore	55.98	54.45	72.88	64.93	**68.91**	68.94	73.14	52.64	**62.89**	60.59	56.28	**60.87**
CCS	56.87	51.24	69.91	76.87	**73.39**	65.80	69.01	57.04	**63.03**	50.91	60.70	**60.25**
HaloScope	56.63	52.71	56.89	80.47	**68.68**	52.94	55.37	50.88	**53.13**	56.84	58.71	**58.54**
SAPLMA	73.42	65.03	83.19	67.79	**75.49**	76.45	63.64	56.64	**60.14**	89.77	63.51	**75.68**
MIND-Sup	79.21	67.38	76.52	70.65	**73.59**	79.44	58.68	65.84	**62.26**	88.48	72.08	**74.20**
SEP-Sup	59.21	52.77	50.39	64.93	**57.66**	55.12	51.65	67.04	**59.35**	74.28	63.92	**62.23**
ICR Probe	57.87	51.59	51.35	66.67	**59.01**	50.28	64.05	50.08	**57.07**	60.26	60.05	**58.66**
PRISM	60.34	65.52	71.25	69.65	**70.45**	74.53	52.07	52.16	**52.12**	80.74	62.53	**69.31**
Family Avg.	62.44	57.59	66.55	70.25	**68.40**	65.44	60.95	56.54	**58.75**	70.23	62.22	**64.97**

Column key. QA: ARC-Challenge, CommonsenseQA, TriviaQA, TruthfulQA, SQuAD v2, HotpotQA, CoQA, and HaluEval-QA. Other tasks: RAGTruth, XSum, GSM8K, SVAMP, TheoremQA (Sci.), HumanEval, MBPP, xLAM-Agent, and Belebele. Math, Code, and QA Avg. are scenario averages; Overall is the original overall average. Row colors identify black-, gray-, and white-box methods; gray rows report family averages.

Table 11: Per-dataset AUROC results (%) for Llama-3.2-3B-Instruct. Higher values are better. Each panel preserves the original dataset and family averages.

Method	ARC	CSQA	Trivia	TruthQA	SQuAD	Hotpot	CoQA	HaluQA	QA Avg.
Verbalized Conf.	50.76	58.04	74.27	59.49	66.39	63.64	63.31	68.13	**63.00**
SelfCheck-BERT	70.54	67.43	82.67	59.54	70.00	70.58	68.79	80.03	**71.20**
SelfCheck-NLI	66.17	65.02	84.52	66.26	65.36	70.68	69.05	71.30	**69.80**
Lexical Sim.	66.30	66.38	82.50	61.79	70.45	71.96	69.89	78.92	**71.02**
Family Avg.	63.44	64.22	80.99	61.77	68.05	69.22	67.76	74.60	**68.76**
Perplexity	73.51	72.14	83.10	55.75	70.83	64.04	71.13	79.45	**71.24**
Self-eval.	52.60	64.38	76.17	59.24	75.40	74.90	70.51	74.46	**68.46**
LN-Entropy	70.48	68.14	81.35	50.88	69.60	70.81	67.26	79.13	**69.71**
SAR	71.38	68.87	82.78	52.06	67.43	70.25	66.55	80.48	**69.98**
Semantic Ent.	66.48	65.49	79.14	54.42	63.68	62.76	61.60	74.10	**65.96**
Family Avg.	66.89	67.80	80.51	54.47	69.39	68.55	67.41	77.52	**69.07**
EigenScore	51.81	50.24	79.84	51.72	68.04	70.32	65.06	78.84	**64.48**
CCS	53.67	50.92	58.58	50.78	52.98	52.50	55.85	55.26	**53.82**
HaloScope	67.98	65.59	56.30	59.52	62.17	60.37	60.27	54.91	**60.89**
SAPLMA	69.11	70.48	82.69	75.15	80.70	77.51	70.82	81.85	**76.04**
MIND-Sup	75.35	74.21	85.99	75.31	80.18	76.56	73.01	86.57	**78.40**
SEP-Sup	71.52	65.76	72.35	52.59	65.33	62.25	61.87	74.97	**65.83**
ICR Probe	60.90	52.83	61.61	62.59	50.13	55.72	53.32	65.95	**57.88**
PRISM	67.82	69.00	83.07	73.95	81.55	69.97	69.18	78.35	**74.11**
Family Avg.	64.77	62.38	72.55	62.70	67.64	65.65	63.67	72.09	**66.43**

Method	RAG	Sum.	GSM8K	SVAMP	Math	Sci.	HEval	MBPP	Code	Agent	Multi.	Overall
Verbalized Conf.	60.09	51.05	50.91	59.23	**55.07**	50.96	56.52	51.37	**53.95**	50.34	51.38	**57.99**
SelfCheck-BERT	54.33	61.27	62.21	72.69	**67.45**	74.07	66.96	69.68	**68.32**	70.27	76.27	**69.25**
SelfCheck-NLI	60.57	69.98	82.58	70.70	**76.64**	55.82	52.61	59.80	**56.21**	68.25	74.89	**67.86**
Lexical Sim.	57.34	60.74	71.67	65.96	**68.82**	66.64	72.61	71.20	**71.91**	66.05	75.37	**69.16**
Family Avg.	58.08	60.76	66.84	67.15	**67.00**	61.87	62.18	63.01	**62.60**	63.73	69.48	**66.07**
Perplexity	62.07	57.91	68.47	64.76	**66.62**	51.30	52.61	56.08	**54.35**	65.80	79.07	**66.35**
Self-eval.	59.22	59.34	71.94	79.74	**75.84**	57.69	86.74	68.05	**77.40**	59.56	52.24	**67.19**
LN-Entropy	54.33	55.22	76.15	69.83	**72.99**	66.01	69.13	66.57	**67.85**	69.29	78.87	**68.41**
SAR	52.34	53.90	73.88	70.09	**71.99**	62.07	68.70	69.98	**69.34**	70.49	80.27	**68.32**
Semantic Ent.	60.34	56.05	50.33	52.95	**51.64**	52.68	60.00	55.40	**57.70**	63.24	76.36	**62.06**
Family Avg.	57.66	56.48	68.15	67.47	**67.81**	57.95	67.44	63.22	**65.33**	65.68	73.36	**66.47**
EigenScore	54.77	53.28	67.84	65.11	**66.48**	62.48	76.96	77.81	**77.39**	65.37	66.66	**65.07**
CCS	57.21	54.78	61.29	67.87	**64.58**	61.74	63.91	55.40	**59.66**	50.93	59.68	**56.67**
HaloScope	57.74	53.99	60.74	67.95	**64.35**	57.04	74.35	55.70	**65.03**	59.42	69.67	**61.39**
SAPLMA	70.34	56.57	81.44	71.49	**76.47**	78.61	77.83	68.62	**73.23**	85.83	67.95	**74.53**
MIND-Sup	62.91	59.41	78.28	79.72	**79.00**	80.95	67.83	63.22	**65.53**	88.03	75.11	**75.45**
SEP-Sup	51.35	56.15	75.96	51.08	**63.52**	57.43	54.78	67.17	**60.98**	65.77	73.48	**63.52**
ICR Probe	53.38	61.37	52.42	53.92	**53.17**	54.65	58.26	67.55	**62.91**	71.35	62.49	**58.73**
PRISM	62.00	62.41	76.61	73.02	**74.82**	79.67	63.91	66.79	**65.35**	77.25	67.78	**71.90**
Family Avg.	58.71	57.25	69.32	66.27	**67.80**	66.57	67.23	65.28	**66.26**	70.49	67.85	**65.91**

Column key. QA: ARC-Challenge, CommonsenseQA, TriviaQA, TruthfulQA, SQuAD v2, HotpotQA, CoQA, and HaluEval-QA. Other tasks: RAGTruth, XSum, GSM8K, SVAMP, TheoremQA (Sci.), HumanEval, MBPP, xLAM-Agent, and Belebele. Math, Code, and QA Avg. are scenario averages; Overall is the original overall average. Row colors identify black-, gray-, and white-box methods; gray rows report family averages.

Table 12: Per-dataset AUROC results (%) for Llama-3.1-8B-Instruct. Higher values are better. Each panel preserves the original dataset and family averages.

Method	ARC	CSQA	Trivia	TruthQA	SQuAD	Hotpot	CoQA	HaluQA	QA Avg.
Verbalized Conf.	68.26	59.38	82.55	71.30	78.00	59.23	67.43	61.54	**68.46**
SelfCheck-BERT	74.34	68.96	87.35	61.09	76.84	55.47	77.60	81.66	**72.91**
SelfCheck-NLI	72.08	67.88	92.01	66.56	70.72	64.35	76.40	79.50	**73.69**
Lexical Sim.	55.72	68.42	75.88	55.17	75.14	57.33	67.20	70.64	**65.69**
Family Avg.	67.60	66.16	84.45	63.53	75.18	59.10	72.16	73.34	**70.19**
Perplexity	79.08	75.77	91.04	61.56	75.59	64.34	78.07	82.46	**75.99**
Self-eval.	67.83	67.57	87.82	55.11	82.22	60.01	76.68	75.16	**71.55**
LN-Entropy	57.61	71.24	73.87	50.83	76.16	60.21	67.25	71.38	**66.07**
SAR	59.46	71.04	73.13	50.39	73.28	59.23	66.27	72.54	**65.67**
Semantic Ent.	55.03	68.51	70.85	55.13	61.53	61.23	59.33	68.35	**62.50**
Family Avg.	63.80	70.83	79.34	54.60	73.76	61.00	69.52	73.98	**68.35**
EigenScore	54.78	50.87	72.95	53.28	73.41	55.67	64.85	70.32	**62.02**
CCS	59.31	50.26	58.95	53.23	50.58	62.34	57.17	52.24	**55.51**
HaloScope	67.98	62.30	56.30	59.52	62.17	61.12	60.27	54.91	**60.57**
SAPLMA	81.27	72.12	91.72	76.89	88.67	71.72	80.27	85.61	**81.03**
MIND-Sup	75.33	77.58	86.87	77.30	80.70	63.21	73.19	85.95	**77.52**
SEP-Sup	81.95	69.31	89.98	76.84	88.25	76.32	79.05	86.26	**81.00**
ICR Probe	68.16	65.24	62.90	59.88	53.61	58.27	52.23	64.87	**60.65**
PRISM	77.46	73.03	91.25	75.88	90.18	79.91	80.96	87.16	**81.98**
Family Avg.	70.78	65.09	76.37	66.60	73.45	66.07	68.50	73.42	**70.04**

Method	RAG	Sum.	GSM8K	SVAMP	Math	Sci.	HEval	MBPP	Code	Agent	Multi.	Overall
Verbalized Conf.	61.23	55.91	62.90	69.90	**66.40**	63.49	60.00	57.52	**58.76**	54.46	51.09	**63.78**
SelfCheck-BERT	69.98	61.50	60.97	85.75	**73.36**	73.92	60.00	74.70	**67.35**	78.82	73.84	**71.93**
SelfCheck-NLI	71.25	64.37	83.26	78.67	**80.97**	52.55	53.04	57.83	**55.44**	66.49	66.07	**69.59**
Lexical Sim.	59.91	56.79	72.32	77.04	**74.68**	63.80	62.61	63.15	**62.88**	62.59	65.77	**65.26**
Family Avg.	65.59	59.64	69.86	77.84	**73.85**	63.44	58.91	63.30	**61.11**	65.59	64.19	**67.64**
Perplexity	70.76	64.35	77.22	79.46	**78.34**	64.48	67.83	59.73	**63.78**	83.24	69.21	**73.19**
Self-eval.	72.95	63.62	77.09	82.58	**79.84**	65.92	74.57	62.77	**68.67**	62.86	50.46	**69.72**
LN-Entropy	68.45	54.57	76.06	75.92	**75.99**	73.35	73.48	68.54	**71.01**	67.41	70.43	**68.04**
SAR	67.33	53.12	74.77	77.58	**76.18**	73.84	67.83	68.92	**68.38**	68.32	71.95	**67.59**
Semantic Ent.	51.26	52.13	51.51	50.29	**50.90**	61.30	73.91	59.35	**66.63**	60.23	64.78	**60.28**
Family Avg.	66.15	57.56	71.33	73.17	**72.25**	67.78	71.52	63.86	**67.69**	68.41	65.37	**67.76**
EigenScore	56.45	50.76	66.23	78.04	**72.14**	51.25	69.57	74.70	**72.14**	62.19	53.00	**62.25**
CCS	54.43	54.08	60.27	77.67	**68.97**	74.77	54.35	52.81	**53.58**	52.63	50.78	**57.40**
HaloScope	61.28	51.67	60.74	67.15	**63.95**	58.90	74.35	55.70	**65.03**	62.05	67.17	**61.39**
SAPLMA	77.98	66.60	83.01	83.62	**83.32**	69.87	81.30	61.17	**71.24**	93.15	68.22	**78.42**
MIND-Sup	65.23	61.79	78.86	89.29	**84.08**	79.23	63.04	64.86	**63.95**	92.18	72.68	**75.72**
SEP-Sup	74.46	64.80	78.62	86.46	**82.54**	76.24	74.35	62.46	**68.41**	87.82	69.76	**77.82**
ICR Probe	57.34	58.14	54.11	59.33	**56.72**	73.90	65.22	65.43	**65.33**	61.36	58.57	**61.09**
PRISM	80.21	67.65	77.24	84.88	**81.06**	80.51	66.96	68.54	**67.75**	87.47	61.38	**78.27**
Family Avg.	65.92	59.44	69.89	78.31	**74.10**	70.58	68.64	63.21	**65.93**	74.86	62.70	**69.05**

Column key. QA: ARC-Challenge, CommonsenseQA, TriviaQA, TruthfulQA, SQuAD v2, HotpotQA, CoQA, and HaluEval-QA. Other tasks: RAGTruth, XSum, GSM8K, SVAMP, TheoremQA (Sci.), HumanEval, MBPP, xLAM-Agent, and Belebele. Math, Code, and QA Avg. are scenario averages; Overall is the original overall average. Row colors identify black-, gray-, and white-box methods; gray rows report family averages.

Table 13: Per-dataset AUROC results (%) for Qwen3-14B. Higher values are better. Each panel preserves the original dataset and family averages.

Method	ARC	CSQA	Trivia	TruthQA	SQuAD	Hotpot	CoQA	HaluQA	QA Avg.
Verbalized Conf.	57.14	59.78	63.72	65.23	61.78	74.00	58.63	75.26	**64.44**
SelfCheck-BERT	58.86	51.65	68.54	66.04	58.17	50.52	53.37	74.84	**60.25**
SelfCheck-NLI	50.66	58.23	82.57	67.08	62.79	56.91	59.41	65.60	**62.91**
Lexical Sim.	50.27	52.89	76.02	64.90	64.05	50.17	56.37	72.14	**60.85**
Family Avg.	54.23	55.64	72.71	65.81	61.70	57.90	56.95	71.96	**62.11**
Perplexity	66.60	67.65	77.96	68.86	65.18	52.84	60.13	78.38	**67.20**
Self-eval.	72.71	67.60	86.13	59.75	79.35	71.66	75.38	78.72	**73.91**
LN-Entropy	52.65	54.53	76.59	65.31	64.37	54.53	59.38	72.83	**62.52**
SAR	54.52	53.59	74.84	62.48	63.44	55.91	56.22	75.18	**62.02**
Semantic Ent.	50.27	52.71	71.36	60.16	61.60	51.36	55.11	66.83	**58.68**
Family Avg.	59.35	59.22	77.38	63.31	66.79	57.26	61.24	74.39	**64.87**
EigenScore	54.77	50.47	74.27	60.50	63.36	53.42	57.50	64.42	**59.84**
CCS	60.72	54.91	66.87	66.71	55.06	63.06	57.18	69.09	**61.70**
HaloScope	55.19	64.94	54.01	56.08	50.12	53.03	58.74	61.03	**56.64**
SAPLMA	69.43	65.70	86.55	75.09	84.22	70.28	73.01	67.52	**73.98**
MIND-Sup	75.73	68.41	88.56	75.27	84.95	74.12	79.01	80.44	**78.31**
SEP-Sup	54.71	66.84	86.31	78.98	85.75	67.59	72.29	70.71	**72.90**
ICR Probe	51.76	67.31	65.03	67.41	54.88	63.68	57.64	54.79	**60.31**
PRISM	58.76	69.55	87.55	75.87	85.38	55.26	58.67	76.71	**70.97**
Family Avg.	60.13	63.52	76.14	69.49	70.47	62.56	64.26	68.09	**66.83**

Method	RAG	Sum.	GSM8K	SVAMP	Math	Sci.	HEval	MBPP	Code	Agent	Multi.	Overall
Verbalized Conf.	67.82	54.23	60.19	57.58	**58.88**	51.28	70.00	51.03	**60.52**	58.94	64.55	**61.83**
SelfCheck-BERT	54.08	54.17	54.45	78.37	**66.41**	64.61	56.96	59.31	**58.14**	70.19	52.66	**60.40**
SelfCheck-NLI	70.04	58.89	53.71	68.44	**61.08**	50.49	52.61	72.39	**62.50**	59.80	67.92	**62.21**
Lexical Sim.	52.66	51.94	66.40	98.37	**82.39**	66.36	64.57	59.89	**62.23**	63.52	55.68	**62.72**
Family Avg.	61.15	54.81	58.69	75.69	**67.19**	58.19	61.04	60.66	**60.85**	63.11	60.20	**61.79**
Perplexity	73.76	54.13	73.72	81.48	**77.60**	55.33	73.04	62.58	**67.81**	79.90	73.73	**68.55**
Self-eval.	53.99	61.57	63.95	99.85	**81.90**	70.92	64.35	58.74	**61.55**	72.09	63.34	**70.59**
LN-Entropy	73.76	50.97	66.09	98.96	**82.53**	69.35	61.52	54.74	**58.13**	68.80	55.69	**64.71**
SAR	74.65	50.63	66.93	89.78	**78.36**	66.13	56.09	53.59	**54.84**	70.74	55.45	**63.54**
Semantic Ent.	57.89	52.01	54.05	69.04	**61.55**	62.33	55.22	58.86	**57.04**	58.49	55.69	**58.41**
Family Avg.	66.81	53.86	64.95	87.82	**76.39**	64.81	62.04	57.70	**59.87**	70.00	60.78	**65.16**
EigenScore	53.12	50.06	67.67	96.30	**81.99**	66.00	54.35	53.10	**53.73**	68.15	54.48	**61.29**
CCS	57.45	53.70	54.08	89.48	**71.78**	65.56	53.48	53.76	**53.62**	55.13	50.74	**60.41**
HaloScope	57.45	52.45	55.87	83.85	**69.86**	60.60	50.00	51.88	**50.94**	57.11	62.37	**57.92**
SAPLMA	76.42	64.95	60.14	94.22	**77.18**	72.30	56.09	67.08	**61.59**	87.97	74.92	**73.29**
MIND-Sup	83.69	66.07	70.88	87.85	**79.37**	80.64	50.00	58.01	**54.01**	87.59	75.20	**75.67**
SEP-Sup	53.90	65.09	57.19	59.41	**58.30**	63.81	74.78	63.97	**69.38**	83.13	71.30	**69.16**
ICR Probe	60.12	51.96	65.19	76.59	**70.89**	55.16	52.17	53.79	**52.98**	50.36	60.83	**59.33**
PRISM	68.97	66.50	53.82	89.93	**71.88**	72.09	57.39	55.31	**56.35**	82.62	57.37	**68.93**
Family Avg.	63.89	58.85	60.61	84.70	**72.66**	67.02	56.03	57.11	**56.57**	71.51	63.40	**65.75**

Column key. QA: ARC-Challenge, CommonsenseQA, TriviaQA, TruthfulQA, SQuAD v2, HotpotQA, CoQA, and HaluEval-QA. Other tasks: RAGTruth, XSum, GSM8K, SVAMP, TheoremQA (Sci.), HumanEval, MBPP, xLAM-Agent, and Belebele. Math, Code, and QA Avg. are scenario averages; Overall is the original overall average. Row colors identify black-, gray-, and white-box methods; gray rows report family averages.

Table 14: Selected AUROC results (%) on Llama-3.3-70B-Instruct. We report selected experiments on seven datasets. Higher values are better.

Llama-3.3-70B-Instruct							
Method	**HumanEval**	**SQuAD**	**ARC-C**	**TruthfulQA**	**MBPP**	**HaluEval-QA**	**TriviaQA**
Black-box detectors: text-output-based							
Verbalized Conf.	68.08	70.52	68.28	73.18	69.00	65.36	80.41
SelfCheck-BERT	52.31	62.13	71.12	66.99	64.43	77.94	75.17
SelfCheck-NLI	61.15	57.10	66.22	62.83	67.79	64.38	72.00
Lexical Sim.	56.92	65.28	57.26	65.34	55.21	76.07	74.28
Family Avg.	59.61	63.76	65.72	67.09	64.11	70.94	75.46
Gray-box detectors: likelihood-based uncertainty							
Perplexity	61.54	68.16	81.09	65.84	66.86	80.29	78.77
Self-eval.	65.58	72.59	75.43	64.57	78.21	75.77	85.78
LN-Entropy	50.77	63.72	57.11	62.23	65.71	76.77	76.03
SAR	52.69	61.21	54.68	59.77	68.43	77.04	75.47
Semantic Ent.	60.19	58.87	57.25	56.72	58.32	72.69	70.17
Family Avg.	58.15	64.91	65.11	61.83	67.51	76.51	77.24
White-box detectors: internal-state-based							
EigenScore	50.77	62.92	59.16	58.98	58.64	78.95	72.43
CCS	65.38	53.22	63.10	52.29	56.64	72.90	56.05
HaloScope	50.77	62.43	70.43	54.38	63.07	66.84	57.02
SAPLMA	61.15	82.31	68.20	81.13	69.79	80.12	87.08
MIND-Sup	68.85	85.23	77.39	83.41	77.29	78.89	86.46
SEP-Sup	76.54	81.12	75.20	77.60	78.29	71.08	85.03
ICR Probe	56.67	50.97	50.54	58.08	64.43	73.82	52.12
PRISM	67.69	85.75	81.64	71.12	77.43	83.48	86.48
Family Avg.	62.23	70.49	68.21	67.12	68.20	75.76	72.83

G Aggregation and Ranking Sensitivity

Data and scope. This analysis uses only the reported two-decimal per-dataset AUROC values for the 17 implementations on four backbones in Appendix F. It does not use per-example scores, rerun a detector, or include the partial 70B evaluation. Small rounding differences from computations using unrounded metrics may occur.

Aggregation definitions. For a fixed backbone, let A_{md} be the reported AUROC of implementation m on dataset d, and let D_s contain the datasets in scenario s. We compute

$$A_m^{\text{dataset}} = \frac{1}{17}\sum_{d=1}^{17} A_{md}, \qquad A_m^{\text{scenario}} = \frac{1}{8}\sum_{s=1}^{8}\frac{1}{|D_s|}\sum_{d\in D_s} A_{md}. \tag{3}$$

The eight groups, in table order, contain $8, 1, 1, 2, 1, 2, 1, 1$ datasets: QA, RAG, summarization, mathematics, science, code, agent, and multilingual evaluation. Within each group datasets receive equal weight. Neither scheme pools predictions across datasets or weights by test sample count. They encode alternative benchmark task priorities rather than estimates of a known deployment distribution.

Ranks and comparisons. We rank the 17 implementations by unrounded recomputed means, using average ranks for ties, and compute Spearman correlations. Table 15 reports both means and ranks for every implementation; Table 16 reports all six backbone pairs under both schemes. Within-backbone correlations between schemes are 0.985, 0.985, 0.953, and 0.963 in the displayed model order. Individual ranks move by at most three positions, and the top implementation changes on one backbone. These correlations describe the reported table; they are not significance tests or measurements of transfer by a fixed fitted detector.

Interpreting the changed winner. On Qwen3-8B, SAPLMA's QA mean exceeds MIND-Sup's by 5.49 AUROC points, computed from the eight constituent datasets. MIND-Sup is higher in five of the eight scenario means, while SAPLMA is higher in the other three. Under scenario weighting, QA

receives less weight and all scenario weights change together; this reaggregation reverses their order. It does not isolate a QA-only intervention or establish a statistically significant performance gap.

Access-group composition. Access-group means average the included implementations, not independent method families; SelfCheck-BERT and SelfCheck-NLI each contribute one entry. For Llama-3.2-3B, black-/gray-/white-box means change from 66.07/66.47/65.91 under dataset weighting to 64.03/64.17/65.17 under scenario weighting. These summaries are affected by the benchmark's task mix, detector roster, supervision, and adaptation choices. They cannot identify the causal effect of an access regime.

Reproduction and limits. The source package includes `analysis/reported_auroc.json`, a transcription of the reported dataset cells, and `analysis/reproduce_aggregation.py`, a standard-library script that produces the complete derived metrics and the following tables. No raw experiment archive is required for this descriptive analysis. Without paired per-example scores or repeated end-to-end runs, we do not attach confidence intervals or significance claims to the aggregate differences or rank correlations. The per-detector bootstrap intervals in Appendix L do not test these aggregate rankings.

Table 15: Aggregation sensitivity for all 17 evaluated implementations. D: equal dataset weighting; S: equal scenario weighting. Each entry reports AUROC (%) with its rank in parentheses. Values are recomputed from the rounded per-dataset results; ranks are descriptive, not significance claims.

Method	Qwen3-8B		Llama-3.2-3B		Llama-3.1-8B		Qwen3-14B	
	D	S	D	S	D	S	D	S
Verbalized Conf.	59.52 (13)	58.48 (13)	57.99 (16)	54.48 (17)	63.78 (12)	59.98 (15)	61.83 (11)	60.08 (12)
SelfCheck-BERT	59.33 (14)	58.03 (14)	69.25 (4)	67.90 (4)	71.93 (6)	71.46 (5)	60.40 (14)	60.06 (13)
SelfCheck-NLI	62.48 (8)	62.08 (8)	67.86 (8)	66.52 (7)	69.59 (8)	66.35 (10)	62.21 (10)	61.70 (10)
Lexical Sim.	60.91 (10)	61.15 (9)	69.16 (5)	67.24 (5)	65.26 (11)	64.01 (11)	62.72 (9)	61.95 (9)
Perplexity	64.84 (5)	64.38 (6)	66.35 (10)	63.54 (11)	73.19 (5)	71.27 (6)	68.55 (6)	68.68 (3)
Self-eval.	71.77 (3)	67.54 (3)	67.19 (9)	63.72 (10)	69.72 (7)	66.98 (9)	70.59 (3)	67.41 (5)
LN-Entropy	64.80 (6)	65.70 (5)	68.41 (6)	66.78 (6)	68.04 (9)	68.41 (7)	64.71 (7)	65.22 (7)
SAR	62.88 (7)	63.78 (7)	68.32 (7)	66.30 (8)	67.59 (10)	68.10 (8)	63.54 (8)	64.10 (8)
Semantic Ent.	58.75 (15)	57.92 (15)	62.06 (13)	60.50 (14)	60.28 (16)	58.72 (16)	58.41 (16)	57.96 (16)
EigenScore	60.87 (11)	60.92 (10)	65.07 (11)	63.86 (9)	62.25 (13)	59.99 (14)	61.29 (12)	60.92 (11)
CCS	60.25 (12)	60.02 (12)	56.67 (17)	57.80 (16)	57.40 (17)	58.09 (17)	60.41 (13)	58.71 (14)
HaloScope	58.54 (17)	57.36 (16)	61.39 (14)	61.01 (13)	61.39 (14)	61.33 (13)	57.92 (17)	58.43 (15)
SAPLMA	75.68 (1)	73.09 (2)	74.53 (2)	73.13 (2)	78.42 (1)	76.43 (1)	73.29 (2)	73.66 (2)
MIND-Sup	74.20 (2)	74.73 (1)	75.45 (1)	73.67 (1)	75.72 (4)	74.58 (4)	75.67 (1)	75.61 (1)
SEP-Sup	62.23 (9)	60.89 (11)	63.52 (12)	61.81 (12)	77.82 (3)	75.63 (3)	69.16 (4)	67.23 (6)
ICR Probe	58.66 (16)	57.09 (17)	58.73 (15)	59.65 (15)	61.09 (15)	61.50 (12)	59.33 (15)	57.83 (17)
PRISM	69.31 (4)	67.49 (4)	71.90 (3)	70.42 (3)	78.27 (2)	76.00 (2)	68.93 (5)	68.34 (4)

Table 16: Cross-backbone Spearman rank correlations over the 17 implementations. Both aggregation schemes are shown; no significance test or transfer experiment is implied.

Backbone 1	Backbone 2	Dataset macro	Scenario macro
Qwen3-8B	Llama-3.2-3B	0.699	0.703
Qwen3-8B	Llama-3.1-8B	0.775	0.647
Qwen3-8B	Qwen3-14B	0.946	0.931
Llama-3.2-3B	Llama-3.1-8B	0.745	0.745
Llama-3.2-3B	Qwen3-14B	0.630	0.659
Llama-3.1-8B	Qwen3-14B	0.824	0.789

H Dataset-level Differences within Scenarios

Comparison protocol. We examine all six pairs of the four fully evaluated backbones using the same reported AUROC cells as Appendix G. For each dataset, we first average over the 17 evaluated implementations and subtract the source-backbone mean from the target-backbone mean, as ordered in Table 17. Scenario changes average these dataset changes with equal weight. We focus on the three scenarios containing multiple datasets: QA (eight), mathematics (two), and code (two). RAG, summarization, science, agent, and multilingual evaluation each contain one dataset in OpenHalDet, so they cannot support a within-scenario comparison; their dataset changes are included in the supplementary computation file.

Coverage across backbones. At least one positive and one negative dataset-mean change occur within QA in all six comparisons, within mathematics in four, and within code in two. When both directions must exceed one AUROC point, the corresponding counts are six, three, and one; the counts are unchanged with a two-point threshold. These thresholds describe the magnitude of the changes and are not significance tests. The six comparisons share four backbones and are not independent replications. Moreover, QA contains more datasets than mathematics or code, so these counts do not rank the degree of heterogeneity across scenarios.

Qwen3-8B versus Qwen3-14B. Table 18 reports every constituent dataset of the three multi-dataset scenarios. All 17 implementations have both increases and decreases within QA, even when each direction must exceed two AUROC points. In mathematics, 12 implementations decrease on GSM8K and increase on SVAMP, while two show the reverse pattern; in code, nine decrease on HumanEval and increase on MBPP, while two show the reverse pattern. As a weighting check, we first merge SelfCheck-BERT and SelfCheck-NLI into one method-family average and then average the 16 families equally. The signs of all 12 constituent dataset-mean changes remain unchanged; the QA, mathematics, and code changes become $+0.11$, $+6.27$, and -0.88 points, respectively. The near-zero QA average therefore does not imply uniformly stable detection, and the positive mathematics average does not imply that both mathematics datasets improve.

Interpretation and limits. These comparisons describe detection AUROC under each backbone's generated responses and evaluation setting. They do not hold generated answers, label composition, or fitted detector parameters fixed, and do not isolate model size or measure transfer by a fixed detector. They also do not measure changes in generation accuracy. Without per-example scores, we cannot test whether these differences are statistically significant; the small code differences warrant particular caution. The implementations share responses and labels, so implementation counts are descriptive rather than independent votes.

Reproduction. The standard-library script `analysis/reproduce_heterogeneity.py` regenerates the main-text table, the following appendix tables, and all dataset- and implementation-level differences from `analysis/reported_auroc.json`. The analysis uses rounded published cells only and does not rerun experiments, refit detectors, or change score orientation.

Table 17: All six backbone comparisons. Each scenario entry reports the mean AUROC change in percentage points, followed by the number of increasing/decreasing constituent dataset means. Q8/Q14: Qwen3-8B/14B; L3: Llama-3.2-3B; L8: Llama-3.1-8B. Arrows specify target minus source.

Comparison	QA (8 datasets)	Math (2 datasets)	Code (2 datasets)
Q8 → Q14	$+0.06$ (5/3)	$+5.97$ (1/1)	-0.58 (1/1)
L3 → L8	$+1.82$ (6/2)	$+5.88$ (2/0)	$+0.19$ (1/1)
Q8 → L3	$+2.68$ (7/1)	$+1.12$ (1/1)	$+5.99$ (2/0)
Q8 → L8	$+4.50$ (6/2)	$+7.00$ (2/0)	$+6.18$ (2/0)
L3 → Q14	-2.61 (1/7)	$+4.85$ (1/1)	-6.57 (0/2)
L8 → Q14	-4.43 (1/7)	-1.03 (1/1)	-6.76 (0/2)

I Cost Analysis

Profiling setup. We profile detector cost on Llama-3.2-3B-Instruct using four representative datasets: ARC-Challenge, CoQA, GSM8K, and MBPP. Each profiling run uses up to 100 examples from the corresponding dataset, with the same train/validation/test split construction as the main benchmark pipeline. All runs are executed on a single NVIDIA H800 PCIe GPU with PyTorch 2.11.0 and CUDA 13.0. The maximum generation length is set to 2048 tokens, and hardware statistics are sampled every 5 seconds when the profiled stage is long enough for periodic sampling.

Cost accounting. We separate detector cost into feature preparation, detector fitting, and detector inference. Feature preparation includes detector-specific evidence acquisition, such as stochastic generations, auxiliary self-evaluation outputs, token log-probabilities, hidden states, or QA-feature caches. Detector fitting is only applicable to methods that train detector-specific parameters. Detector inference measures the final scoring step on the test examples after the required artifacts have been

Table 18: Constituent dataset changes for Qwen3-8B to Qwen3-14B. Means average all 17 implementations. The last column counts implementations with positive/negative changes. Each numerical column is rounded independently.

Scenario	Dataset	Qwen3-8B	Qwen3-14B	Δ	Increase/decrease
QA	ARC-Challenge	62.78	58.51	−4.26	3/14
	CommonsenseQA	62.13	60.40	−1.73	4/13
	TriviaQA	73.46	75.70	+2.24	11/6
	TruthfulQA	65.42	66.81	+1.39	11/6
	SQuAD-V2	64.47	67.32	+2.85	13/4
	HotpotQA	65.44	59.90	−5.54	4/13
	CoQA	60.71	61.65	+0.94	9/8
	HaluEval-QA	66.23	70.85	+4.63	11/6
Math	GSM8K	66.03	61.43	−4.60	4/13
	SVAMP	66.95	83.50	+16.55	14/3
Code	HumanEval	63.25	58.98	−4.27	5/12
	MBPP	55.02	58.12	+3.10	12/5

prepared. Because different detectors require different forms of evidence, we report both wall-clock time and evidence-acquisition statistics.

Reported quantities. Table 19 reports average runtime statistics over the four profiled datasets. *AUROC* is the corresponding average detection performance, included to contextualize the cost measurements. *Cost@100* is the artifact-inclusive wall-clock time for processing up to 100 examples under the profiling setup. *Score-only* is the detector-side time recorded in the isolated detector run, including detector-side feature use, fitting when applicable, and scoring. *Infer.* reports the final detector scoring latency per example after the required artifacts are available. *Extra gen. calls* and *Extra gen. tokens* count only additional model generations beyond the primary response generation. Therefore, a value of 0.0 in these two columns does not mean zero computation; it means that the detector does not require additional generated responses beyond the original model output, although it may still use cached logits, hidden states, feature extraction, fitting, or detector-side scoring.

Observations. The profiling results show that detector cost varies with the type of evidence required. Methods that reuse single-response likelihoods or cached internal features, such as Perplexity, SAPLMA, PRISM, and HaloScope, do not require extra generated responses and therefore have 0.0 extra generation calls and tokens. Prompt-based auxiliary methods require additional model calls: Verbalize uses one short auxiliary response per example, while Self-Evaluator uses one longer auxiliary response. Sample-based methods require five additional generations per example, which is reflected in the extra-generation columns. Among these methods, the final scoring overhead also differs: likelihood- or lexical-similarity-based scoring is relatively light, whereas NLI-, BERTScore-, and semantic-clustering-based scoring adds additional per-example computation.

Interpretation. These results should be interpreted as controlled profiling measurements, not as hardware-independent runtime estimates. Absolute costs depend on the GPU, software stack, sequence lengths, batching behavior, and whether shared artifacts are recomputed or reused. The main practical implication is that detector comparisons should report not only accuracy, but also the evidence required to obtain the detector score.

J Assumptions and Limitations

OpenHalDet is designed as a unified evaluation benchmark for hallucination detection across heterogeneous generation scenarios. We summarize below the main assumptions behind the benchmark protocol and the scope of its limitations.

J.1 Assumptions

Reference-grounded annotation. OpenHalDet assumes that hallucination labels can be assigned by comparing a generated response against the available references, acceptable answers, task context,

Table 19: Average detector runtime cost on Llama-3.2-3B-Instruct across ARC-Challenge, CoQA, GSM8K, and MBPP. Cost@100 is the artifact-inclusive wall-clock time for up to 100 examples under the profiling setup. Extra generation calls and tokens are counted per example beyond the primary response generation.

Detector	AUROC	Cost@100 (s)	Score-only (s)	Infer. ms/ex.	Extra gen. calls/ex.	Extra gen. tokens/ex.
Perplexity	77.3	4.5	4.5	0.06	0.0	0.0
Verbalize	57.7	10.9	10.9	0.05	1.0	3.2
SAPLMA	58.3	4.6	4.6	0.76	0.0	0.0
PRISM	68.2	4.8	4.8	0.93	0.0	0.0
HaloScope	68.5	4.8	4.8	1.93	0.0	0.0
SAR	84.8	5.1	5.1	55.89	5.0	224.3
ICR Probe	53.2	13.0	13.0	1.00	0.0	0.0
CCS	63.0	18.7	18.7	1.63	0.0	0.0
MIND-Sup	62.9	84.3	0.5	4.38	0.0	0.0
Self-Evaluator	74.8	173.9	173.9	0.20	1.0	68.3
EigenScore	73.1	433.7	433.7	7.54	5.0	224.3
LN-Entropy	81.5	435.3	435.3	0.03	5.0	224.3
Lexical Similarity	76.8	435.3	435.3	5.62	5.0	224.3
SelfCheck-NLI	77.0	441.1	441.1	50.81	5.0	224.3
Semantic Entropy	64.4	443.8	443.8	74.02	5.0	224.3
SelfCheck-BERTScore	79.2	444.0	444.0	103.96	5.0	224.3
SEP-SE	73.4	452.9	452.9	0.87	5.0	224.3

Note. Extra generation calls and tokens are counted beyond the primary response generation. A value of 0.0 in these columns does not indicate zero computation; these methods may still require cached logits, hidden states, feature extraction, detector fitting, or detector-side scoring. Score-only time is the detector-side time recorded in the isolated detector run; Cost@100 additionally accounts for the required cached artifacts under the profiling setup. The supplied profiling entry SEP-SE includes five additional generations, whereas the documented SEP-Sup scoring path uses a supervised correctness probe.

and known incorrect answers when provided. This enables a consistent annotation protocol across heterogeneous datasets, but the resulting labels necessarily depend on the coverage and quality of the available references. Responses that are correct but not captured by the provided references may be difficult to label automatically, and ambiguous or under-specified questions may introduce additional uncertainty. To reduce ambiguity, we use a structured LLM-judge annotation protocol and exclude abstention cases from the main binary evaluation. The resulting labels should therefore be interpreted as scalable reference-grounded labels rather than human-certified factuality judgments.

Unified detector scoring. OpenHalDet evaluates each method as a scalar hallucination-risk detector, where larger scores indicate a higher likelihood of hallucination. This common interface is necessary for applying the same AUROC, AUPR, and FPR@95TPR metrics across black-box, gray-box, and white-box methods. For each detector, dataset, and backbone, score orientation is determined using the validation split and held fixed when computing test metrics.

Shared evaluation protocol. The benchmark assumes that detector comparisons should be made under the same response generation, annotation, split construction, and metric implementation. Accordingly, detectors evaluated on the same dataset and backbone operate on the same generated responses and labels, and methods requiring additional evidence use cached artifacts produced by the benchmark pipeline. This design isolates detector behavior from protocol-level differences as much as possible.

J.2 Limitations and Scope

Benchmark-compatible implementations. Several prior detectors were originally developed under task-specific datasets, prompting formats, access assumptions, or computational budgets. We use benchmark-compatible adaptations where required by the unified pipeline. These changes can affect the training objective, supervision, or feature construction, and are disclosed explicitly rather than assumed to preserve the complete original method. For example, supervised probes are fitted on the

OpenHalDet training split, methods requiring stochastic samples use the shared stochastic-generation artifacts, and methods requiring internal states use the shared hidden-state or QA-feature caches. For computationally heavier methods, we use scalable variants that are compatible with broad multi-dataset evaluation; for instance, our SAR implementation uses cached stochastic generations and sequence-level likelihood/similarity signals rather than requiring task-specific token-level processing. In particular, MIND-Sup replaces Wikipedia-derived pseudo-label training with benchmark-label supervision, SEP-Sup predicts correctness instead of semantic entropy, and the HaloScope adaptation selects its PCA and pseudo-label settings using training labels. Results compare these implementations under shared generated data; differences in supervision remain an additional comparison axis. They do not establish the performance of unchanged original methods.

Model and task coverage. OpenHalDet covers a broad set of task scenarios and evaluates detectors on recent open-weight LLMs from the Llama and Qwen families. However, the benchmark does not exhaust the space of possible deployment settings. Closed-source API models, domain-specialized models, multimodal generation, very long-context generation, and fully interactive multi-turn agent environments may exhibit different hallucination patterns and detector behavior. Similarly, although OpenHalDet includes multilingual and agentic scenarios, its coverage remains limited relative to the full diversity of languages, tools, and real-world user intents.

Cost profiling scope. Our cost analysis is intended to compare relative cost patterns under a controlled hardware and software environment. Absolute runtime, memory, and energy values can vary with GPU type, system load, implementation details, sequence length, and whether shared artifacts are recomputed or reused. For this reason, we report artifact-inclusive cost together with explicit evidence-acquisition statistics such as extra model calls and generated tokens. The reported cost results should be interpreted as controlled profiling evidence rather than hardware-independent constants.

Statistical uncertainty. We report bootstrap confidence intervals for representative AUROC comparisons to estimate finite-sample uncertainty over test examples. This procedure does not rerun response generation, annotation, detector training, or stochastic sampling. A more exhaustive uncertainty analysis would require repeated end-to-end runs, which is substantially more expensive for detectors requiring stochastic generations or hidden-state extraction.

Intended use. OpenHalDet is intended for controlled comparison of hallucination detectors under shared generation, annotation, scoring, and evaluation protocols. It does not certify detectors for direct deployment in high-stakes applications, where additional task-specific validation, calibration, human review, and monitoring would be required.

K ARTIFACT

Repository access details are omitted for double-blind review. The repository contains the core OpenHalDet codebase, including dataset adapters, prompt construction utilities, response-generation scripts, annotation scripts, detector implementations, and evaluation code.

Repository contents. The artifact is intended to support inspection and reproduction of the benchmark pipeline. It includes code for converting datasets into the unified schema, generating model responses, running the LLM-judge annotation, preparing detector-specific features, and computing evaluation metrics. The repository also includes basic usage instructions and configuration examples.

Data access. OpenHalDet builds on existing public datasets and benchmark sources. The repository provides adapters and processing scripts rather than claiming ownership of the original datasets. Users should follow the licenses and access terms of the corresponding source datasets.

Compute requirements. Running the full benchmark can require substantial GPU resources, especially for detectors that use stochastic generations or hidden-state extraction. The code therefore supports running selected datasets, backbones, or detectors independently, and reusing cached intermediate artifacts when available.

L STATISTICAL SIGNIFICANCE

To provide a lightweight estimate of finite-sample uncertainty, we report 95% stratified bootstrap confidence intervals for representative AUROC results on Llama-3.2-3B-Instruct. For each selected detector–dataset setting, we retain the score orientation fixed on the validation split, then resample positive and negative test examples with replacement for 1,000 bootstrap trials. The direction remains unchanged across all trials; it is not reselected on the test set or within individual resamples. This procedure does not rerun response generation, annotation, detector fitting, or stochastic sampling; it estimates uncertainty due to the finite test set.

Table 20 reports representative detectors from black-box, gray-box, and white-box access regimes across seven datasets. The intervals should be interpreted as finite-sample uncertainty estimates rather than exhaustive multi-seed variation or formal pairwise significance tests.

Table 20: Representative 95% stratified bootstrap confidence intervals for AUROC on Llama-3.2-3B-Instruct. Each cell reports AUROC followed by the 95% confidence interval in brackets.

Detector	TriviaQA	TruthfulQA	HotpotQA	HaluEval-QA
Verbalized Conf.	74.27 [72.26, 76.34]	59.49 [51.21, 66.89]	63.64 [59.88, 67.26]	68.13 [65.31, 70.86]
SelfCheck-BERTScore	82.67 [80.95, 84.49]	59.54 [51.02, 68.11]	70.58 [66.65, 74.26]	80.03 [77.42, 82.45]
SelfCheck-NLI	84.52 [82.91, 86.27]	66.26 [57.46, 75.60]	70.68 [66.29, 74.69]	71.30 [67.41, 75.10]
Lexical Sim.	82.50 [80.82, 84.33]	61.79 [53.20, 70.41]	71.96 [67.97, 75.92]	78.92 [76.07, 81.75]
Perplexity	83.10 [81.34, 84.89]	55.75 [46.36, 64.41]	64.04 [59.96, 67.97]	79.45 [77.06, 81.96]
Self-Evaluation	76.17 [74.07, 78.23]	59.24 [50.29, 68.01]	74.90 [71.06, 78.46]	74.46 [71.32, 77.77]
LN-Entropy	81.35 [79.60, 83.09]	50.88 [41.78, 59.59]	70.81 [66.48, 74.64]	79.13 [76.45, 81.90]
SAPLMA	82.69 [80.94, 84.47]	75.15 [67.60, 82.01]	77.51 [73.89, 80.92]	81.85 [79.13, 84.42]
MIND-Sup	85.99 [84.55, 87.56]	75.31 [67.46, 82.30]	76.56 [72.66, 80.06]	86.57 [83.96, 88.99]
PRISM	83.07 [81.39, 84.79]	73.95 [66.14, 81.29]	69.97 [65.79, 74.04]	78.35 [75.48, 81.30]

Detector	XSum	SVAMP	HumanEval
Verbalized Conf.	51.05 [46.20, 56.11]	59.23 [51.79, 66.68]	56.52 [50.00, 63.04]
SelfCheck-BERTScore	61.27 [53.25, 69.32]	72.69 [62.44, 81.63]	66.96 [46.52, 86.96]
SelfCheck-NLI	69.98 [62.73, 76.80]	70.70 [58.89, 80.68]	52.61 [30.87, 76.09]
Lexical Sim.	60.74 [52.94, 68.25]	65.96 [53.63, 76.53]	72.61 [53.91, 88.70]
Perplexity	57.91 [49.82, 66.00]	64.76 [52.20, 76.88]	52.61 [29.55, 77.39]
Self-Evaluation	59.34 [50.97, 67.04]	79.74 [69.64, 88.89]	86.74 [72.61, 96.96]
LN-Entropy	55.22 [47.24, 63.38]	69.83 [57.07, 81.09]	69.13 [51.73, 85.65]
SAPLMA	56.57 [48.41, 64.97]	71.49 [59.92, 82.37]	77.83 [56.08, 94.36]
MIND-Sup	59.41 [51.66, 66.97]	79.72 [69.35, 88.58]	67.83 [44.77, 86.52]
PRISM	62.41 [54.95, 70.07]	73.02 [61.57, 82.88]	63.91 [40.00, 83.92]

Note. The number of evaluated examples is dataset-dependent: TriviaQA $N = 1964$, TruthfulQA $N = 154$, HotpotQA $N = 996$, HaluEval-QA $N = 1966$, XSum $N = 200$, SVAMP $N = 140$, and HumanEval $N = 33$. The wider intervals on smaller datasets, especially HumanEval, reflect larger finite-sample uncertainty.

M BROADER IMPACTS

OpenHalDet aims to support more transparent, reproducible, and comparable evaluation of hallucination detectors across diverse generation scenarios. By standardizing response generation, annotation, detector scoring, and metric computation, the benchmark provides a common protocol for analyzing the reliability and cost trade-offs of different detection methods.

At the same time, benchmark results should not be interpreted as certifying the factual reliability or deployment safety of LLM systems. Detector performance can depend on the covered datasets, backbone models, prompting protocols, automatic annotation quality, and evaluation metrics. Misuse could arise if benchmark scores are treated as standalone guarantees for high-stakes applications, or if automatic labels are assumed to be error-free. We therefore document the benchmark scope, annotation protocol, limitations, and intended use, and encourage users to complement OpenHalDet with task-specific validation and human review in safety-critical settings.

N LICENSES

OpenHalDet builds on existing open models and public benchmark datasets. We do not claim ownership of these third-party assets. All models and datasets should be used under the licenses and terms specified by their original providers. Table 21 summarizes the licenses or usage terms associated with the main model and dataset assets used in our benchmark.

Table 21: Provider-stated licenses and terms for all 17 datasets and five generation backbones. Repository licenses do not automatically cover third-party source documents; linked providers supply the authoritative terms.

Asset	Type	License / Terms	Notes
Llama-3.1-8B-Instruct	Model	Llama 3.1 Community License	Meta license and associated use terms.
Llama-3.2-3B-Instruct	Model	Llama 3.2 Community License	Meta license and associated use terms.
Llama-3.3-70B-Instruct	Model	Llama 3.3 Community License	Meta license and associated use terms.
Qwen3-8B	Model	Apache 2.0	Provider model card.
Qwen3-14B	Model	Apache 2.0	Provider model card.
ARC-Challenge	Dataset	CC BY-SA 4.0	Provider dataset-card metadata.
CommonsenseQA	Dataset	MIT	Provider dataset card.
TriviaQA	Dataset	Source-dependent	Repository: Apache 2.0; source questions/documents have separate rights.
TruthfulQA	Dataset	Apache 2.0	Original release repository.
SQuAD v2	Dataset	CC BY-SA 4.0	Official dataset page.
HotpotQA	Dataset	CC BY-SA 4.0	Official dataset page.
CoQA	Dataset	Domain-dependent	CC BY-SA 4.0, MSR-LA, RACE and Apache source terms.
HaluEval-QA	Dataset	MIT (repository)	Repository license; retain upstream content terms.
RAGTruth	Dataset	MIT (repository)	Repository license; retain upstream content terms.
XSum	Dataset	Data: unspecified	Dataset card: unknown; code repository: MIT.
GSM8K	Dataset	MIT	Original release license.
SVAMP	Dataset	MIT	Original release repository.
TheoremQA	Dataset	MIT	Provider dataset card.
HumanEval	Dataset	MIT	Original release license.
MBPP	Dataset	CC BY 4.0	Provider dataset card.
xLAM-Agent	Dataset	CC BY 4.0; see terms	Card also states research purpose and access conditions.
Belebele	Dataset	CC BY-SA 4.0	Evaluation dataset; separate training-set terms.

The OpenHalDet repository contains dataset adapters and processing scripts rather than claiming ownership of the original datasets. When reconstructing the benchmark, users should download or access each dataset from its original source and comply with the corresponding license or usage terms. The code release and the underlying third-party datasets have separate licensing scopes. The release README declares MIT for the benchmark code; the full license text and attribution must accompany the code release.

O ASSETS

Released assets. The anonymized OpenHalDet release includes the benchmark codebase, dataset adapters, unified data schema, prompt and annotation templates, detector wrappers, evaluation scripts, and documentation.

Documentation. The release includes a README and accompanying documentation describing installation, data preparation, schema fields, annotation protocol, detector evaluation, licenses, intended use, limitations, and reproduction commands.

P EXTENDED DETECTOR OVERVIEW

This appendix retains the detailed detector-family discussion accompanying Section 3.

P.1 BLACK-BOX DETECTORS

Black-box detectors perform hallucination detection using only text outputs, without access to token probabilities, sequence likelihoods, hidden states, or attention maps. They therefore suit closed-source or API-only cases, where users can query the model but cannot inspect its internal signals.

We categorize them by the type of textual signal used: *verbalized-confidence methods*, which rely on a self-reported confidence statement from a single output, and *sample-consistency methods*, which estimate hallucination risk from consistency across multiple sampled responses. 1) Verbalized-confidence methods are a small but practical family of black-box detectors that use the model's self-reported confidence as a hallucination signal. Such methods are lightweight and easy to deploy, but their reliability depends on whether the model's stated confidence is calibrated with factual correctness. We include VERBALIZED CONFIDENCE (Lin et al., 2022b) as a representative baseline for this category. 2) Sample-consistency methods estimate hallucination risk by generating multiple responses to the same input and measuring disagreement among sampled outputs. SELFCHECKGPT (Manakul et al., 2023) checks whether independently sampled responses support or contradict the main response, whereas LEXICAL SIMILARITY (Lin et al., 2024) captures surface-level agreement across sampled outputs.

P.2 GRAY-BOX DETECTORS

Gray-box detectors rely on probability-based signals exposed during generation, including token probabilities, sequence likelihoods, and related generation scores. They require more model access than black-box detectors, but do not use hidden states, attention maps, or other internal representations. In OpenHalDet, gray-box detectors are represented by likelihood-based uncertainty methods.

These methods estimate hallucination risk from probability-based signals associated with the model output, including token probabilities, sequence likelihoods, and uncertainty over sampled generations. We organize them into two groups: *single-output likelihood methods* and *sampling-based uncertainty methods*. 1) For single-output likelihood methods, PERPLEXITY (Ren et al., 2023) scores a generated output using token-level likelihoods, while SELF-EVALUATION (Kadavath et al., 2022) asks the model to assess whether its own output is true or false; following our score orientation, we use the probability of the "False" label, equivalently one minus the probability of correctness, as the hallucination score. 2) For sampling-based uncertainty methods, LN-ENTROPY (Malinin & Gales, 2021) measures length-normalized sequence-level uncertainty from sampled outputs and their likelihoods. SAR (Duan et al., 2024) further reweights this uncertainty by semantic relevance among samples; for scalability, we use its sentence-level variant. SEMANTIC ENTROPY (Farquhar et al., 2024) clusters sampled outputs into semantic equivalence classes and computes uncertainty over the resulting semantic distribution. All methods in this group are training-free, but require access to probability or likelihood information from the target model.

P.3 WHITE-BOX DETECTORS

White-box detectors require access to internal model signals, such as hidden states, attention maps, layerwise activations, or derived internal features. In contrast to black-box and gray-box methods, they directly exploit internal representations rather than relying only on observable text or probability-based generation signals. Such information may provide richer evidence for hallucination detection, but imposes the strongest access assumptions. As a result, white-box detectors are typically more applicable to open-source or fully accessible models than to closed-source API-only systems.

These methods use internal representations extracted from the target LLM to detect hallucination. We organize them according to how internal signals are used: *representation-consistency scores*, *contrastive or subspace-based objectives*, *hidden-state probes*, and *prompt- or dynamics-guided internal features*. For *representation-consistency* methods, EIGENSCORE (Chen et al., 2024) measures representation-level consistency across stochastic generations by analyzing the spectral geometry of hidden-state representations. For *contrastive or subspace-based* methods, CCS (Burns et al., 2023) learns a contrast-consistent direction from paired hidden states, while HALOSCOPE (Du et al., 2024) estimates hallucination-related membership in a latent representation subspace and trains a truthfulness classifier from the estimated memberships. For *hidden-state probe* methods, SAPLMA (Azaria & Mitchell, 2023) trains a supervised classifier on input-output hidden states to

predict statement truthfulness. The original MIND (Su et al., 2024) uses generation-time hidden states with Wikipedia-derived pseudo-label training, and the original SEP (Kossen et al., 2025) learns to approximate semantic entropy. Our evaluated variants, MIND-Sup and SEP-Sup, instead use benchmark truthfulness labels; their training and features are documented in Appendix E. For *prompt- or dynamics-guided internal features*, PRISM (Zhang et al., 2025a) uses prompt-guided hidden states to improve cross-domain supervised detection, whereas ICR PROBE (Zhang et al., 2025d) constructs ICR scores from hidden-state updates and attention maps to track cross-layer residual-stream dynamics.

Q EXTENDED BENCHMARK SCOPE

This appendix provides additional detail on the scenario and backbone coverage summarized in Section 2.2 and Table 1.

Fine-Grained Coverage Across QA Settings. Question answering (QA) is widely used for hallucination detection, but it is not a homogeneous task: QA settings vary in answer format, evidence availability, and reasoning demand. OpenHalDet therefore separates QA into multiple-choice, open-ended, reading-comprehension, multi-hop, conversational, and context-grounded settings. ARC-Challenge (Clark et al., 2018) and CommonsenseQA (Talmor et al., 2019) cover multiple-choice science and commonsense reasoning; TriviaQA (Joshi et al., 2017) and TruthfulQA (Lin et al., 2022a) cover open-ended factual and truthfulness-oriented QA; SQuAD v2 (Rajpurkar et al., 2018) and HotpotQA (Yang et al., 2018) evaluate evidence-grounded answer extraction, including unanswerable and multi-hop cases. CoQA (Reddy et al., 2019) adds conversational context, while HaluEval-QA (Li et al., 2023) provides QA-style hallucination examples with human annotations. This design enables fine-grained analysis of detector reliability across QA formats, grounding conditions, and reasoning requirements.

Broad Coverage Beyond Question Answering. OpenHalDet further extends evaluation beyond QA to retrieval-augmented generation, summarization, reasoning, code generation, agentic tool use, and multilingual understanding. This scope captures settings where detection must account for evidence support, summarization faithfulness, multi-step reasoning, code correctness, tool-call validity, and cross-lingual comprehension. RAGTruth (Niu et al., 2024) and XSum (Narayan et al., 2018) cover evidence-grounded generation and abstractive summarization; GSM8K (Cobbe et al., 2021) and TheoremQA (Chen et al., 2023) cover mathematical and theorem-based reasoning; HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) cover code generation. We further include xLAM-Agent (Zhang et al., 2025b) for tool use and Belebele (Bandarkar et al., 2024) for multilingual evaluation. This design enables analysis of detector reliability across a wider range of generation contexts.

Model-Diverse Evaluation. OpenHalDet evaluates detectors across recent open-weight LLMs from the Llama and Qwen families, reflecting the need for robustness beyond a single LLM. Since many prior studies rely on earlier or single backbones, it remains unclear whether their conclusions transfer to newer models. The selected backbones are representative recent open-weight LLMs from two widely used model families. Appendix A summarizes the selected models together with representative Hugging Face download statistics, providing a practical indicator of community adoption and supporting our controlled analysis of detector stability across model families.