# Output Variability in Corpus-Based Large Language Model Analyses Over Time

## I. Introduction: The Challenge of Consistent LLM Outputs in Complex Analyses

### A. Contextualizing the Query

The application of Large Language Models (LLMs) to support complex decision-making processes is rapidly expanding. A significant use case involves LLMs examining a specified corpus of relevant documents and data to evaluate alternatives and outcomes. This often implies the utilization of techniques such as Retrieval Augmented Generation (RAG), where the LLM's responses are grounded in a defined knowledge base to enhance factual accuracy and relevance.[1] A primary concern for users in such scenarios is the consistency of the LLM's output. Specifically, if an analytical prompt, designed to probe complex situations, is submitted to the LLM at different points in time, what degree of variation in the generated answer should be anticipated, and what are the fundamental reasons for such discrepancies? The nature of these tasks, often involving high-stakes analysis in fields like enterprise decision support, legal reasoning, or medical assessment, means that even minor variations in output could carry substantial consequences. Understanding the stability of LLM responses is therefore not merely a technical curiosity but a practical necessity for reliable deployment.

### B. The Criticality of Understanding Output Variability

For tasks demanding nuanced analysis of complex situations, the consistency and reproducibility of LLM-generated outputs are paramount for building user trust and ensuring dependable decision-making frameworks. Inconsistent outputs, where an LLM provides different answers or justifications to the same query over time, can significantly undermine the perceived utility, reliability, and safety of LLM-driven analytical tools.[3] This report aims to provide an expert-level dissection of the multifaceted factors contributing to LLM output variability when prompts are repeated. It will draw upon current research to explore the inherent characteristics of LLMs, the impact of their operational environments, and the specific considerations for systems leveraging external knowledge corpora. It is important to establish from the outset that achieving perfect, bit-for-bit identical output from LLMs across different points in time, especially those accessed via APIs, is often an unrealistic expectation.[3] Consequently, the focus shifts from pursuing absolute determinism to understanding the nature and extent of potential variations and developing strategies

to manage them effectively.

### C. Report Roadmap

This report will systematically explore the landscape of LLM output variability. It begins by examining the fundamental mechanisms of LLM text generation and the inherent stochasticity involved. It then delves into the reasons for non-determinism even when settings are configured for consistency. Subsequently, the report addresses how the passage of time, encompassing model updates and API changes, influences response divergence. The specific role of Retrieval Augmented Generation (RAG) systems in this context is analyzed, followed by a characterization of the types and extent of expected variations. Finally, strategies for navigating and potentially mitigating these variations are discussed, culminating in a set of concluding remarks.

## II. Fundamentals of LLM Output Generation and Inherent Stochasticity

### A. The Probabilistic Core: Next-Token Prediction

At their heart, Large Language Models generate text through a sequential process of predicting the next "token"—which can be a word, part of a word, or a character—based on the sequence of tokens that have come before it, including the initial prompt and any tokens already generated by the model.[6] This predictive capability is not deterministic but fundamentally probabilistic. During each step of the generation process, the LLM calculates a probability distribution across its entire vocabulary, assigning a likelihood score to every possible token that could follow the current sequence. The token ultimately chosen to extend the sequence is typically sampled from this distribution, meaning that even for the same input sequence, different tokens might be selected in different generation runs if any randomness is permitted in the selection process. This probabilistic foundation is a core reason why LLM outputs can vary, as it allows for a spectrum of possible continuations rather than a single fixed response.

### B. Decoding Strategies: Steering the Probabilistic Output

Several decoding strategies are employed to influence how a token is selected from the probability distribution, allowing users to steer the LLM's output towards desired characteristics like creativity or coherence. These strategies directly impact the inherent variability of the generated text.

**Temperature:** This is a crucial hyperparameter that modulates the shape of the probability distribution before sampling.[3] Lower temperature values (e.g., approaching

0) make the distribution "sharper," meaning the model becomes more confident and assigns significantly higher probabilities to the most likely tokens. This leads to outputs that are more focused, appear more deterministic, and are often more repetitive.[4] Conversely, higher temperature values "flatten" the distribution, making the probabilities of different tokens more similar. This increases randomness and diversity in the output, encouraging more creative or unexpected responses, but potentially at the cost of coherence or factual accuracy.[6] Even when temperature is set very low to promote consistency, the underlying probabilistic nature is not entirely eliminated, particularly if multiple tokens possess very similar high probabilities.

**Top-k Sampling:** This strategy limits the sampling pool to the k most probable tokens.[6] By considering only these top candidates, it ensures that the selected token is one of the more likely ones according to the model. While this can improve coherence, if k is too small, it might lead to generic or repetitive text; if too large or not well-tuned, it might still allow less relevant tokens.

**Top-p (Nucleus) Sampling:** This method, also known as nucleus sampling, selects the smallest set of tokens whose cumulative probability exceeds a predefined threshold p.[6] Unlike top-k, top-p dynamically adapts the size of the sampling pool based on the model's confidence at each step. If the model is very certain about the next token (i.e., one token has a very high probability), the nucleus will be small. If the model is uncertain (many tokens have similar probabilities), the nucleus will be larger. Top-p aims to strike a balance between output quality and diversity. However, particularly at higher temperatures, top-p sampling can still permit lower-probability tokens into the sampling pool, potentially leading to incoherent outputs.[6]

**Min-p Sampling:** A more recent development, min-p sampling, is a dynamic truncation method that adjusts the sampling threshold based on the probability of the top-ranked token. The intuition is that token truncation thresholds should be relative to the model's confidence for that specific token, not absolute. When the model is highly confident, min-p restricts the pool to high-probability candidates to maintain coherence. When less confident, it relaxes the sampling pool to allow for more creative and diverse generation.[6] This method has shown promise in improving both the quality and diversity of generated text compared to traditional methods, especially in high-temperature settings, and has been adopted by several LLM frameworks.[6]

The choice and configuration of these decoding strategies are primary determinants of the output's inherent variability for any single, immediate generation attempt. The interplay between these parameters is also complex; for instance, a high temperature setting increases randomness, which top-p sampling then attempts to constrain.

Achieving a desired balance between consistency and output quality often requires careful tuning of these interacting parameters.[6] Users must recognize that striving for zero variation by manipulating these settings often conflicts with the fundamental design principles of many LLMs, which are built to offer a range of plausible outputs.

### C. Stochastic Embeddings and Dynamic Representations

Beyond the decoding process, some advanced LLM architectures may incorporate stochasticity at an even more fundamental level: the token embeddings themselves. Stochastic Concept Embedding Transitions (SCET), for example, propose a probabilistic mechanism where embeddings can transition between different states during the inference process.[10] This approach allows token representations to adjust dynamically, mitigating constraints imposed by static or deterministic embeddings. Such dynamic representations can enhance the model's adaptability to diverse contexts, improve lexical diversity, and lead to more varied sentence structures by reducing over-reliance on high-probability token selections.[10] While this can improve overall generation quality and contextual flexibility, it introduces another layer of controlled randomness at the representation level, which can contribute to variations in the final output.

## III. Unpacking Non-Determinism: Why "Identical Settings" Don't Guarantee Identical Outputs

Even when users attempt to enforce deterministic behavior by, for example, setting the temperature parameter to zero, LLM outputs can still exhibit variability across repeated runs with identical prompts. This phenomenon arises from a confluence of factors related to hardware, software, and model architecture that operate at a level often invisible to the end-user, particularly when interacting with LLMs via APIs. The common advice to set temperature to 0 for deterministic output is an oversimplification; it aims for greedy decoding (always picking the most probable next token), but the process of determining that "greediest" choice is itself subject to subtle variations.

### A. The Limits of temperature=0 (Greedy Decoding)

Setting temperature=0 instructs the LLM to select the token with the absolute highest probability at each step of the generation process. While this significantly reduces randomness, it does not guarantee identical outputs.[3]

**Probability Ties:** In some situations, two or more tokens might have practically the same highest probability. In such cases, the model or the underlying decoding library

might break these ties in an arbitrary fashion, or the choice might be swayed by minuscule, otherwise insignificant, computational differences from run to run.[5]

**Upstream Variability:** More fundamentally, randomness or slight numerical discrepancies can be introduced *before* the greedy selection stage. The computation of the probability distribution for the next token, which involves a vast number of calculations, might not be perfectly deterministic. Even imperceptibly small differences in these computed probabilities, arising from factors discussed below, can alter which token is ultimately identified as the "most probable," thereby changing the course of generation.[5]

**B. Hardware-Level Influences**

The physical hardware on which LLMs run, particularly Graphics Processing Units (GPUs), introduces sources of non-determinism.

**GPU Parallelism and Execution Order:** GPUs achieve their speed by executing a massive number of operations in parallel across many threads.[12] The precise order in which these parallel operations complete can vary slightly between runs, even with the same input and settings. This non-deterministic execution order can affect operations that require synchronization, such as summing up values computed by different threads.[12]

**Floating-Point Arithmetic:** Neural networks, including LLMs, perform an immense volume of calculations using floating-point numbers. These numbers have limited precision, and operations on them can accumulate tiny rounding errors.[5] Crucially, floating-point arithmetic is generally not associative; for example, due to rounding, (a+b)+c is not guaranteed to be exactly equal to a+(b+c).[12] These slight differences, arising from the non-associativity and rounding of floating-point operations, can compound over the billions of calculations performed in an LLM. The specific precision used (e.g., FP32, FP16, BF16) also impacts reproducibility, with lower-precision formats like FP16 or BFLOAT16 being more prone to rounding errors and thus greater potential variability compared to 32-bit floats.[5]

**Hardware Variation:** The exact behavior of floating-point operations can even differ subtly across different GPU models or different physical servers, even if they are nominally of the same type.[5] If an LLM service provider uses a mix of hardware, or if a user's request is routed to different hardware on different occasions, this can contribute to output variations.[13]

These hardware-level factors mean that even if the software and model weights are

identical, the sequence of computations might not be bit-for-bit reproducible, leading to micro-variations. In LLMs, where each generated token depends on the previously generated ones, such tiny divergences can have a "butterfly effect": a minuscule difference in an internal activation value early in the generation, if it causes even a single different token to be chosen, will alter the entire subsequent generation trajectory. This makes pinpointing the exact source of variation exceedingly difficult.

### C. Architectural Contributions to Non-Determinism

The architectural design of some advanced LLMs can also inherently contribute to non-determinism, especially for users interacting with these models through APIs.

**Mixture-of-Experts (MoE) Models:** Many cutting-edge LLMs, including those powering popular APIs, utilize a Mixture-of-Experts (MoE) architecture.[5] In MoE models, the full network is composed of many smaller "expert" sub-networks. For each input token, a routing mechanism selects a small subset of these experts to process it. This design allows models to scale to much larger parameter counts efficiently. However, the routing decisions can be dynamic and depend on other inputs present in the same processing batch.[5] If a user submits the exact same prompt twice, but it gets grouped with different sets of other user prompts in the processing batch each time (which is common in shared API services), the routing decisions for that specific prompt's tokens could differ. This means different experts might process parts of the prompt, leading to variations in the output.[12] Consequently, an MoE model might be deterministic at the batch level (if the entire batch is identical) but not at the single-request level from an API consumer's perspective.[5] This is a significant source of variability for API users of MoE models, largely uncontrollable from their end.

### D. Software and Implementation Nuances

Subtleties in the software environment and code implementation further contribute to potential non-determinism.

**Library Versions:** Different versions of core deep learning libraries (e.g., CUDA, PyTorch, TensorFlow) or other dependencies can have minor implementation differences that affect numerical computations or the order of operations, potentially leading to varied outputs.[12]

**Low-Level Code Execution Order:** Even seemingly trivial changes in how code is structured or compiled by the underlying system—for instance, the order of operations like f(a + b) versus c = a + b; f(c)—can sometimes alter the sequence of floating-point operations enough to cause divergent results due to the accumulation

of rounding errors.[12]

**Seed Parameters:** Some LLM providers or libraries offer a seed parameter, which is intended to initialize random number generators to a fixed state, thereby promoting reproducibility.[5] While using a seed can help, its effectiveness can be limited by the more fundamental hardware-level non-determinism and, particularly in distributed MoE systems, by dynamic batching and routing effects that are not governed by a single seed.[11] Studies have shown instability even when a fixed seed is used in conjunction with temperature=0.[3]

In summary, the "deterministic illusion" often associated with settings like temperature=0 is frequently punctured by a cascade of these low-level, often imperceptible, factors. True bit-for-bit reproducibility across different runs, especially over time or on different systems, remains an exceptionally challenging goal for today's large, distributed LLMs.

## IV. The Time Factor: Why LLM Responses Diverge with Repeated Prompts Later

When the same prompt is submitted to an LLM at significantly different points in time, the likelihood of output variation increases substantially. This long-term drift is primarily due to the dynamic nature of LLM services: the models themselves evolve, and the environments in which they operate are subject to change. Commercially available LLMs, particularly those accessed via APIs, should be considered parts of "living systems" that are continuously updated, optimized, and sometimes silently altered.

### A. Model Evolution and Updates

LLM providers are constantly working to improve their models, which leads to periodic updates that can alter output behavior.

**Retraining and Fine-tuning:** Models are frequently retrained on new, larger, or more diverse datasets to expand their knowledge and capabilities.[5] They also undergo fine-tuning to enhance performance on specific tasks, improve alignment with desired behaviors (e.g., helpfulness, harmlessness, factual accuracy), or introduce new functionalities.[5] For instance, techniques like Chain of Guidance (CoG) may be employed to fine-tune models specifically for better semantic consistency, which, while aiming to reduce unwanted variation for paraphrased inputs, is itself a model update that changes the model's response patterns.[14] These updates inevitably modify the model's internal weights and biases. As the model's "understanding" of

language and its knowledge base shift, the probability distributions it generates for next-token predictions will change, leading to different responses over time for the exact same prompt.

**Architectural Changes:** Although less common than data or fine-tuning updates, the underlying architecture of an LLM might be modified by the provider.[5] This could involve changes to transformer block designs, the number of layers or attention heads, or other core components. Such fundamental architectural shifts can significantly impact the model's generation process and, consequently, its outputs.

### B. API and Operational Environment Shifts

Changes in the Application Programming Interface (API) or the backend infrastructure supporting the LLM service are another major source of temporal variation.

**API Specification Changes:** Providers may update their APIs by altering request or response formats, adding or removing parameters, or changing the default values of existing parameters (e.g., the default temperature setting).[5] If a user relies on default settings, such changes can unknowingly alter the generation behavior.

**Backend Infrastructure Modifications:** Providers continuously optimize their backend infrastructure for efficiency, scalability, reliability, or cost reduction. These optimizations can involve:

- **Hardware Upgrades/Changes:** Switching to newer generations of GPUs or other specialized hardware.[5] As discussed in Section III.B, different hardware can have slightly different floating-point behavior.
- **Software Library Updates:** Updating underlying software libraries (e.g., CUDA, deep learning frameworks) to newer versions.[5]
- **Load Balancing and Request Batching:** Modifying how user requests are distributed across servers or how they are batched for processing.[5] Changes here can affect MoE routing or expose prompts to different micro-environmental conditions. These backend changes can reintroduce or alter the non-deterministic behaviors discussed previously, even if the core model weights themselves have not been updated.

### C. The Specter of Covert Model Substitution

A more opaque reason for output variation over time is the possibility of covert model substitution by API providers.[18] This can occur for several reasons:

- **Cost Reduction:** A provider might silently switch from an advertised, expensive model to a smaller, cheaper, or more heavily quantized (e.g., INT8 instead of FP16)

version to reduce operational costs.
- **Load Balancing or A/B Testing:** Requests might be rerouted to a different model variant if the primary model is under heavy load, or as part of unannounced A/B testing of new model versions.

Such substitutions directly lead to output variation because the underlying generative engine processing the prompt is different. This practice, when undisclosed, erodes user trust, hinders reproducibility (especially in research), and can invalidate benchmarks or performance evaluations conducted via the API.[18] Detecting these substitutions is challenging for users of black-box APIs, although research is ongoing to develop auditing methods.[18] This lack of transparency makes it difficult for users to diagnose the cause of output variation and distinguish between benign updates and potentially problematic regressions in service quality.

**D. Changes to the "Specified Corpus" (if applicable and dynamic)**

The user's query refers to an LLM examining a "specified corpus." If this corpus is not static but is subject to updates—such as the addition of new documents, revisions to existing ones, or deletions—then an LLM employing a RAG approach will naturally produce different answers over time. This occurs because the knowledge source from which the LLM draws information has changed. This type of variation is distinct from the unintended variability of the LLM system itself and is often a desired behavior, reflecting the system's ability to incorporate new information. However, it is a crucial factor to consider when analyzing why outputs change over time in a corpus-based Q&A scenario.

The various changes—model updates, API modifications, infrastructure shifts, and potential model substitutions—often interact. For instance, a major model update might necessitate changes to the backend hardware, and both could contribute to observed output shifts. Therefore, users should generally anticipate that an LLM API endpoint represents a "moving target," and the output for a given prompt is likely to evolve over extended periods.

The following table summarizes the principal sources of LLM output variation discussed:

**Table 1: Principal Sources of LLM Output Variation**

| Category of Variation Source | Specific Factor | Brief Description of Impact on Output | Primarily Affects |
|---|---|---|---|

| | | **Variability** | |
|---|---|---|---|
| **Foundational Generation Process** | Probabilistic Token Selection | Introduces randomness in token choice based on probabilities assigned by the model. | Single Run Stochasticity |
| | Decoding Strategies (Temp, Top-k/p, Min-p) | Modulate how tokens are selected from the probability distribution, affecting randomness/diversity. | Single Run Stochasticity |
| | Stochastic Embeddings | Allows token representations to change dynamically during inference, adding representational variability. | Single Run Stochasticity |
| **Hardware/Low-Level Software** | GPU Parallelism & Non-Deterministic Execution Order | Order of parallel operations can vary, affecting results of synchronized computations. | Run-to-Run Non-Determinism |
| | Floating-Point Non-Associativity & Rounding | Accumulating micro-errors from inexact arithmetic can alter internal states and final token probabilities. | Run-to-Run Non-Determinism |
| | Hardware/Precision Variation | Different GPUs or numerical precisions (FP16 vs FP32) can yield slightly different computational results. | Run-to-Run Non-Determinism |
| | Software | Different versions or | Run-to-Run |

| | | | |
|---|---|---|---|
| | Library/Implementation Differences | minor code changes can alter numerical outcomes. | Non-Determinism |
| **LLM Architecture** | Mixture-of-Experts (MoE) Routing | Dynamic, batch-dependent routing can lead to different experts processing tokens for the same prompt. | Run-to-Run Non-Determinism (API) |
| **Temporal: Model Updates** | Model Retraining/Fine-tuning | Changes model's knowledge, biases, and internal representations, altering output probabilities. | Long-Term Drift |
| | Architectural Model Changes | Fundamental changes to model structure impact generation process. | Long-Term Drift |
| **Temporal: API/Infrastructure** | API Specification/Default Changes | Alters generation parameters if not explicitly overridden by the user, or changes data formats. | Long-Term Drift |
| | Backend Infrastructure/Hardware Updates | Can reintroduce or modify low-level non-deterministic behaviors (e.g., new GPUs, different batching). | Long-Term Drift |
| | Covert Model Substitution | Use of an entirely different (e.g., smaller, quantized) model than advertised. | Long-Term Drift |

| | | | |
|---|---|---|---|
| **Retrieval Augmented Generation (RAG) Process** | Dynamic Corpus | If the underlying knowledge corpus is updated, retrieved context and thus LLM output will change. | RAG Context / Long-Term Drift (Intended) |
| | Variability in Retrieval Mechanism | Changes or inherent non-determinism in chunking, embedding, or similarity search can alter retrieved context. | RAG Context / Run-to-Run or Long-Term |

## V. The "Specified Corpus" Context: Retrieval Augmented Generation (RAG) and Its Impact on Consistency

When an LLM is tasked with examining a "specified corpus of relevant documents and data," it often involves a Retrieval Augmented Generation (RAG) system. RAG architectures are designed to enhance the factual grounding, relevance, and verifiability of LLM responses by dynamically incorporating information from an external knowledge source. While RAG aims to improve overall response quality and consistency with respect to the provided corpus, the retrieval process itself can introduce its own sources of variability, thereby influencing the final LLM output.

### A. RAG's Primary Goal: Enhancing Factual Grounding and Relevance

The core purpose of RAG is to mitigate issues like "hallucinations" (generating plausible but incorrect or unverified information) and to ensure that LLM outputs are based on specific, attributable information from the designated corpus.[1] By retrieving relevant text passages from the corpus and providing them as augmented context to the LLM alongside the user's prompt, RAG systems aim to produce answers that are more accurate, up-to-date (if the corpus is current), and directly supported by the provided documents.[1] This grounding can significantly enhance consistency, particularly in ensuring that the LLM's statements align with the information contained within the specified corpus. For instance, in medical applications, RAG can help LLMs provide preoperative instructions consistent with established guidelines by retrieving and using those guidelines during generation.[20]

### B. Sources of Variability Introduced by the RAG Process Itself

Despite its benefits, the RAG pipeline consists of several components, each of which can contribute to variability in the information retrieved and, consequently, in the LLM's final output. The consistency of the LLM's response becomes heavily dependent on the consistency of this retrieval mechanism.

**Document Chunking:** Before documents can be efficiently searched, they are typically segmented into smaller, manageable pieces or "chunks".[20] The strategy used for chunking (e.g., fixed-size blocks, sentence-based splitting, semantic chunking aware of content boundaries) can significantly affect what contextual information is grouped together and subsequently retrieved. If the chunking strategy is changed over time (e.g., due to updates in the RAG library or a re-indexing process with new parameters), or if the strategy itself has borderline cases where a piece of text could be split in multiple ways, different sets of chunks might be retrieved for the same query at different times.

**Embedding Models:** To enable semantic search, both the query and the document chunks are converted into numerical vector representations (embeddings) using an embedding model.[2] The choice of embedding model (e.g., Sentence-BERT, OpenAI Ada, etc.) and its specific version are critical. If this model is updated or changed, the resulting embeddings will differ, leading to changes in how semantic similarity is calculated between the query and the document chunks. This, in turn, will alter which documents are deemed most relevant and retrieved.[20] Even the same embedding model might exhibit slight non-deterministic variations in its output embeddings, though typically less pronounced than in large generative LLMs.

**Similarity Search Dynamics:** Once embeddings are generated, a similarity search algorithm (commonly cosine similarity or dot product) is used to find and rank document chunks that are most similar to the query embedding.[2] Parameters such as the number of top chunks to retrieve (top_k) or a minimum similarity threshold can influence the result set. Changes to these parameters, updates to the vector database or search index implementation, or even subtle numerical fluctuations in similarity scores (especially if multiple chunks have very close scores to the query) can lead to variations in the retrieved set of chunks.[20] For example, if the system is configured to retrieve the top 5 chunks, and the 5th and 6th most similar chunks have almost identical scores, minor computational differences could swap their order between runs.

**Updates to the Corpus:** As mentioned in Section IV.D, if the "specified corpus" itself is dynamic—meaning documents are added, revised, or deleted over time—the RAG system will naturally retrieve different information for the same prompt submitted at

different times. This is often an intended and desirable form of variation, as it reflects the system's ability to access and use the most current knowledge. However, it's crucial to distinguish this intended variation due to corpus evolution from unintended system variability when the corpus is static.

**Retrieval Strategy Complexity:** Modern RAG systems can employ sophisticated retrieval strategies beyond simple top_k search. These might include multi-hop retrieval (where the initial query is decomposed, or intermediate results are used to formulate new queries), query reformulation, or synthesizing evidence from multiple retrieved chunks before passing it to the LLM.[1] Each additional step in these complex retrieval chains (e.g., an LLM call to reformulate a query, or to summarize retrieved chunks) introduces further potential points for slight variations to occur, which can then propagate and affect the final set of context provided to the main generator LLM. Advanced frameworks like Retroactive Retrieval-Augmented Generation (RetroRAG) aim to improve robustness by allowing the system to revise and update evidence, potentially correcting earlier retrieval errors, but the iterative nature itself can be a source of subtle differences.[1]

### C. Interaction Between Retrieved Context and LLM Generation

The final output generated by the LLM in a RAG system is highly conditioned on the specific context provided by the retrieval module. If the set of retrieved document chunks varies—even slightly in content or order—the LLM will likely generate a different response. This interaction can amplify small variabilities originating from the retrieval stage. For instance, if one run retrieves a chunk emphasizing a particular risk associated with an alternative, while another run (due to slight retrieval differences) retrieves a chunk downplaying that risk or highlighting a benefit, the LLM's subsequent analysis and conclusion regarding that alternative could diverge significantly. The LLM's tendency to generate text that sounds coherent and confident, even if based on slightly incomplete or skewed retrieved information, can make it challenging to trace the origin of the variation back to the retrieval step. This underscores the importance of high-quality and consistent retrieval in RAG systems.

## VI. Characterizing the Expected Variation: Nature, Extent, and Influencing Factors

The variation in LLM outputs, when a prompt is repeated over time, is not a monolithic phenomenon. It can manifest in diverse ways, ranging from inconsequential alterations to significant shifts that impact the utility and trustworthiness of the response. Understanding this spectrum of variation, its documented extent, and the

factors that influence it is crucial for users relying on LLMs for complex analyses.

### A. Spectrum of Variation

The nature of output variation can be categorized as follows:

- **Minor Phrasing Differences:** These include changes in word choice, sentence structure, or stylistic tone that do not alter the core meaning, factual content, or overall conclusion of the answer.[17] For example, "The analysis indicates X" versus "X is suggested by the data." Such variations are often the most common and may be acceptable in many contexts.
- **Selection of Different Supporting Evidence (in RAG systems):** If the underlying corpus contains multiple pieces of information that could support an answer, the LLM (via the RAG process) might select or emphasize different subsets of this evidence across different runs. This is particularly likely if several retrieved chunks have similar relevance scores to the query. The overall conclusion might remain the same, but the justification path may differ.
- **Shifts in Interpretation or Emphasis:** The LLM might interpret nuances in the prompt or the retrieved data differently across runs, leading to answers that emphasize different aspects of a situation or draw slightly different inferences or subjective judgments.[4] This can be more problematic if the interpretation affects the recommended course of action or the assessment of outcomes.
- **Changes in Output Format/Structure:** The LLM might produce outputs with inconsistent formatting (e.g., using bullet points in one instance and numbered lists in another, or varying JSON structures if structured output is requested). Such instability can cause failures in downstream automated parsing systems.[3]
- **Significant Shifts in Factual Claims or Conclusions:** This is the most concerning type of variation, where the LLM provides contradictory factual information, makes different assertions about alternatives, or reaches substantially different overall conclusions or recommendations for the complex situation being analyzed.[3] This directly impacts the reliability and trustworthiness of the LLM for decision support.

### B. Documented Instability and Idiosyncrasies

Research has documented the prevalence and nature of these variations:

- Studies have shown that LLMs can exhibit significant instability even when configured for deterministic output (e.g., temperature=0). One investigation found accuracy variations of up to 15% across runs on common tasks, with none of the tested LLMs consistently delivering repeatable accuracy, let alone identical output strings.[3]

- In tasks requiring specific outcomes, such as legal question answering, leading LLMs have been observed to be unstable, sometimes returning opposite conclusions for the same legal question even with temperature=0 and a fixed seed. Instability rates varied by model, with one model being unstable on over 50% of questions in a dataset.[11]
- LLMs possess "idiosyncrasies"—unique patterns in their outputs (e.g., preferred phrasing, stylistic tendencies) that can be used to distinguish the source model with high accuracy.[24] This suggests that while models have consistent individual "styles," these styles still allow for internal variation. Variation is also observed within the same model family across different model sizes (e.g., 7B vs. 72B parameters).[24]
- LLMs can express inconsistent ideas or preferences when faced with trivial changes in prompt formatting or context, sometimes leading to response variations that exceed real-world cultural differences in evaluation studies.[26] This highlights a certain "fragility" in how LLMs process and respond to inputs, making consistency challenging if input presentation is not meticulously controlled.

### C. Impact on Downstream Systems and Reliability

Output variability can have cascading negative effects on systems that rely on LLMs:

- It can lead to inexplicable errors that are very different from human mistakes, making debugging and quality assurance difficult.[3]
- In systems where multiple LLM calls are chained (e.g., a conversational agent using LLM classifiers for dialogue state management), the instability of each component can compound, leading to a multiplicative reduction in overall system performance and reliability. For example, a system with four LLM components, each 95% stable in producing the desired type of output, would have an overall expected success rate of only $0.95^4 \approx 0.81$ before even considering the accuracy of each component on novel inputs.[3]
- The lack of stable outputs undermines confidence in traditional software testing methodologies, especially for less frequent (long-tail) inputs where errors might be more costly or harder to detect during development.[3] The concept of a simple "unit test" for an AI function is challenged by inherent non-determinism.[3]

### D. Factors Modulating the Degree of Variation

The extent and nature of output variation are influenced by several factors:

- **Model Characteristics:** Different LLMs (e.g., GPT-4o, Claude-3.5, Gemini-1.5, Llama 3) exhibit varying degrees of stability, with some being inherently more prone to variation than others even under similar settings.[3] Model size,

architecture (e.g., dense vs. MoE), and training data all play a role.[24]

- **Prompt Design and Sensitivity:**
  - Ambiguity, subjectivity, or lack of specificity in prompts provides the LLM with more interpretive freedom, which can lead to greater output variation.[4]
  - As noted, LLMs can be highly sensitive to small syntactic rephrasings or formatting changes in the prompt, which can drastically shift the output, particularly if the temperature setting allows for more randomness.[4]
- **Task Complexity:** More complex tasks, such as those requiring multi-hop reasoning, abstract analysis, or dealing with "hard legal questions," are likely to exhibit more variability in LLM responses compared to simple factual retrieval or straightforward summarization.[1]
- **Corpus Quality and Ambiguity (for RAG systems):** If the specified corpus used by a RAG system contains conflicting, ambiguous, or incomplete information related to the query, the LLM's output may vary depending on which specific pieces of information are retrieved and how the LLM resolves these inconsistencies or gaps.

Understanding these characteristics of variation is essential. It moves the discussion from a general concern about variability to a more specific understanding of potential failure modes and areas requiring focused attention for mitigation and risk assessment. The following table helps categorize these manifestations:

**Table 2: Characterizing Types of LLM Output Variation and Potential Impact**

| Type of Variation | Description | Typical Cause(s) | Potential Impact Level (Low/Medium/High) |
|---|---|---|---|
| **Minor Phrasing/Stylistic** | Different wording, sentence structure, or tone, but the core meaning and factual content remain unchanged. | Inherent stochasticity (e.g., temperature > 0.1), subtle non-determinism, model's stylistic preferences. | **Low**, if core meaning is preserved and style is acceptable. |
| **Output Formatting** | Changes in the structure of the output, e.g., use of lists vs. paragraphs, JSON key variations, spacing. | Low-level non-determinism, minor model updates, API changes affecting default formatting. | **Medium to High**, if it breaks downstream parsing logic or automated processing. Low if only visual. |

| | | | |
|---|---|---|---|
| **Evidence Selection (RAG)** | Different relevant snippets from the corpus are retrieved or cited as support, while the conclusion is similar. | RAG retrieval dynamics (e.g., similar scores for multiple chunks), minor query variations, corpus ambiguity. | **Medium**, if all cited evidence is valid but different. High if critical evidence is missed or irrelevant evidence is included. |
| **Interpretation/Emphasis** | Different nuances are highlighted, slightly different inferences are drawn, or subjective aspects vary. | Prompt ambiguity, model biases, inherent stochasticity allowing different reasoning paths, corpus ambiguity. | **Medium to High**, depending on the sensitivity of the interpretation. Can be critical if it leads to different strategic decisions. |
| **Significant Factual Accuracy/Conclusion** | Contradictory facts are presented, different alternatives are favored, or substantially different outcomes are proposed. | Major model updates, critical retrieval errors in RAG, severe non-determinism, significant prompt misinterpretation. | **High**, represents a critical failure in reliability and trustworthiness, potentially leading to incorrect decisions. |

## VII. Strategies for Navigating and Potentially Mitigating Output Variability

While perfect output consistency from LLMs, especially over time and via APIs, is often an elusive goal, users and developers can employ various strategies to understand, manage, and potentially reduce unwanted variability. These approaches range from tuning model parameters and careful system design to acknowledging inherent limitations and building robust applications around them. A combination of proactive measures (taken during design and prompting) and reactive measures (such as output validation and monitoring) is typically most effective.

### A. User-Controlled Parameters (with Caveats)

Users often have access to certain parameters that can influence the LLM's generation process:

- **Temperature:** Setting the temperature to very low values (e.g., 0.0 to 0.3) is the most direct method to reduce the randomness in token selection, making the model favor the most probable outputs.[4] For tasks requiring precision and

consistency, such as generating compliance documentation or financial reports, a low temperature is generally recommended.[8]
    - *Effectiveness and Caveats:* While low temperature significantly reduces one source of stochasticity, it does not eliminate variability stemming from other non-deterministic factors like hardware-level effects or MoE routing, as detailed in Section III.[3] Outputs can also become more repetitive or "boring" at very low temperatures.[8]
- **Seed Parameters:** Some LLM APIs or client libraries provide a seed parameter, which aims to initialize the internal random number generators to a fixed state.[5]
    - *Effectiveness and Caveats:* Using a consistent seed can improve reproducibility for a specific model version running on a consistent hardware setup, provided all other sources of non-determinism are also controlled. However, its effectiveness is often limited for API-based models due to distributed serving architectures, dynamic batching in MoE models, and unannounced backend changes.[5] Studies have reported observing instability even when using a fixed seed with temperature=0.[11]
- **Other Sampling Parameters (Top-k, Top-p):** Parameters like top-k and top-p (nucleus) sampling can be used in conjunction with temperature to further constrain the model's choices.[6] While their primary role is often to balance output quality and diversity, they can contribute to consistency by limiting the pool of potential next tokens. For instance, top-p sampling combined with temperature can prevent the selection of extremely low-probability words, which can be helpful when higher temperatures increase randomness.[8] Min-p sampling is a newer alternative designed to dynamically adjust truncation based on model confidence, potentially offering better quality and diversity control.[6]
    - *Effectiveness and Caveats:* These are generally less about enforcing strict determinism and more about shaping the characteristics of a (still potentially variable) output. Their interaction with temperature needs careful consideration.[6]

It is important to recognize that while users can control these input parameters, many significant sources of variability, particularly those related to the provider's infrastructure and model update cycles, remain outside their direct influence when using commercial LLM APIs.

### B. Prompt Engineering for Consistency

The way a prompt is formulated can significantly influence the consistency of LLM responses:

- **Precision and Lack of Ambiguity:** Crafting prompts that are highly specific,

unambiguous, and provide clear, detailed instructions can reduce the LLM's "interpretive freedom." This narrows the range of plausible responses and can thus decrease variability.[9]

- **Consistent Formatting and Input:** Ensuring that prompts are *exactly* identical across repeated calls is critical. This includes meticulous attention to whitespace, punctuation, casing, and any hidden characters, as even minor differences can lead to divergent outputs.[5]
- **Few-Shot Examples:** Providing consistent, high-quality examples of the desired input-output behavior within the prompt (few-shot prompting) can effectively guide the LLM towards a specific output style, format, and level of detail, thereby enhancing consistency.
- **Structured Input/Output:** When possible, using structured input formats and requesting structured output (e.g., JSON) can help constrain the LLM's responses and make variations easier to detect and manage, although the structure itself can sometimes vary.

### C. System-Level Approaches

Beyond direct interaction parameters, broader system-level design choices can help manage variability:

- **Retrieval Augmented Generation (RAG) Design for Consistency:**
  - Ensure high-quality, consistent retrieval from the specified corpus by using robust embedding models and stable vector indexing and search mechanisms.
  - Implement version control for the corpus itself, as well as for all components of the RAG pipeline (e.g., chunking scripts, embedding models). This helps in tracing sources of variation if outputs change after an update.
  - Consider advanced RAG techniques like RetroRAG, which incorporate mechanisms for revising evidence and reasoning, potentially correcting for inconsistencies arising during the retrieval or initial generation phases.[1]
- **Output Validation and Filtering:** Implement automated checks on the LLM's output. These checks can verify adherence to expected formats, factual consistency against known constraints or the retrieved corpus data, or alignment with a set of predefined quality criteria. Outputs that fail these checks can be flagged or re-processed.
- **Regression Testing with Tolerance:** Traditional unit tests that expect exact output matches are often unsuitable for LLM-generated content due to inherent non-determinism. Instead, regression testing frameworks should be designed to tolerate a certain degree of acceptable variability (e.g., semantic similarity rather

than lexical identity) while still catching significant deviations or regressions in quality or accuracy.[3]
- **Monitoring and Iteration:** Continuously monitor the performance of LLM-generated content against key metrics such as coherence, accuracy, relevance, and user satisfaction.[8] Track output behavior over time to detect drift. Use feedback from monitoring and A/B testing to identify and address inconsistencies, potentially by refining prompts, adjusting parameters, or re-evaluating the LLM or RAG components.[8]

### D. Acknowledging and Designing for Inherent Variability

In many practical applications, achieving perfect, bit-for-bit consistency may be unattainable or economically infeasible. The "cost" of pushing for extremely high levels of consistency might involve trade-offs such as reduced output creativity (if that is desired for some aspects), significant engineering effort for complex validation systems, or reliance on potentially less advanced or slower models/APIs that might offer stronger (but still not absolute) deterministic guarantees.

Therefore, a pragmatic approach involves:

- **Focusing on "Stability":** As defined in some research, stability refers to the LLM's tendency to produce the *same conclusion* or semantically equivalent output, even if the exact phrasing or supporting details differ slightly.[11] For many analytical tasks, this semantic stability is more critical than lexical identity.
- **Designing Downstream Processes for Robustness:** Applications that consume LLM outputs should be designed to be robust to minor variations. This might involve using flexible parsers, employing semantic similarity checks instead of exact string matching for comparisons, or having error handling and fallback mechanisms.
- **Setting Realistic Expectations:** It is crucial to communicate clearly with stakeholders about the inherent nature of LLM output variability and the level of consistency that can realistically be expected for the specific use case and chosen technology stack.

By combining these strategies, users can better navigate the challenges posed by LLM output variability, enhancing the reliability and trustworthiness of LLM-powered analyses even in the face of inherent non-determinism and temporal changes.

## VIII. Conclusion: Embracing Realistic Expectations for LLM Output Consistency

The analysis of Large Language Model (LLM) behavior, particularly when applied to complex situations using a specified corpus of documents, reveals that output variation upon repeated prompting over time is not an anomaly but an inherent characteristic shaped by multiple factors. Achieving absolute, bit-for-bit identical output across different invocations, especially with API-based models and over extended periods, is generally an unrealistic expectation.[3]

### A. Recapitulation of Key Drivers of Variability

The variability in LLM responses stems from a confluence of sources:

1. **Foundational LLM Stochasticity:** The probabilistic nature of next-token prediction and the influence of decoding strategies (like temperature and sampling methods) introduce a baseline level of potential variation even in a single session.[6]
2. **Low-Level Non-Determinism:** Hardware-level factors (GPU parallelism, floating-point arithmetic idiosyncrasies) and subtle software implementation details can lead to micro-variations in computation, which can accumulate and result in different outputs even when parameters like temperature are set to zero.[5] Architectural choices like Mixture-of-Experts (MoE) further contribute to this, especially for API users, due to dynamic, batch-dependent routing.[5]
3. **Temporal Changes:** LLM providers continuously update their models (retraining, fine-tuning) and the underlying API infrastructure (hardware, software, default parameters). Covert model substitution is also a potential, albeit opaque, factor.[5] These changes mean that an LLM service is a "moving target" over time.
4. **RAG-Specific Factors:** When a specified corpus is used via Retrieval Augmented Generation, the retrieval process itself (chunking, embedding, similarity search) can introduce variability. Furthermore, if the corpus itself is dynamic, this will naturally lead to changed outputs, which is often an intended behavior.[1]

The degree of variation can span a wide spectrum, from minor, inconsequential phrasing differences to significant shifts in factual claims, interpretations, or recommended outcomes.[3]

### B. The Inevitability of Some Output Variation

Given these pervasive factors, some level of output variation is inevitable. The practical goal for most applications should therefore shift from seeking perfect determinism to understanding, characterizing, and managing what might be termed "bounded instability." This involves ensuring that while outputs may vary, they remain within acceptable semantic, factual, and structural boundaries that are defined as

appropriate for the specific task and its risk profile.

### C. Recommendations for Users in Complex, Corpus-Based Analyses

For users relying on LLMs to analyze alternatives and outcomes in complex situations using a specified corpus, the following recommendations are pertinent:

1. **Prioritize Robust Evaluation and Characterization:** Before deploying an LLM for critical tasks, conduct thorough testing with the specific model, RAG setup (if used), and representative prompts. Characterize the typical range and nature of output variation observed. This establishes a baseline understanding of the system's stability.
2. **Define Acceptable Variation Thresholds:** Based on the use case, establish clear criteria for what constitutes an acceptable variation versus a meaningful and problematic change in the LLM's output. This will depend on whether minor phrasing changes are tolerable, but shifts in core conclusions are not.
3. **Implement Comprehensive Monitoring and Adaptation Strategies:** Continuously monitor LLM outputs in production for drift in quality, accuracy, and consistency against the established baselines and acceptance criteria.[8] Be prepared to adapt prompts, validation logic, or even re-evaluate the choice of LLM or RAG components if consistency degrades beyond acceptable levels over time.
4. **Maintain Rigorous Version Control:** Implement version control not only for prompts but also for the specified corpus (if it's meant to be static for a period) and all components of the RAG pipeline (e.g., embedding models, chunking algorithms, retrieval parameters). This aids in isolating the source of changing outputs when variations are detected.
5. **Design Consistency-Aware Applications:** Applications built on top of LLMs should be designed with the explicit assumption that LLM outputs can and will vary. This involves building resilience and adaptability into the application logic itself, such as using flexible parsers for structured data, employing semantic similarity checks where appropriate instead of exact string matches, and incorporating robust error handling and fallback mechanisms.
6. **Communicate Limitations Transparently:** Ensure that all stakeholders, including end-users and decision-makers, have a realistic understanding of the inherent limitations regarding LLM output consistency and the potential for variation.

### D. Future Outlook

The field of LLMs is rapidly evolving. Ongoing research is focused on developing

models and techniques that offer greater controllability and predictability in generation. There is also a growing call for increased transparency from API providers regarding model updates, architectural details, and changes to service infrastructure, which could help users better anticipate and manage output variability. As these models become more deeply integrated into critical workflows, the demand for reliable and consistent behavior will continue to drive innovation in both LLM development and the methodologies for their robust deployment. Ultimately, successfully leveraging LLMs for complex analysis requires a nuanced understanding of their capabilities and limitations, coupled with diligent engineering and realistic expectations.

**Works cited**


1. Retrieval-Augmented Generation by Evidence Retroactivity in LLMs - arXiv, accessed May 17, 2025, https://arxiv.org/html/2501.05475v1
2. Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey - arXiv, accessed May 17, 2025, https://arxiv.org/html/2504.14891v1
3. Non-Determinism of "Deterministic" LLM Settings - arXiv, accessed May 17, 2025, https://arxiv.org/html/2408.04667v3
4. An overview of model uncertainty and variability in LLM-based sentiment analysis. Challenges, mitigation strategies and the role of explainability - arXiv, accessed May 17, 2025, https://arxiv.org/html/2504.04462v1
5. Does Temperature 0 Guarantee Deterministic LLM Outputs ..., accessed May 17, 2025, https://www.vincentschmalbach.com/does-temperature-0-guarantee-deterministic-llm-outputs/
6. MIN-p SAMPLING FOR CREATIVE AND COHERENT LLM OUTPUTS - arXiv, accessed May 17, 2025, https://arxiv.org/pdf/2407.01082?
7. Turning Up the Heat: Min-p Sampling for Creative and Coherent LLM Outputs - arXiv, accessed May 17, 2025, https://arxiv.org/html/2407.01082v4
8. LLM Temperature: A Complete Guide to Understanding This Critical ..., accessed May 17, 2025, https://www.citrusx.ai/posts/llm-temperature-a-complete-guide
9. What is LLM Temperature? - Hopsworks, accessed May 17, 2025, https://www.hopsworks.ai/dictionary/llm-temperature
10. Latent Structure Modulation in Large Language Models Through Stochastic Concept Embedding Transitions - arXiv, accessed May 17, 2025, https://arxiv.org/html/2502.05553v1
11. LLMs Provide Unstable Answers to Legal Questions - arXiv, accessed May 17, 2025, https://arxiv.org/html/2502.05196v1
12. [D] Non-deterministic behavior of LLMs when temperature is 0 : r ..., accessed May 17, 2025, https://www.reddit.com/r/MachineLearning/comments/1ie15ev/d_nondeterministic_behavior_of_llms_when/

13. arxiv.org, accessed May 17, 2025, https://arxiv.org/pdf/2502.05196
14. Improving Consistency in Large Language Models through Chain of Guidance - arXiv, accessed May 17, 2025, https://arxiv.org/html/2502.15924v1
15. Improving Consistency in Large Language Models through Chain of Guidance | OpenReview, accessed May 17, 2025, https://openreview.net/forum?id=asiBW1bB9b
16. arxiv.org, accessed May 17, 2025, https://arxiv.org/pdf/2502.15924
17. Switching LLM Providers: Why It's Harder Than It Seems - Requesty ..., accessed May 17, 2025, https://www.requesty.ai/blog/switching-llm-providers-why-it-s-harder-than-it-seems
18. Are You Getting What You Pay For? Auditing Model Substitution in LLM APIs - arXiv, accessed May 17, 2025, https://arxiv.org/html/2504.04715v1
19. arxiv.org, accessed May 17, 2025, https://arxiv.org/pdf/2504.04715
20. Retrieval augmented generation for 10 large language models and ..., accessed May 17, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11971376/
21. arxiv.org, accessed May 17, 2025, https://arxiv.org/pdf/2501.05475
22. Improving the Reliability of LLMs: Combining Chain-of-Thought Reasoning and Retrieval-Augmented Generation - arXiv, accessed May 17, 2025, https://arxiv.org/html/2505.09031v1
23. arxiv.org, accessed May 17, 2025, https://arxiv.org/pdf/2505.09031
24. arxiv.org, accessed May 17, 2025, https://arxiv.org/html/2502.12150v1
25. Uncovering Gaps in How Humans and LLMs Interpret Subjective Language | OpenReview, accessed May 17, 2025, https://openreview.net/forum?id=gye2U9uNXx
26. Randomness, Not Representation: The Unreliability of Evaluating Cultural Alignment in LLMs - arXiv, accessed May 17, 2025, https://arxiv.org/html/2503.08688v2
27. Randomness, Not Representation: The Unreliability of Evaluating Cultural Alignment in LLMs - arXiv, accessed May 17, 2025, https://arxiv.org/html/2503.08688v1