InterpDetect: Interpretable Signals for Detecting Hallucinations in Retrieval-Augmented Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Retrieval-Augmented Generation (RAG) reduces hallucinations by incorporating external knowledge, yet models still produce content inconsistent with retrieved evidence. Effective detection requires separating external context from parametric knowledge, which prior methods conflate. We find hallucinations often emerge when later-layer FFNs inject excessive parametric knowledge into the residual stream. To address this, we explore *external context scores* and *parametric knowledge scores*, mechanistic features derived from Qwen3-0.6b across layers and attention heads. Using these signals, we train lightweight classifiers and evaluate against LLMs and detection baselines. Furthermore, classifiers trained on Qwen3-0.6b signals generalize to GPT-4.1-mini responses, demonstrating the potential of `proxy-model evaluation`. Our results show that mechanistic signals offer efficient, generalizable predictors for hallucination detection in RAG. [1]

1 Introduction

Large language models (LLMs) excel at tasks such as question answering and summarization [1, 2], yet they frequently generate *hallucinations*—outputs that are factually incorrect or unsupported [3]. Retrieval-Augmented Generation (RAG) reduces hallucinations by grounding responses in external knowledge [4], but models still produce outputs inconsistent with retrieved content [5, 6, 7], making reliable detection essential for domains such as healthcare and finance. Recent advances in RAG hallucination detection follow two directions. Corpus-based approaches (e.g., RAGTruth, LettuceDetect, Mu-SHROOM) provide fine-grained supervision and cross-lingual benchmarks [8, 9, 10]. Mechanistic approaches (e.g., ReDeEP, LRP4RAG) analyze internal activations to disentangle external context from parametric knowledge, showing that overactive FFNs and mis-weighted attention often drive hallucinations [11, 12]. We follow the line of ReDeEP work [11] by introducing **External Context Score (ECS)** and **Parametric Knowledge Score (PKS)** as predictive features for hallucination detection. ECS measures grounding in retrieved content, while PKS captures FFN-driven parametric knowledge injection. Together, they decompose generation into external vs. internal contributions. Using Qwen3-0.6b, we compute ECS and PKS across layers and attention heads, then train regression-based classifiers for hallucination detection. We further demonstrate `proxy-model evaluation`: classifiers trained on Qwen3-0.6b transfer effectively to larger models (e.g., GPT-4-mini), offering scalable, low-cost detection.

Our contributions are threefold:

1. We extend the ReDeEP framework for ECS/PKS computation with a fully open-sourced implementation built on TransformerLens [13], enabling any GPT-like model without modifying the underlying model library.

[1] Code and data: `https://anonymous.4open.science/r/interpretability-hallucination-experiments-75D3`.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

2. We conduct a systematic evaluation of multiple regression-based classifiers, identifying the optimal model for hallucination prediction.
3. We demonstrate that leveraging a 0.6b-parameter model as a proxy allows effective and economical computation, facilitating practical application to large-scale, production-level models.

2 Methodology

Figure 1 outlines our methodology. We first run a standard RAG pipeline to generate responses. The model's *parametric knowledge* is treated as internal knowledge, elicited by prompting with the query. For each context–response span, we compute ECS from attention heads and PKS from feed-forward layers, capturing external vs. internal contributions. After validating their correlation with hallucination labels, these scores serve as features for binary classifiers, whose span-level outputs are aggregated into response-level predictions.

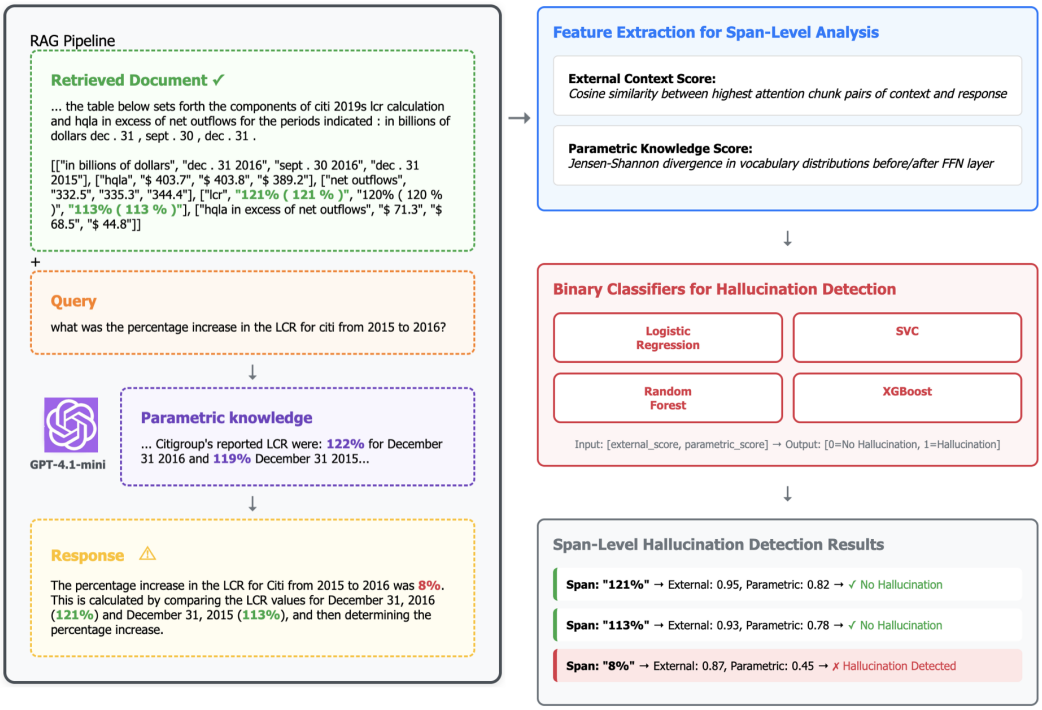

Figure 1: Pipeline for Span-Level Hallucination Detection

Data Curation We use the FinQA dataset from RagBench [14], which contains 12.5k training and 2.29k test instances. Each instance pairs an S&P 500 financial report with a grounded question and an LLM-generated response. For our study, we regenerate responses with Qwen3-0.6b (used for interpretability scores) and GPT-4.1 mini to test cross-model generalization. Our data pipeline consists of three steps (Appendix A shows an example):

- **Response Generation:** We subsample 3,000 training instances from the original dataset and generate responses separately with Qwen3-0.6b and GPT-4.1 mini.
- **Labeling:** We label hallucinated spans using LettuceDetect [9], refined with two LLM judges (llama-4-maverick-17b-128e-instruct and gpt-oss-120b for Qwen3-0.6b-generated response; claude-sonnet-4 for GPT-4.1-mini-generated response to reduce family bias). Majority voting is applied at the response level, and samples are retained if at least one judge confirms the LettuceDetect label, yielding 1,852 instances for analysis.
- **Chunking:** To compute accurate ECS and PKS, we operate at the span level by chunking documents (using `documents_sentences` from FinQA) and splitting responses into individual sentences.

Mechanistic Metrics Our work draws on concepts from mechanistic interpretability research. Specifically, we leverage TransformerLens, an open-source library that provides access to internal model parameters—such as attention heads, feed-forward network layers (FFNs), and residual streams—in GPT-like models. For a detailed description of the architecture, we refer the reader to the original TransformerLens work [13]. We primarily use TransformerLens to compute two metrics: the *External Context Score* and the *Parametric Knowledge Score*, following the definitions in ReDeEP [11]. While the original ReDeEP framework supports both token-level and chunk-level hallucination detection, computing scores at the token level is computationally expensive and does not fully capture context. Therefore, we restrict our calculations at the chunk level. Details about the computation is given in Appendix B.

Classifier for Hallucination Detection Hallucination detection is formulated as a binary classification task. As input features, we use ECS computed for each attention head and layer, together with PKS computed for each layer. Predictions are generated at the span level and can subsequently be aggregated to obtain response-level results. Training details are provided in Appendix C.

3 Experiments

Correlation Analysis We examine mechanistic signals for their correlation with hallucinations in RAG outputs. Comparing ECS values between truthful and hallucinated responses (Figure 2a) shows that hallucinated responses rely less on external context. Pearson correlation analysis using inverse hallucination labels confirms that higher ECS corresponds to lower hallucination likelihood (Figure 2b). While copying-head proxies (`OV_copying_score`) offer interpretable, low-cost approximations [15], we did not observe strong correlations in Qwen3-0.6b, so we include all layers and heads for ECS and apply feature selection during classification. For PKS, later-layer FFNs exhibit higher scores for hallucinated responses (Figure 3a), and Pearson correlations show positive associations with hallucinations (Figure 3b). Together, these results indicate that RAG hallucinations arise from under-utilization of external context and over-reliance on parametric knowledge.

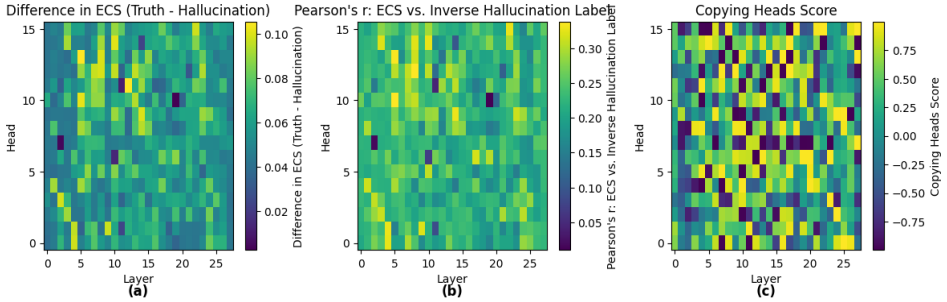

Figure 2: Relationship Between LLM Utilization of External Context and Hallucination

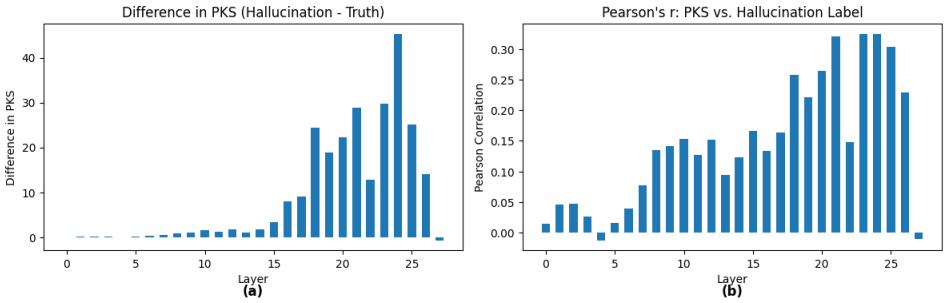

Figure 3: Relationship Between LLM Utilization of Parametric Knowledge and Hallucination

Hallucination Detection Response-level labels are obtained by aggregating span-level predictions from the trained classier: a response is labeled hallucinated if any span is predicted as such. We

evaluate **self-evaluation** (responses from Qwen3-0.6b) and **proxy-based evaluation** (responses from other models, e.g., GPT-4.1 mini). We compare against proprietary models (GPT-5, GPT-4.1), open-source LLMs (GPT-OSS-20b, LLaMA-3.3-70b, LLaMA-3.1-8b, Qwen3-32b, Qwen3-0.6b), and commercial detection systems (RAGAS [16], TruLens [17], RefChecker [18]). Baseline prompts and implementation details of commercial detection systems are in Appendices D and E. Results are summarized in Table 1. Key observations are:

- Overall, our method achieves moderate performance. In self-evaluation, it outperforms TruLens and LLaMA-3.1-8b and matches RefChecker. In proxy evaluation, it surpasses nearly all models except GPT-5 and RAGAS, despite using only a 0.6b parameter model, while higher-performing models are proprietary or much larger.
- We also include Qwen3-0.6b as a detection model to demonstrate that, by itself, the 0.6b model is insufficient for hallucination detection. However, when combined with a strong classifier that leverages its internal signals, performance is substantially improved.
- Our model favors recall over precision, likely due to low-confidence false-positive spans retained during data curation (Section 2). Nonetheless, we argue that higher recall is generally desirable, as false-positive cases can be further verified in downstream.
- In proxy evaluation, all models (from tiny 0.6b to GPT-5) show higher precision than recall. This trend may stem from models such as GPT-4.1-mini producing more reasonable-sounding responses where subtle inaccuracies are harder to detect. While in self-evaluation, errors from Qwen3-0.6b are more readily detected by stronger models.

Table 1: Response-level Detection Performance (%)

	Self-Evaluation			**Proxy-based Evaluation**		
Model	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
GPT-5	77.27	**92.97**	**84.40**	91.67	66.27	**76.92**
GPT-4.1	76.39	85.94	<u>80.88</u>	**94.29**	39.76	55.93
GPT-OSS-20b	82.79	78.91	80.80	92.31	43.37	59.02
llama-3.3-70b-versatile	81.03	73.44	77.05	<u>93.75</u>	18.07	30.30
llama-3.1-8b-instant	69.23	49.22	57.53	70.37	22.89	34.55
Qwen3-32b	79.55	82.03	80.77	86.11	37.35	52.10
Qwen3-0.6b	70.27	20.31	31.52	78.79	31.33	44.83
RAGAS	68.45	<u>89.84</u>	77.70	75.29	77.11	<u>76.19</u>
TruLens	**89.61**	53.91	67.32	49.08	**96.39**	65.04
RefChecker	<u>84.62</u>	68.75	75.86	71.43	12.05	20.62
Ours	63.89	<u>89.84</u>	74.68	62.90	<u>93.98</u>	75.36

Note: RAGAS, TruLens and RefChecker use GPT-4.1 under the hood.

4 Conclusion and Limitations

We developed a RAG hallucinations detection method by decoupling contributions from parametric knowledge and external context. Our correlation analysis shows that hallucinations arise from over-reliance on parametric knowledge and insufficient use of external context. We trained classifiers to predict span-level hallucinations and aggregate them for response-level detection. Additionally, we demonstrated our method as a proxy for evaluating large-scale models at low computational cost.

Despite these strengths, several limitations remain. Computing ECS and PKS across all layers and heads is computationally intensive, particularly when projecting hidden states to vocabulary distributions. Future work could identify the most critical layers—our analysis suggests late-layer signals may suffice—reducing computation without sacrificing accuracy. We also rely solely on ECS and PKS, which limits performance relative to larger models; incorporating uncertainty-based or representation-level features could improve detection. Finally, while mechanistic signals could enable response intervention to steer models toward truthful outputs, effective real-time manipulation would require substantial computational resources and more advanced tooling beyond the current proxy-based framework.

References

[1] Siwei Wang et al. Infibench: Evaluating the question-answering capabilities of code large language models. *OpenReview*, 2024.

[2] Juyong Jiang et al. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[3] Zhiwei Ji, Yiming Zhang, Yicheng Liu, et al. A survey on hallucination in large language models. *arXiv preprint arXiv:2311.05232*, 2023.

[4] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

[5] Juntong Song, Xingguang Wang, Juno Zhu, Yuanhao Wu, Xuxin Cheng, Randy Zhong, and Cheng Niu. Rag-hat: A hallucination-aware tuning pipeline for llm in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1548–1558, 2024.

[6] Haichuan Hu, Congqing He, Xiaochen Xie, and Quanjun Zhang. Lrp4rag: Detecting hallucinations in retrieval-augmented generation via layer-wise relevance propagation. *arXiv preprint arXiv:2408.15533*, 2024.

[7] Likun Tan, Kuan-Wei Huang, and Kevin Wu. Fred: Financial retrieval-enhanced detection and editing of hallucinations in language models. *arXiv preprint arXiv:2507.20930*, 2025.

[8] 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 *Proceedings of ACL 2024*, 2024.

[9] Ádám Kovács and Gábor Recski. Lettucedetect: A hallucination detection framework for rag applications. *arXiv preprint arXiv:2501.00232*, 2025.

[10] Raúl Vázquez, Timothée Mickus, Elaine Zosa, Teemu Vahtola, Jörg Tiedemann, et al. Semeval-2025 task 3: Mu-shroom, the multilingual shared task on hallucinations and related observable overgeneration mistakes. *arXiv preprint arXiv:2504.11975*, 2025.

[11] Zhongxiang Sun, Xiaoxue Zang, Kai Zheng, Jun Xu, Xiao Zhang, Weijie Yu, and Han Li. Redeep: Detecting hallucination in retrieval-augmented generation via mechanistic interpretability. *arXiv preprint arXiv:2410.11414*, 2024.

[12] Yuchen Liu, Yiming Zhang, Hao Chen, et al. Lrp4rag: Detecting hallucinations in retrieval-augmented generation via layer-wise relevance propagation. *arXiv preprint arXiv:2408.15533*, 2024.

[13] Neel Nanda and Joseph Bloom. Transformerlens. `https://github.com/TransformerLensOrg/TransformerLens`, 2022.

[14] Robert Friel, Masha Belyi, and Atindriyo Sanyal. Ragbench: Explainable benchmark for retrieval-augmented generation systems. *arXiv preprint arXiv:2407.11005*, 2024.

[15] Anonymous (TransformerLens Documentation). Transformerlens main demo notebook – eigenvalue copying score analysis. `https://transformerlensorg.github.io/TransformerLens/generated/demos/Main_Demo.html`, 2025. Shows empirical finding that OV_copying_score and full_OV_copying_score are "highly (but not perfectly!) correlated".

[16] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, 2024.

[17] Anupam Datta, Matt Fredrikson, Klas Leino, Kaiji Lu, Shayak Sen, Ricardo Shih, and Zifan Wang. Exploring conceptual soundness with trulens. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, volume 176, pages 302–307. PMLR, 2022.

[18] Xiangkun Hu, Dongyu Ru, Lin Qiu, Qipeng Guo, Tianhang Zhang, Yang Xu, Yun Luo, Pengfei Liu, Yue Zhang, and Zheng Zhang. Refchecker: Reference-based fine-grained hallucination checker and benchmark for large language models. *arXiv preprint arXiv:2405.14486*, 2024.

A Preprocessing Example

A sample entry from our preprocessed dataset is shown below:

```
question: "What is the growth rate in CCG operating income in 2015?"

documents: [
  "Management 2019's discussion and analysis of financial condition
     and results of operations (continued). "
  "The following results drove changes in CCG operating income by
     approximately the amounts indicated:"
]

prompt: "Given the context, please answer the question based on the
    provided information from the context. Include any reasoning with
    the answer.\n\nContext: Management 2019's discussion and analysis
    of financial condition and results of operations (continued). The
    following results drove changes in CCG operating income by
    approximately the amounts indicated:\nQuestion: What is the growth
     rate in CCG operating income in 2015?\nAnswer:"

response: "The exact growth rate is derived from the differences in
    the values provided. Thus, the answer is that the growth rate in
    CCG operating income in 2015 is approximately -20.7%."

labels: [
  {start: 673, end: 722, confidence: 0.8037, text: "Growth Rate} = \\
     frac{-2161}{10327} \\times 100\\%"},
  {start: 730, end: 734, confidence: 0.5833, text: "20.7"}
]

labels_llama: 1
labels_gpt: 1

prompt_spans: [[0,135],[136,144],[145,348],[349,1177],[1178,3465]]
response_spans: [[0,383],[384,597],[598,663]]
```

Notes:

- `labels` provide span-level hallucination annotations with start and end indices in the response text.
- `labels_llama` and `labels_gpt` indicate whether the respective LLM judges marked the response as hallucinated (1) or not (0).
- `prompt_spans` and `response_spans` segment the prompt and response into sentence or phrase-level chunks for span-level scoring.

B Computation Details of Mechanistic Metrics

External Context Score: The External Context Score (ECS) quantifies the extent to which a language model leverages external context when generating a response. In mechanistic interpretability, attention heads are responsible for retrieving relevant information from the context. To measure this utilization, ECS captures the semantic alignment between response segments and the context chunks most strongly attended to by the model.

Let the external context be partitioned into chunks $\{\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_M\}$, and the generated output into response chunks $\{\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_N\}$, corresponding to `prompt_spans` and `response_spans` in

Appendix A. For each attention head h at layer l, the most relevant context chunk for each response chunk $\tilde{r}_j$ is identified as

$$\tilde{c}_j^{\ell,h} = \arg\max_{\tilde{c}_i} A(\tilde{r}_j^{\ell,h}, \tilde{c}_i^{\ell,h}),$$

where A denotes the token-level attention weight matrix, and l and h indicate the layer and head indices. The chunk-level ECS is defined as the cosine similarity between the embeddings of $\tilde{r}_j$ and its corresponding context chunk $\tilde{c}_j$:

$$\mathrm{ECS}_{\tilde{r}_j}^{\ell,h} = \cos\left(e(\tilde{r}_j^{\ell,h}),\, e(\tilde{c}_j^{\ell,h})\right),$$

where $e(\cdot)$ represents the embedding function, and $\cos(\cdot,\cdot)$ denotes cosine similarity.

Parametric Knowledge Score: The Parametric Knowledge Score (PKS) quantifies the extent to which the FFN contributes to *parametric knowledge*, i.e., knowledge stored in the model's weights, as opposed to information coming from external context. This is done by measuring the difference between residual stream states **before** the FFN layer and **after** the FFN layer.

Since the residual stream itself does not directly indicate "which token is being suggested"—it is a latent vector—we map it through the unembedding/projection matrix to the vocabulary distribution. This allows us to observe how the residual change after the FFN influences predicted token probabilities.

We then apply the Jensen-Shannon divergence (JSD) to compute the distance between the two vocabulary distributions, which defines the token-level PKS. The chunk-level PKS is computed by averaging the token-level PKS over all tokens in the chunk. Mathematically, this is expressed as

$$\mathrm{PKS}_{t_n}^{\ell} = \mathrm{JSD}\left(p(x_n^{\mathrm{mid},\ell}),\, p(x_n^{\ell})\right), \;\; \mathrm{PKS}_{\tilde{r}}^{\ell} = \frac{1}{|\tilde{r}|} \sum_{t_n \in \tilde{r}} \mathrm{PKS}_{t_n}^{\ell}.$$

where $p(\cdot)$ denotes the mapping from residual stream states to vocabulary distributions, and $x_n^{\mathrm{mid},\ell}$ and x_n^{ℓ} refer to the residual stream states before and after the FFN layer, respectively. $\mathrm{PKS}_{t_n}^{\ell}$ and $\mathrm{PKS}_{\tilde{r}}^{\ell}$ stand for token-level and chunk-level PKS, respectively.

The computation was performed using the TransformerLens library on the Qwen3-0.6B model. Inference was executed on a Google Colab L4 GPU with 24-GB memory, using `torch.float16` precision to reduce activation storage costs. Sentence-level semantic similarity was computed using the BAAI/bge-base-en-v1.5 encoder, also hosted on GPU to avoid transfer overhead.

The average execution time for an end-to-end ECS/PKS computation was 42 seconds per example. GPU allocation remained within 1.9–2.1 GB, with reserved memory peaking at 2.2 GB across iterations. After each iteration, tensors were explicitly released, and memory was reclaimed using `torch.cuda.empty_cache()` and `torch.cuda.ipc_collect()` to prevent fragmentation.

C Training Details of Classifiers

Training was conducted at the span level using 1,852 instances, corresponding to 7,799 span-level samples (4,406 negative, 3,393 positive). We used **External Context Scores** (ECS) and **Parametric Knowledge Scores** (PKS) as input features. For Qwen3-0.6b (28 layers, 16 attention heads per layer), this initially yielded 476 features, which were reduced to 341 after feature selection. The dataset was split 90/10 for training and validation with stratification on hallucination labels. Feature preprocessing included standardization (`StandardScaler`), removal of near-constant and duplicate features (`DropConstantFeatures`, `DropDuplicateFeatures`), and correlation-based filtering (`SmartCorrelatedSelection`, Pearson threshold 0.9) using a `RandomForestClassifier` with max depth 5. Four classifiers—Logistic Regression, Support Vector Classification (SVC), Random Forest, and XGBoost—were evaluated using pipelines combining preprocessing and classification. Random Forest and XGBoost had max tree depth 5 with other hyperparameters default. SVC achieved the highest validation F1 score and was selected as the final model, while XGBoost, despite strong training performance, overfit severely, highlighting the need for regularization or more data. Overall, SVC provided the best balance between complexity and generalization (see Table 2).

Table 2: Span-level Detection performance (%)

Classifier	Train Prec.	Val Prec.	Train Rec.	Val Rec.	Train F1	Val F1
LR	79.71	74.84	77.05	71.09	78.36	72.92
SVC	84.44	79.00	79.24	74.34	81.76	76.60
RandomForest	79.87	74.92	76.13	72.27	77.95	73.57
XGBoost	99.74	76.45	99.77	73.75	99.75	75.08

D Prompt for Baselines

Below is the prompt used for hallucination detection when using models GPT-5, GPT-4.1, GPT-OSS-20b, LLaMA-3.3-70b-versatile, LLaMA-3.1-8b-instant, Qwen3-32b and Qwen3-0.6b.

User Prompt

```
You are an expert fact-checker. Given a context, a question, and a response,
your task is to determine if the response is faithful to the context.
Context: context
Question: question
Response: response
Is the response supported and grounded in the context above? Answer "Yes"
or "No", and provide a short reason if the answer is "No". Be concise and
objective.
```

E Implementation of Commercial Detection Systems

We describe our implementation of three commercial tools—RAGAS, TruLens, and RefChecker—for hallucination detection below.

RAGAS: We use RAGAS's `faithfulness` metric to evaluate how well a model's responses align with the provided context documents. GPT-4.1 is used as the evaluator model. For each data point, a faithfulness score between 0 and 1 is computed. To determine the optimal threshold that maximizes F1 score, we evaluate F1 across candidate thresholds in [0,1]. Predictions are binarized at each threshold, and the threshold that maximizes F1—balancing precision and recall—is selected.

TruLens: TruLens provides a framework for evaluating hallucination via its groundedness feedback mechanism. We use the `groundedness_measure_with_cot_reasons` function to compute groundedness scores in [0,1]. The F1-optimal threshold is determined using the same procedure as in RAGAS.

RefChecker: RefChecker extracts factual claims from model responses and verifies them against reference documents. It consists of two components: an `LLMExtractor` that extracts claims, and an `LLMChecker` that classifies claims as `Entailment`, `Neutral`, or `Contradictory`. A response is labeled hallucinated if any claim is `Contradictory`, and non-hallucinated if all claims are either `Entailment` or `Neutral`.