ChatGPT

Reducing Hallucinations in Large Language Models (LLMs)

Hallucinations – outputs that are fluent but factually incorrect or unsupported – remain a key challenge for deploying LLMs in real-world applications [1] [2]. Below we explore **methods to prevent hallucinations** during generation and **techniques to detect** them post-hoc. We cover strategies for open-ended generation, retrieval-augmented generation (RAG), summarization, and other use cases (code, medical, legal), comparing their effectiveness, speed, readiness, scalability, and domain generality.

Hallucination Prevention Techniques

Prompt Engineering and Guardrails

Careful prompt design can **guide models away from making up facts**. Explicit instructions not to guess and to stay truthful can noticeably cut down hallucinations [3]. For example, simply telling the model *what not to do* (e.g. "do not fabricate information") significantly lowered hallucination rates, and *suggesting alternative actions* (like "if you don't know, say you cannot answer") reduced them even further [4]. Structured or tagged prompts provide additional constraints: using fixed templates or special tokens to anchor the model's output to given facts has been shown to nearly eliminate fabricated details (one study achieved **98.9% success** in avoiding made-up info by embedding context tags) [5] [6]. Similarly, frameworks like **NVIDIA NeMo Guardrails** or Microsoft's **Guidance** let developers enforce rules (e.g. requiring the model to cite a source for factual claims) and control output format, helping catch or prevent unsupported content. These prompt/guardrail solutions are **fast (no extra model calls)** and easy to implement, making them attractive for production. However, they are not foolproof – overly strict instructions can degrade answer completeness or lead the model to over-use refusals, and clever prompts only mitigate hallucinations rather than wholly eliminate them.

Retrieval-Augmented Generation (RAG)

Grounding the model's output in external knowledge is one of the most effective defenses against hallucinations. RAG pipelines supply the LLM with relevant documents or facts (from a vector database, search engine, or knowledge graph) **before or during generation**, so the model can focus on truthful information instead of relying purely on its internal weights [7] [8]. By **augmenting prompts with retrieved evidence**, RAG greatly improves factual accuracy – effectively mitigating hallucinations in many settings [9] [10]. This approach works across domains: e.g. providing up-to-date medical research excerpts for a health query or law texts for a legal question can keep the model's answers grounded in those sources. RAG is already a **production-ready** strategy (used in search chatbots and enterprise QA systems) and is **scalable**: modern vector databases handle large corpora, and retrieval adds only modest latency (typically tens to a few hundred milliseconds). The key caveat is that RAG is only as good as the retrieved context – if the knowledge base lacks an answer or returns irrelevant passages, the LLM may still hallucinate or confidently assert unsupported facts [11]. To address this, RAG deployments often include safeguards: for instance, if no high-similarity document is found, the system can prompt the model to

respond with uncertainty or an apology instead of inventing an answer. Overall, grounding with RAG markedly reduces hallucinations **without needing to modify the base model**, making it a practical high-impact solution in many domains.

Chain-of-Thought and Reasoning Strategies

Another prevention tactic is to have the model **reason step-by-step** or verify details as it generates an answer. Techniques like *chain-of-thought (CoT) prompting* encourage the model to break down the problem and check intermediate facts, which can catch logical inconsistencies and curb some hallucinations [12]. CoT prompting (e.g. "let's think this through step by step") has been shown to **mitigate hallucinations in many situations** by focusing the model on reasoning instead of jumping to a conclusion [12]. For example, in a multi-hop QA, the model can be prompted to retrieve or recall relevant facts one by one and verify each, rather than making a wild guess. Another variant is *chain-of-verification*, where the model generates a draft answer and then systematically checks each claim (possibly querying a tool or knowledge source for each) – this approach significantly reduced factual errors in research experiments [13]. These reasoning-based methods **improve reliability** but often incur **latency costs**: the model may need to perform multiple inference steps or tool calls. In practice, CoT prompting is still fast (just a single prompt with the instruction to reason), while more elaborate verify-and-refine loops can slow down responses. They are reasonably general (logical reasoning helps in math, code, and complex QA tasks, and verification steps help ensure faithfulness in summarization). Many production systems use simplified versions of this idea, like having the model double-check its answer with a second pass or using an agent that can call external APIs (e.g. a calculator for math, a web search for open-domain facts) – trading a bit more computation for a safer answer.

Model Selection and Training (RLHF, Fine-Tuning)

Choosing a more robust model or further training it can greatly reduce hallucinations from the outset. Larger, more advanced models tend to hallucinate less: for instance, **GPT-4 substantially outperforms GPT-3.5** in factuality, making far fewer incorrect assertions under the same conditions [14]. If latency permits, using a state-of-the-art model is a straightforward way to improve truthfulness. **Temperature tuning** also matters – a low decoding temperature (closer to 0) makes the model deterministic and likely to stick to high-probability (usually safer) statements, whereas a high temperature can produce more diverse but less grounded outputs [15]. Beyond model choice, **Reinforcement Learning from Human Feedback (RLHF)** and instruction fine-tuning have been key to aligning LLMs with factuality. RLHF optimization can explicitly penalize blatant hallucinations and reward correct, helpful answers, yielding models that are **noticeably more truthful and grounded** after instruction tuning [16]. In fact, modern chat models (OpenAI, Anthropic, etc.) undergo such alignment training, which has led to marked improvements in hallucination mitigation [17]. Fine-tuning on domain-specific data is another lever: if you have a domain (e.g. legal contracts, medical texts), fine-tuning an LLM on that **high-quality, factual corpus** can anchor its knowledge and style to the domain, reducing the chance of out-of-scope fabrications [18]. OpenAI has noted that a fine-tuned GPT-3.5 on a specialized task can in some cases **outperform a zero-shot GPT-4** for that domain [19] – partly because the fine-tuned model learns to not stray beyond the provided data or expected answers. These training-based solutions require **more upfront investment** (compute for fine-tuning or RLHF, and careful curation of training signals), but once deployed, they do not add runtime cost – the model is inherently more reliable. They are already used in production (virtually all top-tier LLM services use RLHF). A limitation is domain generality: a model fine-tuned to be factual in one area (say medical Q&A) might still hallucinate in another (like programming) if it lacks knowledge there. Nonetheless, combining a **well-**

trained base model with the above prompt and retrieval strategies yields a strong foundation for hallucination prevention.

Domain-Specific Tools and Checks

In certain use cases, integrating **specialized tools** can virtually eliminate hallucinations of a particular kind. For example, in code generation, hallucinations might manifest as syntactically incorrect code or calls to non-existent functions. Many coding assistants now run the generated code (or at least a syntax check/test suite) and have the model **self-correct based on errors**, which addresses hallucinated code issues effectively (the model learns from the runtime feedback). For arithmetic or scientific questions, hooking the LLM up to a calculator or a knowledge base for formulas ensures it doesn't invent numeric values. These tool-augmented approaches essentially prevent hallucinations by delegating to a system that **guarantees correctness** in its domain (e.g. a compiler for code, a database for factual lookups). They are highly effective within their scope (a math tool won't help prevent a historical date error, but will stop math mistakes). The latency depends on the tool – e.g. running code or querying an API adds time – but for many applications the trade-off is worth it for accuracy. The scalability of tool use is usually bounded by how often queries need to be made (which can often be tuned, e.g. only call the tool if the model's answer is uncertain or requires verification). In terms of production readiness, this approach is growing: frameworks exist for LLMs to call tools, and products like Bing Chat or Wolfram-Alpha integration demonstrate its viability. **Domain generality is achieved by using the right tool for the domain** – legal LLMs might query a statute database, medical LLMs might cross-check a symptoms database, etc. While implementing tool use requires additional infrastructure, it can **drastically reduce hallucinations in critical domains** by offloading tasks that the free-form generative model isn't trustworthy for.

(Also, note that beyond model-centric methods, product design choices can mitigate the impact of hallucinations: for instance, allowing users to edit or approve AI-generated content, providing source citations for users to verify [20] [21], limiting response length (since long outputs often accumulate more errors), or offering a "precision mode" that might use a stricter model or style [22] [23]. These measures don't stop a model from hallucinating, but they reduce the chance that unverified information slips through to end-users. In high-stakes deployments, combining such product-level guardrails with the model techniques above is common.)

Hallucination Detection Techniques

Even with preventive measures, some hallucinations will occur. Therefore, robust **detection mechanisms** are needed to identify when an LLM's output may be untrue or unsupported. Detection can trigger an alert, a correction step, or a refusal before misinformation reaches the user. We outline key detection methods and their trade-offs:

Knowledge-Based Fact-Checking (Retrieval-Based Detection)

One approach is to **fact-check the model's output against trusted external data**. This often involves extracting the key claims or entities from the LLM's response and then querying a knowledge source (a search engine, database, or Wikipedia index) to see if those claims are supported. If the external evidence can't be found, or contradicts the claim, the output is flagged as a hallucination. Research shows this retrieval-based evaluation correlates well with human judgments of factuality [24] [10]. For example, the FActScore metric breaks a generation into atomic facts and checks each against Wikipedia; using retrieval in this way yielded better factuality judgments than relying on an LLM alone [25] [26]. In practice, a system

might take a chatbot answer about a medical condition, extract statements like "X is a symptom of Y," and lookup a medical database or perform a web search to verify those statements. This method can achieve high **detection accuracy** when the knowledge source is comprehensive – it will reliably catch outright false claims that have no support in the corpus. It's fairly **domain-general** as long as you have a relevant knowledge base (e.g. a legal case database for legal answers, etc.), and it can be scaled by caching and parallelizing searches. The downsides are **latency** (each claim might require a search query or database hit, which can slow down response finalization) and the risk of false positives if the knowledge base is incomplete (the system might flag a true statement that simply isn't in the database). Overall, retrieval-based detection is **production-feasible** – indeed, some AI writing assistants already include a "fact check" button that effectively performs a web search on the output. Tools like the *SAFE* evaluator even use an LLM agent to iteratively issue search queries and reason about the results, outperforming individual human fact-checkers on some long-form answers [27]. While a multi-step agent may be too slow for realtime use, a simplified retrieval check (one or two queries) is often acceptable for high-value use cases, providing a strong backstop against hallucinations.

Model-Based Verification (NLI and QA Evaluation)

Another category of detectors uses **auxiliary AI models to judge the factual consistency** of an LLM's output. One common technique is to treat it as a Natural Language Inference (NLI) problem: for tasks with a source (like summarization or translation), the model output is the "hypothesis" and the original text is the "premise" – an NLI model can predict if the hypothesis is entailed, contradicted, or unrelated to the premise. A specialized factuality classifier (like FactCC for summaries) will label an output as faithful or hallucinated based on features of the source and summary [28]. These learned detectors can achieve good accuracy – often better precision/recall than simple heuristics – **but rely on high-quality training data** and tend to be task-specific [29] [30]. Another variant is **question-answering (QA) based checks**: the system generates questions from the LLM's output and tries to answer them using the source or a search engine, then compares if the answers match. This method (e.g. Q^2 and SelfCheckGPT) implicitly verifies each factual claim. It has shown strong results in detecting hallucinations in dialogues and summaries by catching discrepancies between the generated answer and source-based answers [31] [32]. Model-based verification can even be done with a large LLM as the checker – for instance, using GPT-4 to evaluate GPT-3.5's answer. This often yields very high accuracy since GPT-4 knows a lot of facts and can reason, but it's **computationally expensive** to double-up models (and not infallible either). For better efficiency, open-source NLI or QA models (e.g. a fine-tuned RoBERTa or T5) can be employed; these run fast (milliseconds per example) and can be integrated into pipelines. Many current evaluations of LLMs use such automatic metrics (like an entailment score or a QA-based consistency score) as a proxy for factual correctness. In production, an NLI/QA checker could automatically refuse or redact parts of an answer that the checker model deems unsupported. The **latency** is minimal (a single forward pass of a smaller model), so it's quite scalable. However, **domain generalizability** may be limited – a classifier trained on news summaries might not generalize to code or medical text without retraining. Maintaining separate detectors for each domain/task is possible but adds complexity. Despite these challenges, **production readiness** is growing: for example, companies have released open models like *Vectara's HHEM* specifically to detect hallucinations in a RAG setting [33]. In summary, model-based verification provides a fast, automated check with decent accuracy, especially when specialized to the content domain, serving as a valuable line of defense.

Heuristic Signals: Uncertainty and Self-Consistency

A more lightweight detection approach is to examine **signals from the LLM itself**. One such signal is the model's own *confidence or uncertainty*. If the model can estimate how likely its answer is correct, we could flag low-confidence answers. In practice, raw output probabilities are not always well-calibrated – LLMs often sound confident even when wrong. However, research indicates larger models are somewhat better calibrated on multiple-choice questions [34], and certain formats (like having the model explicitly rate its confidence) can help. Some papers have found that the internal activations of a model contain clues about whether it "knows" the answer or is guessing [35] [36]. Leveraging this, one could train a secondary model to read the LLM's hidden state or logits and predict hallucination likelihood. This approach can be **fast (no external calls)**, but it's not trivial to deploy unless you have low-level access to the model internals (which is easier with open-source models than with closed APIs). Moreover, if the model is *over*-confident in a wrong answer, uncertainty-based detection will fail – and indeed, uncalibrated confidence is a known issue [37]. Another heuristic method is **self-consistency checks**: essentially, ask the model (or ensemble of models) the same question in different ways or multiple times with some randomness, and see if the answers agree. The intuition is that when the model truly knows the answer, it will consistently reproduce it, but when it's hallucinating, different runs might produce conflicting information. SelfCheckGPT implemented this by sampling multiple continuations and checking for factual consistency across them [32]. In cases like open-ended fiction or opinion, inconsistency isn't a red flag, but in factual QA or translation, significant divergence suggests at least some of those answers are wrong. Self-consistency detection does catch certain errors (especially logical contradictions), and it doesn't need an external reference, making it domain-agnostic. The trade-off, however, is **latency and cost** – generating say 5 answers instead of 1 can be five times slower and use five times more tokens, which might not be acceptable in a live setting. It also won't catch a hallucination that the model **repeats consistently** (e.g. if the model always erroneously claims a specific false fact due to a training bias, all samples might agree on the same falsehood). Thus, consistency checks are a useful supplement but not a standalone solution for subtle factual errors [38]. In summary, heuristic signals like uncertainty and answer stability provide quick hints of hallucination with **minimal overhead**, but they are less reliable than knowledge- or model-based methods. They might be used for preliminary screening (e.g. "if the model seems unsure, route to a more rigorous fact-checker or have it ask for clarification from the user").

Learned Hallucination Classifiers

With enough labeled examples of hallucinated vs. truthful outputs, one can train a **supervised classifier** to detect hallucinations. This could be a standalone model (e.g. a smaller transformer that takes in the LLM's response and possibly the prompt or source) and outputs a binary label or a "hallucination score." Such classifiers have been built in academia and competitions – for instance, the *SHROOM* task at SemEval 2024 had participants develop classifiers to identify hallucinated translations/summaries [39] [40]. The top methods often combined multiple features: similarity to source text, perplexity, entailment scores, etc., feeding them into a logistic regression or neural network [41] [42]. These approaches can achieve quite high detection accuracy on the data they're trained on (one ensemble reached ~82.6% accuracy in a model-agnostic hallucination detection track [43]). In specific domains (like biomedical summaries), a fine-tuned detector can learn domain-specific cues for factual errors. The benefit is that once trained, the classifier is **fast at inference** and straightforward to deploy alongside the LLM. Many companies haven't published their own hallucination classifiers, but it's plausible that internal evaluation models exist (similar to how toxicity classifiers are used in production to filter outputs). The challenge is **generalizability and maintenance**: a classifier trained on one type of content might not work on another type. If the distribution

of prompts or the LLM's style shifts (say you switch to a new model), the detector may need retraining or at least re-calibration. There's also a **data bottleneck** – obtaining high-quality ground truth labels for hallucinations can be expensive, since it requires experts to verify each output. Despite these hurdles, this approach is promising when paired with a specific application: e.g. a company could curate hundreds of QA pairs where the AI answered either correctly or with hallucinations, and train a model to recognize patterns. In production, a learned detector could give a probability that "this answer contains unsupported info" and if above a threshold, the system might either suppress the answer or append a disclaimer. In practice, purely learned detectors are **less common in current deployments** (compared to on-the-fly fact retrieval or prompt-based safeguards), but research surveys suggest combining a learned approach with other signals is the most robust strategy [44]. For example, one might use a classifier together with a retrieval check – any response flagged by either the knowledge check or the classifier would be treated as potentially hallucinated. This layered approach can yield very high reliability, at the cost of added complexity.

Comparison and Practical Insights

No single method completely solves hallucinations, but **combining complementary techniques** can dramatically reduce them to acceptable levels [45]. Here we compare methods across key dimensions:

- **Effectiveness:** Using external knowledge is among the most effective mitigations. RAG (prevention) ensures most statements are backed by data, and retrieval-based **post-hoc checks** catch many unsupported claims – making knowledge integration a high-impact solution for factuality [9] [10]. Model improvements (size and RLHF) also yield large gains in truthfulness (e.g. GPT-4's jump in reliability over GPT-3.5) [14]. Prompt engineering and guardrails can handle many surface-level hallucinations (especially extrinsic ones where the model wanders off topic), though intrinsic inaccuracies can slip through if the model is confident in a wrong "fact". Self-consistency and uncertainty heuristics tend to be moderately effective; they catch blatant issues (or signal when the model is guessing), but **miss subtle errors** and can be fooled by confident gibberish [46]. Ultimately, the highest accuracy in detection comes from **layering methods** – e.g. a system might use a retrieval check and an NLI model and only trust the answer if both find it supported. Research surveys conclude that a **hybrid approach** (combining learning-based, retrieval, and other signals) is the most robust for detection [44]. In prevention too, systems often stack methods (for example, an aligned model with RLHF *and* RAG *and* carefully crafted prompts) to minimize hallucinations from all angles.

- **Latency:** There is a spectrum from virtually zero overhead methods to those that significantly slow down response time. Prompt tweaks, lower temperature, or a fine-tuned model incur **no additional latency per request** – they change the output quality directly. RAG adds a bit of latency for the retrieval step, but this is often on the order of milliseconds to a couple seconds depending on the complexity of the query and size of the index; for many enterprise applications, this delay is acceptable given the accuracy boost. On the other hand, strategies like self-consistency (multiple model calls) or agent-based verification (iterative search queries) can multiply the latency – these might be relegated to offline analysis or on-demand checks rather than every user query. Learned classifiers and lightweight NLI/QA models are very fast (often <100ms) and can be deployed in parallel with generation, so they usually don't become a bottleneck. In summary, methods like **prompt engineering and classification are "fastest"**, whereas heavy retrieval or multi-call approaches are slower. A practical deployment often uses a cascading approach: fast checks first,

and slower, thorough checks only if needed (e.g. if the fast check suspects a problem and the context is high-stakes, then do a more in-depth verification).

- **Production Readiness:** Today, **RAG and RLHF-aligned models are widely used in production**, underscoring their readiness and trustworthiness [9] [16]. There are established tools and platforms for building RAG systems and many off-the-shelf models fine-tuned with RLHF that one can use via APIs. Prompt engineering is essentially part of normal prompt design for any LLM integration. Guardrails frameworks (like NeMo Guardrails) are relatively new but have seen uptake for enterprise AI where control is paramount – they are production-ready in the sense that major vendors support them, though they require careful rule-writing by developers. Automatic hallucination detectors are just beginning to transition from research to practice. We don't yet see many public "factuality check" APIs, but organizations with sensitive applications are building custom detectors (for instance, Meta's *FacTool* prototype for multi-domain fact checking [47] [48], or internal evaluation suites at AI firms). For an end-user product, a simpler implementation like highlighting sentences that *might* be incorrect (using an NLI model or even GPT-4 in evaluation mode) can be done right now using available models. The **most production-ready detection** methods are those piggybacking on existing mature tech: using a search engine for verification or using a well-known NLI model. In contrast, methods requiring deep model internals or large ensembles (like reading the hidden state or running 10 samples) are not commonly integrated into real-time systems yet.

- **Scalability:** Methods that require heavy computation per query (like multiple LLM calls or exhaustive searches) will scale worse under high load. RAG itself scales well with appropriate infrastructure – vector databases are designed for scale, and one can cache frequent queries. High-throughput applications (e.g. customer support bots answering thousands of queries) benefit from **caching retrieved contexts** and even caching model outputs for repeated similar questions. Detection via small models or heuristics scales easily (they can often run on CPU or a single GPU for many parallel requests). A potential scalability concern is *cost*: using a 175B parameter model for every verification is expensive, so some companies use a two-tier setup (e.g. a cheaper model for initial answer and only invoke the expensive model to check or regenerate if needed). Human-in-the-loop review, while the gold standard for quality, is the least scalable – it's used only for sampling and improving systems over time, not per query, due to cost and latency. Thus, scalable deployments lean on **automated, low-cost detectors** and efficient retrieval. Importantly, scaling to new domains might require adding new data (for RAG) or fine-tuning detectors, but these are one-time or infrequent costs.

- **Domain Generalizability:** In domains like **medical, legal, or coding**, the tolerance for hallucination is low, and specialized challenges emerge. No single method works universally best across all domains, but some adapt better than others:

- *Retrieval augmentation* is extremely general-purpose – you just need the right knowledge source. It shines in knowledge-dense domains (law, finance, medicine) where up-to-date, verifiable information is essential. The system must ensure the model actually uses the retrieved info; prompts that instruct "use only the provided documents" help here. In summarization, providing the source text to the model is a form of RAG that constrains it to factual content (and any summary fact not found in the source can be treated as a hallucination).
- *Fine-tuning on domain data* can instill domain-specific knowledge and terminology, reducing the model's tendency to improvise. For example, a medically fine-tuned model will know standard

protocols and is less likely to output a nonsensical treatment. However, it might also become **overconfident in its domain** – pairing it with retrieval of medical literature and a verification step (perhaps an NLI model trained on medical texts) yields greater reliability.

- *Detection in specialized domains* often requires domain knowledge: a general NLI model might fail to catch a subtle legal error. For critical domains, it's worth training or at least evaluating detectors on domain-specific hallucinations (e.g. have it focus on checking citations in legal answers, or verifying units and values in scientific answers). Alternatively, using domain-expert tools (like a drug interaction database to verify a medical advice output) can serve as a precise detector for certain statements.
- Code is a special domain where hallucinations manifest as incorrect logic or API misuse. Here, execution is the ultimate test – if the code runs and produces the expected result, it's not hallucinated. Many coding assist tools now do this behind the scenes. For non-executable outputs (like an algorithm description), one might use test cases or known problem solutions as references. The structured nature of code also allows format-based detection (e.g. if the model output calls a function that doesn't exist in a library, a simple reference check can flag it).

In practice, deploying LLM solutions in high-stakes fields uses a **belt-and-suspenders approach**. For example, a medical chatbot might use a **grounded LLM (RAG + fine-tuned model)** to generate an answer, then run a **fact-checking pass**: cross-checking critical facts against a medical database and maybe having a smaller model classify if the advice seems unsafe or hallucinated. Only if all checks pass does the answer reach the user, possibly with sources cited. This layered approach may increase complexity and inference time, but it significantly boosts trust. As one AI researcher aptly summarized, with current LLM technology hallucinations are an expected side-effect that cannot be entirely eliminated, **but a combination of strategies can mitigate them to a level where they're manageable for most use cases** [45]. By carefully balancing accuracy and latency – perhaps using fast automatic checks for routine cases and slower, rigorous checks for answers that matter the most – organizations are increasingly able to **reap LLM benefits while keeping hallucinations in check**.

Sources:

- Survey of hallucination causes, detection, and mitigation in LLMs [49] [46]
- Blog: *Mitigating LLM Hallucinations: a multifaceted approach* [9] [14] [16]
- Weng, Lilian. "Extrinsic Hallucinations in LLMs." (2024) – discussion of factuality eval methods [10] [27]
- SemEval 2024 *SHROOM* competition overview – detecting hallucinations in generation [50] [41]
- Vectara Open Source Hallucination Detector (HHEM) and SelfCheckGPT QA approach [33] [31]
- OpenAI alignment notes – TruthfulQA benchmark results (model vs human truthfulness) [51]
- Various research works on prompt techniques (e.g. tagged prompts, CoT), RAG, and tool use for grounding [52] [12] [53]

---

[1] [5] [6] [7] [8] [28] [29] [30] [37] [38] [44] [46] [49] [52] Large Language Models Hallucination: A Comprehensive Survey
https://arxiv.org/html/2510.06265v2

[2] Two Types of LLM Hallucinations
https://galileo.ai/blog/deep-dive-into-llm-hallucinations-across-generative-tasks

3 4 9 11 12 14 15 16 17 18 19 20 21 22 23 45 53 Mitigating LLM Hallucinations: a multifaceted approach - AI, software, tech, and people. Not in that order. By X
http://amatria.in/blog/hallucinations

10 24 25 26 27 34 47 48 51 Extrinsic Hallucinations in LLMs | Lil'Log
https://lilianweng.github.io/posts/2024-07-07-hallucination/

13 35 36 GitHub - EdinburghNLP/awesome-hallucination-detection: List of papers on hallucination detection in LLMs.
https://github.com/EdinburghNLP/awesome-hallucination-detection

31 32 33 39 40 41 42 43 50 Detecting LLM hallucinations and overgeneration mistakes @ SemEval 2024 | by Alejandro Mosquera | Medium
https://medium.com/@alejandro-mosquera/detecting-llm-hallucinations-and-overgeneration-mistakes-semeval-2024-cbd54200bb60