VeriFact — Expert Interview Preparation Guide

200+ In-Depth Questions & Answers

> Prepared for an expert panel interview on the VeriFact Automated Neural Claim Verification System.

PART A: THEORETICAL FOUNDATIONS

1. Natural Language Inference (NLI)

Q1. What is Natural Language Inference? How does it differ from text classification? NLI determines the logical relationship (entailment, contradiction, neutral) between a premise and a hypothesis. Unlike text classification which assigns a label to a single text, NLI reasons about the relationship *between* two texts — it answers "does text A logically support, contradict, or have no bearing on text B?"

Q2. Explain the three NLI labels. Give an example of each in the context of fact-checking.

- **Entailment:** Evidence supports the claim. Premise: "NASA confirmed Earth is an oblate spheroid." Hypothesis: "The Earth is round." → Entailment.
- **Contradiction:** Evidence refutes the claim. Premise: "WHO declared the pandemic over in 2023." Hypothesis: "The COVID pandemic is still ongoing in 2025." → Contradiction.
- **Neutral:** Evidence is topically related but inconclusive. Premise: "The PM visited France." Hypothesis: "The PM signed a trade deal." → Neutral.

Q3. Why did you choose NLI over training a binary fake/real classifier? Supervised classifiers trained on datasets like LIAR or FakeNewsNet suffer from **dataset obsolescence** — a model trained on 2020 political claims can't detect 2025 science misinformation. NLI models learn *logical reasoning* (entailment vs. contradiction), not topic-specific patterns. This makes them domain-agnostic and future-proof without retraining.

Q4. What is the MNLI (Multi-Genre NLI) corpus? Why is it significant for your model? MNLI contains 433k human-annotated premise-hypothesis pairs spanning 10 genres (fiction, government, travel, etc.). Training on MNLI teaches the model genre-agnostic logical reasoning. DeBERTa-v3-small achieves 88% on MNLI, meaning it understands entailment/contradiction regardless of domain — critical for a general-purpose fact-checker.

Q5. What are the limitations of NLI for fact-checking? Where does it fail? NLI understands *linguistic* relationships, not *factual truth*. It can detect "X contradicts Y" but not "X is factually wrong." It fails on: (1) sarcasm/irony, (2) entity-level facts ("X is Y's son" — high topic overlap confuses NLI), (3) implicit reasoning requiring world knowledge, (4) claims requiring numerical/temporal reasoning.

Q6. What is the difference between zero-shot NLI and direct NLI? Why did you migrate? Zero-shot NLI (e.g., BART-MNLI) repurposes NLI by constructing synthetic hypotheses like "This text is about support." Direct NLI feeds the actual premise-hypothesis pair. Direct NLI is architecturally correct for stance detection because we're literally asking "does this evidence entail/contradict this claim?" Zero-shot adds an abstraction layer that loses semantic precision.

Q7. How does NLI handle negation? Can your model detect "Trump was NOT assassinated"? DeBERTa-v3 handles explicit negation reasonably well through its attention mechanism. However, complex negations (double negatives, implicit negation) can confuse it. This is precisely why I built the **outcome mismatch detection** pre-filter — it catches negating modifiers like "survived," "attempted," "failed" *before* NLI runs.

Q8. What is the difference between textual entailment and paraphrase detection? Paraphrase detection is symmetric — "A paraphrases B" implies "B paraphrases A." Textual entailment is asymmetric — "A entails B" does NOT imply "B entails A." Example: "The cat sat on the mat" entails "Something sat on the mat," but not vice versa. NLI captures this directionality, which is essential for fact-checking.

---

2. Transformer Architecture & Model Selection

Q9. Why DeBERTa-v3 over BERT or RoBERTa for NLI? DeBERTa-v3 uses **disentangled attention** — it separates content and position embeddings, computing attention with both content-to-content AND content-to-position interactions. This better captures positional relationships between words. On MNLI, DeBERTa-v3-small outperforms BERT-base (88% vs 84%) while being smaller. It also uses ELECTRA-style training (replaced token detection) which is more sample-efficient.

Q10. What is disentangled attention in DeBERTa? How does it improve NLI? Standard attention computes Q·K using combined content+position embeddings. DeBERTa decomposes this into four components: content-to-content, content-to-position, position-to-content, and position-to-position. For NLI, this means the model can separately reason about *what* words mean and *where* they appear — crucial for detecting negation placement ("not dead" vs. "dead, not").

Q11. Explain the difference between a cross-encoder and a bi-encoder. Why is a cross-encoder better for NLI? A bi-encoder encodes each text independently (parallel encoding) and compares embeddings — fast but loses cross-attention. A cross-encoder concatenates both texts and processes them jointly through all Transformer layers, enabling token-level cross-attention. For NLI, cross-encoders outperform bi-encoders by 5–8% because detecting entailment/contradiction requires fine-grained token interactions between premise and hypothesis.

Q12. What is the computational trade-off of using a cross-encoder vs. a bi-encoder? Cross-encoders are $O(n^2)$ in the combined length of both texts (full attention). Bi-encoders are $O(n^2)$ per text independently but allow pre-computation and caching of embeddings. For my system, I use SBERT (bi-encoder) for fast semantic filtering (~10ms/sentence) and DeBERTa (cross-encoder) for accurate NLI (~50ms/pair). This is a deliberate two-stage architecture.

Q13. Why `nli-deberta-v3-small` and not `nli-deberta-v3-base` or `large`? Memory constraints on a 4GB EC2 instance. Base (~440MB) + SBERT (~90MB) + spaCy (~12MB) would exceed available RAM under concurrent requests. Small (~180MB) achieves 88% MNLI — only 2% below base — while using less than half the memory. The accuracy-to-resource ratio favors small for CPU-only inference.

Q14. What is Sentence-BERT (SBERT)? How does it differ from standard BERT embeddings? Standard BERT produces contextual token embeddings, not fixed sentence embeddings. Naive approaches (e.g., [CLS] token or mean pooling) yield poor sentence representations. SBERT fine-tunes a siamese/triplet network on

NLI + STS data to produce semantically meaningful fixed-size sentence embeddings. Two similar sentences have high cosine similarity in SBERT space.

Q15. Why `all-MiniLM-L6-v2` for embeddings? How does it compare to larger models? MiniLM-L6-v2 produces 384-dimensional embeddings with ~78% Spearman correlation on STS benchmarks. Larger models like `all-mpnet-base-v2` (768-dim, ~83% STS) are more accurate but 3× larger and slower. Given I'm running on CPU with memory constraints, MiniLM's accuracy-to-speed ratio is optimal. The 384-dim embeddings are compact yet expressive enough for sentence-level similarity.

Q16. What is the softmax function? Why is it applied to NLI logits? Softmax converts raw logits (unbounded real numbers) into a probability distribution that sums to 1: $σ(z_i) = e^{z_i} / Σe^{z_j}$. Applied to NLI logits [contradiction, entailment, neutral], it gives calibrated probabilities for each label. This allows comparison across labels and enables confidence thresholding.

Q17. Why did you set `torch.set_grad_enabled(False)` globally? Disabling gradient computation eliminates the memory overhead of storing intermediate activations for backpropagation. Since the system is inference-only (no training), gradients are unnecessary. This reduces memory footprint by ~30% and marginally improves inference speed.

Q18. What is the `max_length=512` truncation in your NLI tokenizer? What happens to long texts? DeBERTa's maximum context window is 512 tokens. Texts exceeding this are truncated from the right. For NLI, the input is "[CLS] premise [SEP] hypothesis [SEP]" — both must fit within 512 tokens. My pipeline mitigates this by selecting the single best matching *sentence* (typically 20-50 tokens) rather than feeding full articles.

---

3. System Architecture & Design Philosophy

Q19. Describe the high-level architecture of VeriFact. A six-stage retrieval-augmented verification pipeline: (1) Claim Extraction via spaCy sentence scoring, (2) Query Generation via LLM decomposition or NER fallback, (3) Web Search via 3-tier API chain, (4) Evidence Processing with parallel scraping + SBERT filtering, (5) Stance Detection via local cross-encoder NLI with pre-filters, (6) Verdict Computation with weighted scoring + optional LLM tiebreaker + AI summary.

Q20. What is "retrieval-augmented verification"? How does it relate to RAG? RAG (Retrieval-Augmented Generation) retrieves documents and feeds them to an LLM for generation. My system is RAV (Retrieval-Augmented Verification) — it retrieves documents but uses deterministic NLI reasoning instead of generative LLM. The key difference: RAG generates free-text (hallucination-prone), RAV produces structured verdicts traceable to specific evidence. The LLM is only used optionally for summaries, not for verdict computation.

Q21. Why prioritize provenance over accuracy optimization? Accuracy without explainability is dangerous in fact-checking — a black-box "LIKELY FALSE" verdict is useless without showing *why*. Provenance means every verdict traces to specific URLs and sentences. This enables: (1) user verification of the system's reasoning, (2) audit trails for accountability, (3) identification of systematic errors.

Q22. Why is VeriFact inference-only? What would change if you added training? Inference-only means no custom fine-tuning — it leverages pre-trained models' generalized reasoning. Adding training would: (1) create dataset dependency and obsolescence risk, (2) require labeled fact-checking data (expensive, biased), (3) risk overfitting to specific claim types, (4) need periodic retraining as topics shift. The NLI model's domain-agnostic reasoning is more sustainable.

Q23. Explain the layered architecture. Why four layers? (1) **Interface Layer** — Flask API with validation, rate limiting, CORS — separates protocol concerns, (2) **Controller Layer** — `check_claim()` orchestrates pipeline — single point of control flow, (3) **Core Services** — 11 modules with single responsibilities — enables independent testing and replacement, (4) **External Interfaces** — API wrappers — isolates third-party dependencies. This follows the clean architecture principle of dependency inversion.

Q24. Why synchronous pipeline instead of async/event-driven? Fact-checking is inherently sequential — you can't detect stance without evidence, can't get evidence without search queries, can't generate queries without the claim. The only parallelizable step is evidence processing (scraping multiple URLs), which I handle with ThreadPoolExecutor. Full async would add complexity without significant benefit for the critical path.

Q25. What is the Single Responsibility Principle? How do your 11 modules follow it? Each module has one reason to change: `claim_extractor` only changes if extraction logic changes, `stance_detector` only if NLI logic changes, `source_scorer` only if credibility weights change. This enables: independent testing (8 test files), independent deployment, and easy replacement (e.g., swapping SBERT model requires changing only `model_registry` and `embedder`).

---

4. Claim Extraction Pipeline

Q26. How does your sentence importance scoring work? Walk through the algorithm. For each of the first 10 sentences: (1) +2 per unique entity type (PERSON, ORG, GPE), (2) +1 if contains numeric tokens, (3) +1 if 30-200 chars, (4) +1.5 if contains quotation marks, (5) position bonus: 1st sentence +1.5, 2nd +1.0, 3rd +0.5. Sentences are sorted by total score; the top scorer becomes the claim. This balances entity richness, statistical content, and position bias.

Q27. Why not always take the first sentence as the claim? Articles often bury the verifiable claim deep within context. A headline might say "Breaking News" (not a claim), the first sentence provides background, and the actual assertion appears in sentence 3 or 4. The scoring heuristic identifies which sentence is most "claim-like" — containing entities, numbers, quotes, and optimal length.

Q28. Explain your embedding-based keyword extraction. How is it equivalent to KeyBERT? Same algorithm as KeyBERT: (1) generate n-gram candidates from noun chunks + named entities + POS-filtered tokens, (2) embed all candidates + full text using SBERT, (3) rank candidates by cosine similarity to the full text, (4) apply MMR (λ=0.7) to select diverse, relevant keywords. The only difference: I use my local MiniLM model instead of KeyBERT's default.

Q29. What is Maximal Marginal Relevance (MMR)? Why λ=0.7? MMR balances relevance and diversity: $MMR = λ × sim(candidate, text) - (1-λ) × max(sim(candidate, selected_i))$. λ=0.7 means 70% weight on relevance, 30% on diversity. This prevents selecting redundant keywords (e.g., "climate change" and "climate crisis"). I chose 0.7 because queries need to be relevant first, diverse second.

Q30. What happens when keyword extraction fails? What's the fallback? Falls back to spaCy noun chunks — iterates `doc.noun_chunks`, filters out stop words and short phrases, returns the first `top_n` results. Less accurate than embedding-based extraction but reliable and fast. This is wrapped in a try/except to ensure the pipeline never halts due to keyword extraction failure.

Q31. Why cap text processing at 5000 characters for claim extraction? spaCy's NLP pipeline (tokenization, NER, dependency parsing) is O(n^2) for some operations. Processing 50,000+ characters would be slow and waste compute on content unlikely to contain the main claim. 5,000 chars covers the first ~800 words — sufficient for most article introductions.

5. Query Generation

Q32. Why generate multiple search queries instead of searching the raw claim? Raw claims suffer from **keyword myopia** — "Elon Musk lives on Mars" would search for those exact words, missing evidence phrased differently. Multiple queries cover: (1) the exact claim, (2) fact-check variations, (3) entity-focused queries, (4) queries designed to *disprove* the claim. This improves recall and reduces confirmation bias.

Q33. Explain the dual-strategy query generation (LLM + NER fallback). LLM path: Claim → Groq Llama-3.3-70b with structured prompt → 4-5 diverse queries including disproving queries → add raw claim + "fact check" baseline → dedup via `dict.fromkeys()` → cap at 10. **NER fallback:** spaCy extracts entities → generates exact claim + suffixes ("fact check", "hoax") + entity-centric queries + keyword queries → dedup via `set()` → cap at 10.

Q34. Why does the LLM prompt explicitly ask for queries that could DISPROVE the claim? To avoid **confirmation bias** in evidence retrieval. If you only search for supporting evidence, you'll find it (even for false claims). Including disproving queries ("X hoax", "X debunked") ensures the evidence pool represents both sides, allowing the NLI model to make a balanced assessment.

Q35. Why `dict.fromkeys()` for LLM dedup but `set()` for NER dedup? `dict.fromkeys()` preserves insertion order — important for LLM queries because the model orders them by importance. `set()` doesn't preserve order — acceptable for NER queries since they're heuristic-generated and order doesn't matter.

Q36. What happens if the LLM returns malformed JSON for queries? The response is parsed with `json.loads()`. If it starts with markdown code fences (```), those are stripped first. If parsing fails (JSONDecodeError), the system logs a warning and falls back to NER-based queries. The LLM is *never* in the critical path — failure always degrades gracefully.

6. Web Search & Evidence Retrieval

Q37. Explain the 3-tier search fallback chain. Why this specific order? (1) **Tavily** — best quality, `advanced` search depth, fastest response, but requires paid API key. (2) **Brave** — 2,000 free requests/month, reliable, but rate-limited. (3) **DuckDuckGo** — completely free, no API key, but hit-or-miss quality and aggressive rate limiting. The order prioritizes result quality while ensuring the system always has a search provider.

Q38. How do you handle DuckDuckGo rate limiting? Three mechanisms: (1) Uses `lite` backend instead of default (lighter, less likely to trigger limits), (2) 1.5-second delay between queries, (3) 3 retries with exponential backoff (5s, 10s, 15s). If web search returns empty, falls back to DDG's news search API. If all attempts fail, the system proceeds with whatever results were obtained.

Q39. Why cache API clients as singletons? What's the benefit? Creating a new Tavily client or HTTP session per request wastes TCP connection overhead (TLS handshake, DNS resolution). Singletons reuse the same connection pool. The `_tavily_client` and `_brave_session` are module-level globals initialized on first use — subsequent calls reuse the same authenticated session with persistent connections.

Q40. How does URL deduplication work across search tiers? Each tier maintains a `seen_urls` set. As results come in, URLs are checked against the set before being added. However, deduplication is currently *within* each tier, not *across* tiers — if Tavily fails and Brave is tried, Brave starts with a fresh set. Cross-tier dedup happens implicitly because if Tavily succeeds, Brave is never called.

Q41. What is the `search_depth="advanced"` parameter in Tavily? Tavily's advanced mode performs deeper crawling and analysis of search results, returning more relevant and higher-quality content compared to basic mode. It accesses more pages and provides better content extraction. This justifies using Tavily as the primary search provider despite requiring a paid API key.

---

7. Scraping & Content Extraction

Q42. Why trafilatura for web scraping? How does it compare to BeautifulSoup? trafilatura performs *intelligent* content extraction — it uses heuristics and machine learning to identify the main article body, stripping ads, navigation bars, sidebars, footers, and boilerplate. BeautifulSoup is a general-purpose HTML parser — you'd need to write extraction rules per site. trafilatura is specifically designed for article text extraction.

Q43. Explain the connection pooling in your scraper. I use a `requests.Session` with an `HTTPAdapter` configured for 10 persistent connections (`pool_connections=10, pool_maxsize=10`). This reuses TCP connections across requests, avoiding the overhead of TLS handshakes for each URL. The adapter also implements retry logic: 2 retries with 0.5s exponential backoff for 429/5xx status codes.

Q44. Why an 8-second timeout for scraping? Trade-off between coverage and latency. 8 seconds catches 95%+ of responsive sites. Sites taking longer are likely behind CDNs, paywalls, or load balancers that won't serve useful content anyway. The original 15-second timeout made total pipeline latency unacceptable. Reducing to 8s cut worst-case evidence processing time by ~40%.

Q45. How do you handle 403 (Cloudflare/paywall) responses? Gracefully skip — log the failure and continue processing remaining URLs. The pipeline doesn't halt for individual scraping failures. This is critical because many high-quality news sites (NYT, Washington Post) return 403 for automated requests. The source credibility scorer still assigns appropriate weights to successfully scraped sources.

---

8. Semantic Similarity & Embeddings

Q46. What is cosine similarity? Why use it over Euclidean distance for text? Cosine similarity measures the angle between vectors: $cos(θ) = (A·B)/(||A|| × ||B||)$. It's magnitude-invariant — a long document and short sentence about the same topic will have high cosine similarity despite different vector magnitudes. Euclidean distance would penalize the length difference. For normalized embeddings (which SBERT produces), cosine similarity is ideal.

Q47. Why a 384-dimensional embedding space? What's the trade-off with higher dimensions? 384 dimensions capture sufficient semantic granularity for sentence-level comparison. Higher dimensions (768, 1024) capture more nuance but: (1) increase memory (2× for 768-dim), (2) increase computation time for cosine similarity, (3) risk the curse of dimensionality. MiniLM-L6-v2's 384 dims achieve ~78% STS correlation — competitive with 768-dim models at half the resource cost.

Q48. Explain the LRU cache on claim embeddings. Why `maxsize=32`? The claim embedding is computed once and cached using `@lru_cache(maxsize=32)`. When processing multiple evidence sources against the same claim, the claim is embedded only once. 32 entries handles typical usage (one claim per request, with some headroom for concurrent requests across Gunicorn workers). The cache key requires converting the embedding to a tuple for hashability.

Q49. Why is the similarity threshold set to 0.25? What would happen at 0.1 or 0.5? 0.25 is calibrated for optimal recall/precision. At 0.1: too many irrelevant sentences pass through → wasted NLI compute + false positives from topic overlap. At 0.5: only near-paraphrases pass → misses relevant evidence phrased differently. 0.25 captures "topically related" content while filtering pure noise. Sentences below 0.25 are marked "discusses" with zero confidence, skipping expensive NLI inference.

Q50. What is the early-exit optimization? How much compute does it save? If the best-matching sentence from a source has similarity < 0.25, stance detection (DeBERTa inference, ~50ms) is entirely skipped. In practice, 30-40% of scraped sources produce irrelevant content. Skipping NLI for these saves ~100-200ms per request. The evidence is still included in results (as "discusses") but contributes zero to the verdict score.

---

9. Stance Detection Deep Dive

Q51. Walk through the `detect_stance()` function step by step. (1) Check for outcome mismatch (negating modifiers, explicit negations) — if found, return override. (2) Check if claim is a relationship claim — if contradiction detected, return override. (3) Run NLI inference: tokenize premise+hypothesis, forward pass through DeBERTa, softmax logits → entailment/contradiction/neutral scores. (4) Map highest score to label (supports/refutes/discusses). (5) Apply minimum confidence thresholds (0.50 supports, 0.40 refutes). (6) Apply temperature calibration. (7) Apply high-stakes/relationship thresholds if applicable.

Q52. What is temperature scaling? Why T=1.3? Temperature scaling adjusts confidence calibration: $calibrated = raw^{1/T}$. With T=1.3, a raw 0.90 becomes 0.92^(1/1.3) ≈ 0.85, pulling overconfident predictions downward. T=1.0 means no change, T>1 reduces confidence, T<1 increases it. I chose 1.3 empirically — enough to reduce false confidence without collapsing the confidence range.

Q53. Why are support and refute confidence thresholds asymmetric (0.50 vs 0.40)? False support is more common than false refutation in NLI. Topic overlap frequently causes the model to output moderate entailment scores (0.45-0.50) for content that merely *discusses* the topic without truly supporting the claim. Refutation is harder to trigger falsely — the model must detect genuine contradiction. The asymmetry reduces false positive "supports" labels.

Q54. What is the high-stakes claim threshold? Why 0.70 for support? Claims involving "killed," "assassinated," "nuclear," "war" etc. have severe consequences if incorrectly validated. The standard 0.50 threshold risks confidently supporting dangerous claims based on topical overlap. Raising to 0.70 for support means the model must be very confident before saying a high-stakes claim is supported. Refutation threshold remains unchanged because refuting dangerous claims is safer.

Q55. Explain outcome mismatch detection. Give a concrete example. Claim: "Trump was assassinated." Evidence: "Trump survived an assassination attempt." SBERT similarity is very high (same entities, same event). But semantically they're opposite — "was assassinated" vs. "survived." The outcome mismatch detector catches the word "survived" in evidence but not in claim, and overrides to "refutes" with 0.75 confidence *before* NLI even runs.

Q56. Why does outcome mismatch detection run BEFORE NLI? Because NLI models are confused by high topical similarity with opposite outcomes. DeBERTa might output high entailment for "assassination attempt" evidence against an "assassination" claim because the tokens are nearly identical. The pre-filter catches these cases cheaply (string matching, ~0.1ms) before expensive NLI inference (~50ms).

Q57. How does relationship claim verification work? Why is NLI insufficient? NLI understands logical relationships but not entity-level facts. "Rahul Gandhi is Modi's son" vs. "Rahul Gandhi is Rajiv Gandhi's son" — NLI might say entailment because both discuss Rahul Gandhi's parentage. The relationship module extracts (subject, relationship, object) tuples via regex and detects *different objects* for the same subject+relationship → contradiction with 0.85 confidence.

Q58. What regex patterns do you use for relationship extraction? Three patterns: (1) `X is Y's [rel]` — captures possessive form, (2) `X is the [rel] of Y` — captures "of" form, (3) `X is Ys [rel]` — captures possessive without apostrophe. Each extracts a (subject, relationship_type, object) tuple. The patterns handle common relationship words: son, daughter, father, mother, wife, husband, brother, sister, child, parent.

---

10. Evidence Aggregation

Q59. How does `aggregate_sentence_stances()` work when you have multiple sentences? Weighted voting: each sentence contributes `similarity × confidence` to its stance category (support/refute/neutral). The winning stance has the highest total weight. Confidence is the winner's weight divided by total weight. Example: 2 sentences support (weights 0.3, 0.4) and 1 refutes (weight 0.5) → support wins (0.7 vs 0.5) with confidence 0.7/1.2 ≈ 0.58.

Q60. Why use ThreadPoolExecutor with 3 workers? Why not 5 or asyncio? 5 workers caused OOM kills on 4GB EC2 — each thread loads evidence into memory while running NLI inference. 3 workers balances parallelism with memory safety. asyncio would be more memory-efficient but requires async-compatible libraries (trafilatura is synchronous). ThreadPoolExecutor provides simple parallelism for I/O-bound scraping with minimal code complexity.

Q61. What is the `as_completed()` pattern? Why use it over `map()`? `as_completed()` yields futures as they finish, regardless of submission order. This means fast-responding URLs are processed immediately without waiting for slow ones. `map()` would wait for results in submission order — a slow first URL would block all subsequent results. For scraping with variable latency, `as_completed()` minimizes total wait time.

PART B: VERDICT LOGIC, DEPLOYMENT & ADVANCED TOPICS

---

11. Verdict Engine & Scoring

Q62. Walk through the complete scoring formula for a single evidence source. $Score = Sim × Conf × Dir × W_{src} × Boost$. Example: similarity=0.8, confidence=0.7, stance=supports (+1), source=Reuters (1.5), boost=$1.0+(0.8-0.5)×0.5=1.15$. Score = $0.8 × 0.7 × 1 × 1.5 × 1.15 = 0.966$. This single source strongly supports the claim.

Q63. What is the similarity boost factor? Why is it necessary? $Boost = 1.0 + (Sim - 0.5) × 0.5$. It ranges from 0.75× (sim=0.0) to 1.25× (sim=1.0). This amplifies high-similarity evidence (which is more likely to be genuinely relevant) and dampens low-similarity evidence (which may be tangentially related). Without it, a low-similarity source with high NLI confidence could disproportionately influence the verdict.

Q64. Why is the verdict threshold ±0.35? How were these values tuned? Originally ±0.4. Lowered to ±0.35 for better recall — the system was producing too many MIXED verdicts for claims with moderate but clear evidence. The tuning was empirical: testing known true/false claims and adjusting until the system correctly classified them without increasing false positives. Lower thresholds = more decisive verdicts, higher thresholds = more MIXED.

Q65. Why use sigmoid for confidence rather than just normalizing the net score? Sigmoid maps any real number to (0, 1) with a natural S-curve: $σ(x) = 1/(1+e^{-x})$. Linear normalization would require defining min/max bounds (which are unknown). Sigmoid provides: (1) bounded output, (2) diminishing returns for extreme scores (score=5 doesn't feel 5× more confident than score=1), (3) natural interpretation as probability-like confidence.

Q66. What happens when net_score is exactly 0.35? Is it LIKELY TRUE or MIXED? The threshold is strict: `net_score > 0.35` for LIKELY TRUE. So exactly 0.35 falls into MIXED/MISLEADING. This is intentional — borderline cases should be flagged as inconclusive rather than confidently true.

Q67. How does the explanation builder work? What information does it expose? Builds a 5-step reasoning chain: (1) Evidence Collection — count of sources, (2) Stance Analysis — support/refute/neutral counts, (3) Credibility Weighting — trusted source count, (4) Score Calculation — net score with support/refute breakdown, (5) Verdict — decision reason with threshold reference. Also includes a breakdown object with counts, weights, and threshold info.

Q68. Why does the verdict engine import `llm_helper` lazily inside the function? Two reasons: (1) Avoids circular imports — `verdict_engine` and `llm_helper` could reference each other indirectly. (2) Makes LLM truly optional — if Groq packages aren't installed, the import fails inside a try/except and the system continues without LLM features. Eager top-level imports would crash the module on startup.

---

12. LLM Integration (Groq)

Q69. What is Groq? Why Groq over OpenAI or Anthropic? Groq is an AI inference company that provides API access to open-source LLMs (Llama-3.3-70b) with extremely low latency (~1-2s for generation). Advantages over OpenAI/Anthropic: (1) uses open-source models (no vendor lock-in), (2) faster inference via custom hardware (LPU), (3) generous free tier for development, (4) the model (Llama-3.3-70b) is transparent and well-documented.

Q70. What are the three LLM-powered features? Are they all optional? (1) **Claim decomposition** — generates precise search queries. (2) **Verdict tiebreaker** — resolves MIXED verdicts. (3) **AI summary** — plain-language explanation of verdict. All three are optional. Without Groq: (1) falls back to NER-based queries, (2) MIXED verdicts stay MIXED, (3) falls back to rule-based decision reason. The core NLI pipeline is entirely LLM-independent.

Q71. When does the LLM tiebreaker activate? What are its constraints? Only activates when: (1) NLI produces a MIXED verdict ($-0.35 ≤ NetScore ≤ 0.35$), AND (2) Groq is available. The LLM receives the claim

+ up to 8 evidence sources with stances. Its verdict is only accepted if: confidence ≥ 0.60 AND verdict is LIKELY TRUE, LIKELY FALSE, or UNVERIFIABLE. Below 0.60 confidence, the MIXED verdict is preserved.

Q72. Can the LLM tiebreaker override the NLI verdict? Only partially. If NLI produces a decisive verdict (LIKELY TRUE or LIKELY FALSE), the LLM is never consulted. The LLM only activates for MIXED verdicts. Even then, it can only change MIXED → LIKELY TRUE/FALSE/UNVERIFIED, never the reverse. The NLI pipeline is always the authority when confident.

Q73. How do you prevent prompt injection via evidence text in the tiebreaker? The LLM receives evidence text that came from scraped web pages — an attacker could inject instructions. Mitigation: (1) the LLM is never the primary decision-maker, only an auxiliary signal, (2) its verdict is constrained (must be one of three values), (3) confidence must exceed 0.60, (4) the structured JSON output format limits free-form manipulation. However, this remains a known attack surface.

Q74. How does the AI summary prompt prevent technical jargon? The prompt explicitly instructs: "Do NOT mention scores, thresholds, algorithms, or technical details. Do NOT use phrases like 'based on our analysis.' Write as if you are a journalist explaining the finding." This ensures the output is user-friendly. If the LLM produces jargon despite instructions, the response is still usable.

Q75. What is the `temperature=0.1` parameter in Groq calls? Why so low? Temperature controls randomness in LLM output. 0.1 means highly deterministic — nearly always picks the highest-probability token. For fact-checking, we want consistent, reproducible outputs — not creative or varied responses. The summary prompt uses 0.3 for slightly more natural language, but still constrained.

---

13. Source Credibility Scoring

Q76. Explain the three-tier credibility system. How are weights assigned? Tier 1 (W=1.5): Wire services (Reuters, AP) and fact-checkers (Snopes, PolitiFact) — editorially rigorous, low bias. Tier 1.5 (W=1.3-1.4): Major newspapers (BBC, NYT, Guardian) and gov/edu domains. Tier 2 (W=1.0): Unknown/unlisted domains — default neutral weight. Tier 3 (W=0.3-0.6): Social media — user-generated, unverified content.

Q77. How does domain extraction work? What about subdomains? `urlparse()` extracts the netloc, then `www.` prefix is stripped. Example: `https://www.bbc.com/news/article` → `bbc.com`. This handles most cases but doesn't handle subdomains like `news.bbc.co.uk` — it would check `news.bbc.co.uk` against the lookup, which wouldn't match `bbc.co.uk`. However, `bbc.co.uk` is separately listed in the trusted sources.

Q78. Why is Reddit scored higher (0.6) than Facebook (0.4)? Reddit's upvote/downvote system and community moderation provide some signal filtering — factual discussions tend to surface. Facebook has less content curation and more sharing of unverified claims. However, both are still social media and significantly penalized compared to editorial sources.

Q79. What's the biggest weakness of static credibility scoring? New high-quality sources (e.g., a new investigative outlet) receive the default 1.0 weight. Known unreliable sources not in the list also get 1.0. The system cannot adapt to changing credibility — a once-reputable outlet that becomes biased still gets its original weight. A dynamic approach using external credibility APIs would address this.

Q80. How would you improve source scoring in a production system? (1) Integrate with external credibility APIs (e.g., NewsGuard, Media Bias/Fact Check), (2) implement domain-level credibility learning from user feedback, (3) add temporal weighting (recent articles from a domain vs. historical trustworthiness), (4) analyze author credibility, not just domain, (5) cross-reference with fact-checking databases like ClaimReview.

---

14. Deployment & Infrastructure

Q81. Describe your Docker setup. Why `python:3.10-slim`? `python:3.10-slim` is Debian-based but minimal (~120MB vs ~900MB for full). Includes only essential runtime libraries. I add `gcc`, `g++` (for compiling C extensions like numpy/scipy), and `curl` (for health checks). Non-root user `appuser` is created for security. ML models are NOT baked into the image — they're lazy-loaded at runtime.

Q82. Why lazy-load models instead of baking them into the Docker image? Baking would add ~360MB (DeBERTa + MiniLM + spaCy) to the image, making pulls/pushes slower and CI/CD more expensive. Lazy loading means: (1) smaller image size (~2GB vs ~2.4GB), (2) faster deployment, (3) models are cached after first download. The `/api/warmup` endpoint lets you pre-load after deployment.

Q83. Explain your Gunicorn configuration: 2 workers × 4 threads. Workers are OS processes — each gets its own memory space and Python GIL. Threads share memory within a worker. 2 workers × 4 threads = 8 concurrent request handlers. Why not more workers? Each worker loads models into memory (~360MB). 2 workers × 360MB = 720MB just for models. On 4GB RAM, more workers would cause OOM.

Q84. Why `gthread` worker type? What about `gevent`? `gthread` uses OS threads — good for I/O-bound work (network calls) alongside CPU-bound work (Transformer inference). `gevent` uses cooperative green threads — excellent for pure I/O but can't parallelize CPU-bound work. Since NLI inference is CPU-intensive, `gthread` allows real parallelism for I/O (scraping) while handling CPU inference.

Q85. What is the `--max-requests 500` flag? Why is it important? After 500 requests, the worker process is restarted. This prevents gradual memory leaks (common with ML models and large NumPy arrays) from accumulating. The `--max-requests-jitter 50` adds randomization so all workers don't restart simultaneously, ensuring continuous availability.

Q86. Explain the Docker HEALTHCHECK. What's the `start-period`? `HEALTHCHECK --interval=30s --timeout=15s --start-period=30s --retries=3`: checks `/api/health` every 30s. `start-period=30s` gives the container 30 seconds to start before health checks begin — enough for Python imports and Flask initialization. After 3 consecutive failures, Docker marks the container as unhealthy.

Q87. Describe the CI/CD pipeline. What happens on each push to main? GitHub Actions: (1) **Test** — runs `pytest` suite, fails the pipeline if any test fails. (2) **Build** — builds Docker image, pushes to DockerHub as `adetyajamwal/fake-news-detector:latest`. (3) **Deploy** — SSHs into EC2, runs `docker system prune` (free disk space), pulls new image, stops old container, starts new container with env vars.

Q88. What GitHub Secrets are needed? Why can't they be in the repo? `DOCKERHUB_USERNAME`, `DOCKERHUB_TOKEN`, `EC2_HOST`, `EC2_SSH_KEY`, `TAVILY_API_KEY`. These contain credentials and private keys — committing them to the repo would be a security vulnerability. GitHub Secrets are encrypted and only available to Actions workflows during execution.

Q89. What is WhiteNoise? Why use it instead of Nginx for static files? WhiteNoise serves static files directly from the WSGI app in production. It handles proper Content-Length headers, compression, and caching. Nginx would be overkill for this single-app deployment. WhiteNoise eliminates the need for a separate reverse proxy layer, simplifying the Docker setup.

Q90. What happens if the EC2 instance is stopped and restarted? Docker's `--restart unless-stopped` policy auto-starts the container on reboot. However: (1) the public IP changes (unless Elastic IP is configured), (2) ML models need re-downloading (unless Docker volume is persistent), (3) the model warmup adds latency to the first request. Elastic IP + warmup in startup script mitigates these issues.

---

15. Testing Strategy

Q91. Describe your testing approach. What types of tests do you have? 7 test files covering: (1) Unit tests — `test_verdict_engine.py` (scoring, thresholds), `test_source_scorer.py` (credibility weights), `test_claim_extractor.py` (extraction logic), `test_query_generator.py` (query generation). (2) Integration tests — `test_integration.py` (end-to-end pipeline). (3) Feature tests — `test_llm_summary.py` (AI summary), `test_web_search.py` (search fallback). All run via `pytest`.

Q92. How do you test the verdict engine without calling real APIs? `test_verdict_engine.py` constructs synthetic evidence dictionaries with known values (similarity, stance, confidence, source_weight) and passes them to `compute_final_verdict()`. This tests the scoring math, threshold logic, and explanation builder in isolation. No network calls, no model loading — pure unit tests.

Q93. How do you test search fallback behavior? Mock the API clients — simulate Tavily returning errors, Brave returning empty results, and verify DuckDuckGo is called. The fallback chain logic can be tested by manipulating environment variables (removing API keys) and mocking the search functions.

Q94. What edge cases does `test_verdict_engine.py` cover? Empty evidence (→ UNVERIFIED), single strong support (→ LIKELY TRUE), single strong refute (→ LIKELY FALSE), balanced support+refute (→ MIXED), all neutral evidence (→ MIXED), extreme scores, boundary conditions at ±0.35 threshold, and evidence with zero similarity.

---

16. Security & Robustness

Q95. What input validation does the API perform? Pydantic `CheckRequest` model validates: (1) `max_results` must be 1-10, (2) `url` must start with http:// or https://, (3) at least one of `claim`, `text`, or `url` must be provided, (4) extracted claim must be ≥10 characters. Invalid inputs return 400 with structured error details.

Q96. How does rate limiting work? Why 5/min per IP? Flask-Limiter with in-memory storage. 5 requests/minute per IP prevents abuse — each request involves web scraping and ML inference (expensive). 100/hour global limit provides burst capacity. In-memory storage means limits reset on restart — acceptable for a single-instance deployment.

Q97. What is CORS? Why is it enabled? CORS (Cross-Origin Resource Sharing) allows the frontend (served from the same domain) and external clients to make API requests. Without CORS, browsers would block requests from different origins. `flask-cors` adds the `Access-Control-Allow-Origin` header to responses.

Q98. How does the system handle adversarial inputs? Defense layers: (1) Input validation (length, format), (2) rate limiting, (3) similarity threshold filters irrelevant noise, (4) minimum confidence thresholds prevent low-confidence verdicts. However, adversarial *semantic* attacks (e.g., claims designed to exploit NLI blind spots) are an open problem — the system does not claim adversarial robustness.

Q99. What happens if a user sends a malicious URL? The URL is passed to trafilatura which fetches the HTML. Risks: (1) SSRF — the server fetches an internal URL. Mitigation: trafilatura doesn't access internal networks by default. (2) Malicious HTML — trafilatura extracts text only, doesn't execute scripts. (3) Very large pages — text is capped at 10,000 characters. No user-provided code is ever executed.

Q100. Why run as `appuser` instead of root in Docker? Security best practice — if an attacker exploits a vulnerability in the app, they're limited to `appuser` privileges. Root would give full container access, potentially allowing container escape or filesystem manipulation. `appuser` (UID 1000) has only read/execute permissions on `/app`.

---

17. ML Theory & Concepts

Q101. What is transfer learning? How does VeriFact use it? Transfer learning reuses a model trained on one task for a different task. VeriFact uses DeBERTa-v3 (pre-trained on MNLI for NLI) and applies it directly to stance detection without fine-tuning. The model's understanding of entailment/contradiction transfers seamlessly to fact-checking because stance detection IS a form of NLI.

Q102. What is the attention mechanism? How does it help NLI? Attention computes weighted relationships between all token pairs: $Attention(Q,K,V) = softmax(QK^T/√d_k)V$. For NLI, attention enables the model to directly compare tokens in the premise with tokens in the hypothesis — discovering relationships like "survived" in evidence attending to "killed" in claim, signaling contradiction.

Q103. What is multi-head attention? Why multiple heads? Multi-head attention runs N parallel attention computations with different learned projections, then concatenates results. Each head can focus on different relationship types: one head might attend to entity relationships, another to negation patterns, another to temporal indicators. This gives the model richer representational capacity.

Q104. What is the difference between encoder and decoder Transformers? Which does DeBERTa use? Encoders process both directions (bidirectional attention) — good for understanding. Decoders process left-to-right (causal attention) — good for generation. DeBERTa is encoder-only — it processes the full premise+hypothesis bidirectionally, which is ideal for classification tasks like NLI where you need global understanding of the input.

Q105. What is tokenization? How does DeBERTa's tokenizer work? Tokenization converts text into subword units. DeBERTa uses SentencePiece (similar to BPE). "Assassination" might become ["Ass", "ass", "ination"]. Subword tokenization handles out-of-vocabulary words by breaking them into known pieces. The tokenizer also adds special tokens: [CLS] premise [SEP] hypothesis [SEP].

Q106. What is the curse of dimensionality? How does it relate to embeddings? In high-dimensional spaces, distances between points become increasingly uniform, making similarity measures less discriminating. With 384 dimensions (MiniLM), this is manageable. With 10,000+ dimensions (sparse TF-IDF), cosine similarity becomes less meaningful. Dense, low-dimensional embeddings (SBERT) mitigate this by learning compact, meaningful representations.

Q107. What is overfitting? Why does the inference-only approach avoid it? Overfitting occurs when a model memorizes training data patterns instead of learning generalizable features. Since VeriFact doesn't train any models — it uses pre-trained models in inference mode — overfitting to specific claim types is impossible. The model's behavior is fixed; only the input data changes.

Q108. What is the bias-variance trade-off in the context of your system? High bias (underfitting): the NLI model might be too general, missing domain-specific nuances. High variance (overfitting): would happen if fine-tuned on specific claim types. My approach accepts slightly higher bias (generic NLI reasoning) to get near-zero variance (consistent behavior across all domains). For fact-checking, this trade-off favors reliability.

Q109. Explain precision vs. recall in the context of your verdict thresholds. Precision: of claims labeled LIKELY FALSE, how many are actually false? Recall: of all false claims, how many are labeled LIKELY FALSE? Raising the threshold (e.g., ±0.5) increases precision (fewer false positives) but decreases recall (more false negatives become MIXED). I tuned to ±0.35 for balanced precision-recall.

---

18. Python & Software Engineering

Q110. What is the double-checked locking pattern? Where do you use it? In `model_registry.py`: check if model is None (fast, no lock), acquire lock, check again (prevents race condition where two threads both pass the first check), then initialize. This ensures thread-safe singleton initialization without the overhead of acquiring a lock on every access.

Q111. What is a WSGI server? Why Gunicorn over Flask's built-in server? WSGI (Web Server Gateway Interface) is the standardized interface between Python web apps and servers. Flask's built-in server is single-threaded and not production-ready. Gunicorn is a production WSGI server supporting multiple workers and threads, process management, graceful restarts, and proper signal handling.

Q112. What is connection pooling? How does your scraper implement it? Connection pooling reuses TCP connections across HTTP requests. My scraper's `requests.Session` with `HTTPAdapter(pool_connections=10, pool_maxsize=10)` maintains 10 persistent connections. When scraping multiple URLs from the same domain, the session reuses existing connections, avoiding TLS handshake overhead (~100ms per new connection).

Q113. Why use `@lru_cache` for claim embeddings? What's the memory impact? `@lru_cache(maxsize=32)` stores the 32 most recent claim embeddings in memory. Each embedding is 384 floats × 8 bytes = ~3KB. 32 entries = ~96KB — negligible memory cost for significant compute savings (avoid re-embedding the same claim for each evidence source).

Q114. What is Pydantic? Why use it for input validation? Pydantic provides data validation using Python type hints. It automatically: (1) type-coerces inputs, (2) validates constraints (field_validators), (3) generates structured error messages. Compared to manual validation, Pydantic is declarative, self-documenting, and produces consistent error responses — critical for a public API.

Q115. Explain the `os.environ["USE_TF"] = "NO"` line at the top of `app_flask.py`. The `transformers` library auto-detects available backends (PyTorch, TensorFlow, JAX). If TensorFlow is partially installed (common in some environments), it triggers slow, error-prone TF initialization. Setting `USE_TF=NO` and `USE_TORCH=YES` forces PyTorch-only mode, preventing TF-related startup issues.

---

19. Scenario-Based Questions

Q116. Claim: "Elon Musk died yesterday." Walk through how your system processes this. (1) Claim extracted directly (short, contains entity), (2) high-stakes keywords detected ("died"), (3) queries generated: "Elon Musk died" + "Elon Musk death fact check" + "Elon Musk alive" etc., (4) search returns recent news, (5) evidence likely says "Elon Musk is alive" or discusses recent activities, (6) outcome mismatch detector catches "alive" vs "died", (7) sources override to "refutes", (8) high-stakes threshold (0.70) applies for any "supports", (9) verdict: LIKELY FALSE with high confidence.

Q117. Claim: "Water boils at 100°C." How does your system handle well-known facts? Search returns scientific references confirming this. SBERT finds high-similarity sentences about boiling points. NLI classifies as entailment with high confidence. Source credibility likely high (educational sites, .edu). Net score strongly positive → LIKELY TRUE. This is a best-case scenario — simple factual claim with abundant, consistent evidence.

Q118. Claim: "Vaccines cause autism." How does the system handle debunked claims? Search returns fact-checking articles (Snopes, WHO, CDC). Evidence contains phrases like "debunked," "no evidence," "false claim" → outcome mismatch detector catches these explicit negation patterns. NLI detects contradiction. Fact-checking sources get high credibility weights (1.5). Verdict: LIKELY FALSE with high confidence.

Q119. What if all top search results are from biased sources on the same side? This is a limitation. If search results all come from one editorial perspective, the system reflects that bias. Mitigations: (1) query generation includes disproving queries, (2) source credibility weights amplify diverse authoritative sources, (3) social media is dampened. But if all authoritative sources agree (which is often correct), the system follows them.

Q120. What if the claim contains sarcasm? "Oh sure, the earth is definitely flat." The system will likely mishandle this. NLI models are generally poor at detecting sarcasm. The literal content ("earth is flat") will be fact-checked, and the sarcastic intent will be lost. Evidence will refute the literal claim → verdict LIKELY FALSE. This is actually the "correct" answer for the literal statement, but misses the user's sarcastic agreement.

Q121. A claim is in English but about a non-English event with limited English coverage. What happens? Search returns fewer results (limited English-language coverage). Fewer evidence sources → less confident verdict. If no evidence found → UNVERIFIED. The system correctly acknowledges its limitation rather than guessing. Improving coverage would require multilingual search and models.

Q122. What if your Tavily API key expires mid-request? The `_tavily_search()` call fails with an exception, which is caught. The `web_search()` function falls back to Brave (if key available), then DuckDuckGo. The request continues with alternative search results. The user never sees the Tavily failure — only a log entry records it.

Q123. What if the scraping phase returns zero successful articles? `build_evidence()` returns an empty list. `compute_final_verdict([])` returns UNVERIFIED with confidence 0.0 and the summary "No relevant evidence was found to verify or refute this claim." The system gracefully handles the absence of evidence.

Q124. A claim mixes truth and falsehood: "The moon is made of rock and cheese." How does your system respond? Search returns evidence supporting "moon is made of rock" (strongly) and contradicting "moon is made of cheese." Evidence will be split between supports and refutes. Net score likely near zero → MIXED/MISLEADING. This is actually the *correct* verdict — the claim contains both true and false elements.

---

20. Comparative Analysis & Future Work

Q125. How does VeriFact compare to Google's Fact Check API? Google's API indexes *existing* fact-checks from certified organizations — it tells you if someone has already checked a claim. VeriFact performs *original* verification — it works even for claims nobody has checked before. Google's approach is more reliable (human-verified) but limited in coverage. VeriFact is unlimited in scope but automated.

Q126. How does your system compare to ClaimBuster? ClaimBuster uses supervised learning to rank claim *check-worthiness* — it identifies which sentences in a text are worth fact-checking. VeriFact actually *performs* the fact-check. They solve different problems. ClaimBuster could be used as a pre-filter for VeriFact — identify claims, then verify them.

Q127. What would a GPU deployment look like? What improvements would you expect? A T4 GPU (~$0.50/hr on AWS) would: (1) reduce NLI inference from ~50ms to ~5ms per pair, (2) reduce SBERT encoding from ~10ms to ~1ms per sentence, (3) enable batch inference (process all evidence simultaneously). Total pipeline time could drop from 8-18s to 3-8s. The main bottleneck would shift from inference to web scraping latency.

Q128. How would you add multilingual support? Swap models: DeBERTa-v3 → mDeBERTa or XLM-RoBERTa (multilingual NLI), MiniLM-L6-v2 → multilingual-MiniLM. Add multilingual search APIs. The architecture supports this — it would be model swaps in `model_registry`, not structural changes. Main challenge: multilingual NLI models are less accurate than English-only ones.

Q129. How would you implement a feedback loop for continuous improvement? (1) Log user corrections (when they dispute the verdict), (2) aggregate correction patterns to identify systematic failures, (3) adjust confidence thresholds based on error rates, (4) update source credibility weights based on historical accuracy, (5) potentially fine-tune models on verified corrections — but carefully, to avoid overfitting to specific topics.

Q130. What would you change for a real-time social media monitoring use case? (1) Replace synchronous API with async (FastAPI + asyncio), (2) add claim deduplication (don't re-check identical claims), (3) implement caching of recent results (Redis), (4) add streaming support (WebSockets), (5) reduce evidence sources (1-2 instead of 3) for faster response, (6) add batch processing for high-volume ingestion.

Q131. How would you handle claims about very recent events (last hour)? Search engines may not have indexed the event yet. Mitigation: (1) prioritize news-specific search APIs with real-time indexing, (2) include social media results (despite lower credibility), (3) set higher UNVERIFIED threshold for recent events — if evidence is sparse, default to UNVERIFIED rather than guessing, (4) show recency warnings to users.

Q132. How does the system handle claims requiring numerical reasoning? Poorly — NLI models understand linguistic relationships but not mathematics. "India's GDP grew 7% in 2024" requires: (1) finding actual GDP data, (2) calculating growth rate, (3) comparing. The current system would find topically relevant evidence and attempt NLI, but couldn't verify the specific number. A numerical reasoning module would be needed.

---

21. Ethics & Misinformation Theory

Q133. What ethical concerns exist with automated fact-checking? (1) **False authority** — users may trust the system more than warranted, (2) **Censorship risk** — if used to suppress content, automated errors become suppression, (3) **Bias amplification** — if search results or source weights are biased, verdicts inherit that bias, (4) **Transparency** — users must understand the system's limitations, (5) **Accountability** — who is responsible when the system is wrong?

Q134. What is the difference between misinformation and disinformation? Misinformation is false information shared without malicious intent (genuine mistakes, outdated claims). Disinformation is deliberately fabricated or manipulated to deceive. VeriFact treats both the same — it verifies the claim against evidence regardless of intent. The system cannot infer intent behind a claim.

Q135. Can your system be weaponized? How? Yes: (1) Bad actors could use it to craft claims that *barely* pass as MIXED (by testing and refining claims against the system), (2) mass querying could be used to identify which false claims the system misclassifies, (3) the verdict could be cited as "AI-approved" to lend false authority. Mitigation: rate limiting, clear disclaimers, provenance trails.

Q136. What is confirmation bias in evidence retrieval? How do you mitigate it? The tendency to find only evidence supporting the initial hypothesis. Mitigation: (1) query generator explicitly includes disproving queries ("X hoax", "X debunked"), (2) source diversity through multi-API search, (3) equal treatment of supporting and refuting evidence in scoring (no asymmetry in direction weights).

---

22. Performance & Optimization

Q137. What is the end-to-end latency breakdown? Claim extraction: <1s | Query generation (LLM): 1-2s | Web search: 2-4s | Scraping (3 parallel sources): 3-8s | Embedding + Stance (per source): ~0.5s | Verdict computation: <0.1s | LLM summary: 1-2s | **Total: 8-18s**.

Q138. What is the bottleneck in your pipeline? Web scraping (3-8s) — it's I/O-bound, dependent on external servers' response times. Even with ThreadPoolExecutor parallelism, the slowest URL determines total scraping time. Mitigation: 8-second timeout, 3 workers, skip failed URLs. The next bottleneck is web search API latency (2-4s).

Q139. How would you reduce latency to under 5 seconds? (1) GPU inference — NLI and SBERT drop to ~6ms total, (2) reduce evidence sources from 3 to 1, (3) async scraping with timeouts, (4) result caching (Redis) for repeated claims, (5) pre-computed search indexes instead of live search, (6) streaming responses — send partial results as evidence arrives.

Q140. What is the memory footprint of the system at runtime? Base Python + Flask: ~50MB. DeBERTa-v3-small: ~180MB. MiniLM-L6-v2: ~90MB. spaCy en_core_web_sm: ~12MB. Per-request overhead (embeddings, evidence): ~20-50MB. Total steady state: ~350-400MB. With 2 Gunicorn workers, peak usage approaches ~800MB-1GB (models partially shared via copy-on-write).

---

23. Advanced NLP Concepts

Q141. What is zero-shot classification? Why did you move away from it? Zero-shot classification repurposes NLI by creating synthetic hypotheses: "This text is about [label]." For example, labels ["support", "refute"] become hypotheses "This text is about support." This adds an abstraction layer — the model isn't directly comparing evidence to claim, but rather classifying the evidence-claim pair into predefined categories. Direct NLI is more semantically honest.

Q142. What is Named Entity Recognition (NER)? How does spaCy's NER work? NER identifies and classifies named entities in text (PERSON, ORG, GPE, DATE, etc.). spaCy's `en_core_web_sm` uses a CNN-based tagger trained on OntoNotes. For VeriFact, NER feeds query generation — entities in claims become search query components. spaCy's NER is ~95% accurate for standard entity types.

Q143. What is sentence boundary detection? Why use spaCy instead of splitting on periods? Splitting on "." fails for abbreviations ("U.S.A."), decimal numbers ("3.14"), and other edge cases. spaCy uses a dependency parser and statistical model to identify true sentence boundaries. This is critical for accurate sentence segmentation — a mis-segmented sentence would produce incorrect NLI input.

Q144. What are word embeddings vs. sentence embeddings? Word embeddings (Word2Vec, GloVe) map individual words to vectors. Meaningfully composing them for sentences (averaging, etc.) loses order and structural information. Sentence embeddings (SBERT) map entire sentences to single vectors, capturing overall semantic meaning. For fact-checking, sentence-level semantics are what matter.

Q145. What is semantic textual similarity (STS)? How is your system evaluated against it? STS measures how semantically similar two sentences are (0-5 scale in benchmarks). SBERT is trained/evaluated on STS benchmarks (STS-B: Spearman correlation ~78% for MiniLM). VeriFact uses STS implicitly — cosine similarity between claim and evidence sentences IS an STS computation. Higher STS benchmark performance → better evidence filtering.

---

24. Architecture & Design Patterns

Q146. What is the Singleton pattern? Where is it used in your system? Singleton ensures only one instance of a class/object exists. Used in: (1) `model_registry` — one instance each of DeBERTa, SBERT, spaCy, Groq client, (2) `web_search` — one Tavily client, one Brave session, (3) `scraper` — one requests.Session. Thread-safe via double-checked locking with `threading.Lock`.

Q147. What is the Strategy pattern? How does query generation implement it? The Strategy pattern defines a family of algorithms and makes them interchangeable. `generate_queries()` implements two strategies: LLM decomposition (primary) and NER-based expansion (fallback). The controller doesn't know which strategy is used — it gets the same output (list of query strings). This enables adding new strategies without changing the controller.

Q148. What is the Chain of Responsibility pattern? How does web search implement it? The Chain of Responsibility passes a request through a chain of handlers until one succeeds. `web_search()` implements this: Tavily → (if fails) → Brave → (if fails) → DuckDuckGo. Each "handler" tries the search and returns results if successful, or passes to the next handler if it fails.

Q149. What is graceful degradation? Give three examples from your system. (1) No Groq API key → falls back to NER queries + rule-based explanations, (2) Tavily fails → Brave → DuckDuckGo, (3) scraping fails for a URL → skip and continue with remaining URLs. The system always produces *some* result, even if less optimal.

Q150. What is the Observer pattern? How would you use it to add logging/monitoring? The Observer pattern notifies subscribers of events. Currently, logging is inline (scattered `logger.info()` calls). An Observer pattern would define events (CLAIM_EXTRACTED, SEARCH_COMPLETED, VERDICT_COMPUTED) and let multiple observers react — one for logging, one for metrics, one for alerts. This would decouple observation from business logic.

---

25. Deep Domain Knowledge

Q151. What is the LIAR dataset? Why is it insufficient for your use case? LIAR contains 12.8K human-labeled claims from PolitiFact with 6 labels (pants-fire to true). Limitations: (1) only political claims, (2) claims from 2007-2016 are outdated, (3) labels reflect editorial judgment, not evidence-based reasoning, (4) training on LIAR teaches pattern matching, not verification. VeriFact's approach is topic-agnostic and never requires labeled data.

Q152. What is the FakeNewsNet dataset? How does it differ from LIAR? FakeNewsNet includes news articles (not just claims) with social context (user engagement data) from PolitiFact and GossipCop. It's larger and includes full article text. But like LIAR, it's a static snapshot — claims from its creation period. VeriFact doesn't need training data because it retrieves live evidence and reasons about it.

Q153. What is the ClaimReview schema? How could VeriFact integrate with it? ClaimReview is a schema.org markup that fact-checkers use to annotate their verdicts in machine-readable format. Google uses it for Fact Check panels. VeriFact could: (1) check ClaimReview databases before running its pipeline (if already fact-checked, return that), (2) publish its verdicts in ClaimReview format for consumption by other tools.

Q154. What is stance detection vs. rumor detection vs. fact verification? Stance detection: does text A support/refute/discuss text B? Rumor detection: is this claim a rumor (likely unverified)? Fact verification: is this claim true or false based on evidence? VeriFact combines stance detection + evidence retrieval for fact verification. It doesn't classify claims as "rumors" — it verifies them against evidence.

Q155. What is the "semantic gap" problem in NLI? Give an example. When two texts share vocabulary and topic but have opposite meanings. "Trump was assassinated" and "Trump survived an assassination attempt" have high lexical overlap (Trump, assassination) and high semantic similarity, but opposite truth values. Standard NLI can conflate these. My outcome mismatch detection addresses this specific gap.

---

26. API Design & REST Principles

Q156. Why REST API over GraphQL for this system? REST is simpler for a single-resource API (claims). GraphQL shines when clients need flexible queries over multiple related resources. VeriFact has one primary operation (check a claim) with a fixed response structure. REST's simplicity reduces development and debugging overhead.

Q157. Why JSON over XML for API responses? JSON is lighter-weight, natively supported by JavaScript, and the de facto standard for modern APIs. XML would be unnecessarily verbose for this use case. JSON also maps directly to Python dictionaries, simplifying serialization.

Q158. What HTTP status codes does your API return? When? 200 (success), 400 (invalid input — bad JSON, missing fields, invalid URL), 429 (rate limit exceeded), 500 (internal error). The system never returns 201 (no resource creation) or 404 (API paths are fixed, not dynamic).

Q159. How would you version your API? Currently not versioned (implicit v1). Proper versioning would use URL prefix (`/api/v1/check`) or header-based versioning (`Accept: application/vnd.verifact.v1+json`). URL prefix is simpler and more discoverable.

---

27. Concurrency & Thread Safety

Q160. Explain thread safety issues in Python. What is the GIL? The GIL (Global Interpreter Lock) allows only one thread to execute Python bytecode at a time. However, it's released during I/O operations (network calls, file reads). This means threading helps for I/O-bound work (scraping, API calls) but not for CPU-bound work (pure Python computation). Fortunately, PyTorch releases the GIL during inference, making threads useful for DeBERTa too.

Q161. Why are thread-safe metrics counters necessary? `REQUEST_COUNT` and `ERROR_COUNT` in `app_flask.py` are global integers modified by multiple Gunicorn threads. Without `threading.Lock`, concurrent increments could cause lost updates (read-modify-write race condition). The `_metrics_lock` ensures atomic increments.

Q162. What is a race condition? Where could one occur in your system? A race condition occurs when two threads access shared state concurrently. Potential locations: (1) model initialization (two threads both see model=None and both try to load) — solved by double-checked locking, (2) metrics counters — solved by lock. (3) LRU cache updates — Python's `@lru_cache` is thread-safe by default.

---

28. Docker & Containerization

Q163. What is multi-stage Docker build? Should your Dockerfile use it? Multi-stage builds use separate build and runtime images to reduce final image size. Currently not used because the build stage (pip install) doesn't produce large artifacts to exclude. However, if compiling C extensions, a multi-stage build could exclude gcc/g++ from the final image, saving ~200MB.

Q164. Why `--no-cache-dir` in `pip install`? `--no-cache-dir` prevents pip from storing downloaded packages in a cache directory. Since the Docker image is built once and cached, the pip cache wastes space inside the image. Removing it reduces image size by ~100-200MB.

Q165. What is the NLTK data download in the Dockerfile? Why is it done at build time? NLTK tokenizers (`punkt`, `stopwords`, `punkt_tab`) are ~5MB data packages needed for sentence splitting. Downloading at build time (in the Dockerfile's `RUN` command) ensures they're available when the container starts, avoiding runtime downloads. The `NLTK_DATA` environment variable tells NLTK where to find them.

---

29. Additional Expert Questions

Q166. What is "hallucination" in AI? How does VeriFact minimize it? Hallucination is when an AI generates plausible but factually incorrect information. VeriFact minimizes it by: (1) never generating facts — it only retrieves and reasons about existing content, (2) the NLI model classifies relationships, not generates text, (3) the LLM is constrained to summarization with provenance, (4) every verdict traces to specific source URLs and sentences.

Q167. What is the cold start problem? How does the `/api/warmup` endpoint address it? Cold start: the first request after deployment triggers model downloads and loading (~3 minutes), causing a timeout. The `/api/warmup` endpoint pre-loads all models (spaCy, DeBERTa, SBERT) and runs verification tests (NLI prediction, SBERT encoding). Called once after deployment, it ensures subsequent requests are fast.

Q168. What is model quantization? Could it help your system? Quantization reduces model precision (FP32 → INT8/FP16), shrinking size by 2-4× and increasing inference speed by 2-3×. DeBERTa-v3-small could be quantized from ~180MB to ~45MB with ~1% accuracy loss. This would help on memory-constrained deployments. PyTorch provides `torch.quantization` for post-training quantization.

Q169. What is ONNX? How could it improve inference performance? ONNX (Open Neural Network Exchange) is a model format that enables optimized inference with ONNX Runtime. Converting DeBERTa and MiniLM to ONNX could improve CPU inference by 30-50% through graph optimizations, operator fusion, and memory planning. This would be a drop-in replacement in `model_registry.py`.

Q170. What is ensemble learning? Could you ensemble multiple NLI models? Ensemble combines predictions from multiple models. Running both DeBERTa-v3 and BART-MNLI, then averaging scores, would improve accuracy by ~2-3% (error reduction via diversity). Trade-off: double the memory and compute. On a 4GB instance, this isn't feasible, but on GPU it would be worthwhile.

Q171. How would you implement A/B testing for verdict accuracy? Route 50% of requests to the current pipeline, 50% to a modified pipeline (e.g., different thresholds, different model). Log both verdicts and actual outcomes (when available). Compare accuracy metrics after sufficient sample size. Requires: (1) traffic splitting middleware, (2) result logging, (3) delayed ground-truth labeling.

Q172. What is model drift? Could your NLI model's performance degrade over time? Model drift occurs when the data distribution changes and model accuracy degrades. Since DeBERTa-v3 is pre-trained on static MNLI data, its NLI capability doesn't drift. However, the *types* of claims it encounters may shift — new topics, new language patterns. The zero-shot nature provides resilience, but novel linguistic constructs could challenge it.

Q173. What is few-shot learning? Could it enhance your system? Few-shot learning provides a model with a few examples at inference time (via prompt). With the LLM tiebreaker, I could include 2-3 examples of correctly resolved claims in the prompt, improving accuracy. For DeBERTa, few-shot would require fine-tuning (not applicable for inference-only). Prompt-based few-shot via Groq is the practical path.

Q174. What is retrieval-augmented generation (RAG)? How does your system differ? RAG: retrieve documents → feed to LLM → generate answer. VeriFact: retrieve documents → NLI model reasons about them → deterministic verdict. Key difference: RAG's output depends on the LLM's generation (hallucination-prone). VeriFact's output depends on deterministic scoring (traceable, reproducible). The LLM is used only for optional summaries, not for verdict computation.

Q175. How would you handle claims about future events? "The economy will crash in 2027" — no evidence exists yet. Search returns predictions and opinions, not facts. NLI might detect entailment with opinion pieces, producing a potentially misleading LIKELY TRUE. Mitigation: detect temporal indicators (future tense), flag as inherently unverifiable, and override to UNVERIFIED or add a "prediction" disclaimer.

Q176. How does your system handle claims with multiple sub-claims? Currently, it extracts the single most important sentence as the claim. "India won the World Cup and Modi resigned" contains two independent claims. The system would verify only one (whichever scores highest). Improvement: decompose compound claims into individual assertions and verify each separately.

Q177. What is the difference between abstractive and extractive summarization? Which does your AI summary use? Extractive: selects existing sentences. Abstractive: generates new text. The LLM summary is abstractive — it synthesizes a new 2-3 sentence explanation from the evidence. The rule-based fallback is extractive — it uses a pre-written template with filled-in values. Abstractive is more natural but risks hallucination (constrained by the prompt).

Q178. What is TF-IDF? Why did you choose SBERT over TF-IDF for similarity? TF-IDF creates sparse vectors based on term frequency and inverse document frequency. It's efficient but purely lexical —

"automobile" and "car" have zero similarity. SBERT captures semantic meaning — "automobile" and "car" have high similarity. For fact-checking, semantic understanding is essential (evidence may use different words than the claim).

Q179. What is BM25? How does it compare to SBERT for evidence retrieval? BM25 is a probabilistic lexical retrieval model (improvement over TF-IDF). It's fast and effective for keyword matching. SBERT captures semantic similarity. Ideal systems combine both: BM25 for fast initial retrieval (high recall), SBERT for re-ranking (high precision). My system relies on search APIs (which likely use BM25 internally) for retrieval and SBERT for re-ranking.

Q180. What is the difference between fine-tuning and prompt engineering? Fine-tuning updates model weights on new data — permanent, requires training data and compute. Prompt engineering designs input prompts to guide model behavior — no weight changes, immediate, reversible. VeriFact uses prompt engineering for Groq (decomposition, tiebreaker, summary prompts) and neither for DeBERTa/SBERT (used as-is from pre-training).

---

30. System Failure & Recovery

Q181. What happens if DeBERTa model download fails during lazy loading? The `get_nli_model()` function in `model_registry` would raise an exception. Since `nli_predict()` is called inside `detect_stance()`, the exception propagates up. The `process_single_result()` in evidence_aggregator catches it and returns None for that evidence. If ALL evidence processing fails, the verdict defaults to UNVERIFIED.

Q182. What if Gunicorn workers crash repeatedly? Gunicorn automatically restarts crashed workers. `--max-requests 500` prevents gradual degradation. If workers crash on every request (e.g., OOM), Gunicorn enters a restart loop. Docker's `--restart unless-stopped` restarts the entire container. Persistent crashes require debugging via `docker logs`.

Q183. How do you handle network partitions (no internet access)? All search APIs fail → no evidence → UNVERIFIED verdict. The NLI and SBERT models run locally, so core ML inference works offline. The system handles this gracefully. If models aren't downloaded yet (first run without internet), model loading fails → all evidence processing returns None → UNVERIFIED.

Q184. What is your disaster recovery plan? (1) Docker image is on DockerHub — can be pulled to any EC2 instance, (2) source code is on GitHub — can rebuild from scratch, (3) no persistent state — the system is stateless, (4) CI/CD automates deployment. Recovery time: ~10 minutes (launch new EC2 + pull image + warm up models).

---

31. Final Expert-Level Questions

Q185. Compare your architecture to Google's Perspective API. Perspective API detects toxicity using supervised classifiers trained on annotated data. VeriFact verifies claims using retrieval + NLI. Different problems: Perspective asks "is this text toxic?" VeriFact asks "is this claim supported by evidence?" Perspective is a classifier; VeriFact is a verification pipeline.

Q186. What is the Winograd Schema Challenge? Does it relate to NLI? Winograd schemas test commonsense reasoning: "The trophy didn't fit in the suitcase because *it* was too big." NLI models struggle with such coreference-dependent reasoning. If a claim requires knowing who "it" refers to, DeBERTa may fail. VeriFact doesn't explicitly handle coreference resolution.

Q187. What is the difference between correlation and causation in the context of your evidence? "Countries with high ice cream sales have high drowning rates." Evidence supports correlation (both true), but not causation. NLI would say "entailment" for "ice cream sales correlate with drownings." The system can't distinguish correlation from causation — it reports what evidence says, not what it means.

Q188. How would you integrate knowledge graphs (e.g., Wikidata) into your system? Knowledge graphs provide structured factual data. Integration: (1) for entity-based claims, query Wikidata for verified facts, (2) cross-reference relationship claims against knowledge graph triples, (3) use knowledge graph entities to improve query generation. This would strengthen the relationship claim module significantly.

Q189. What is federated learning? Could it help improve your system across deployments? Federated learning trains models across multiple deployments without sharing data. Multiple VeriFact instances could collaboratively learn better confidence calibration or source credibility weights without sharing user claims. Practical challenges: VeriFact is inference-only, so there's no training to federate without architectural changes.

Q190. What is the attention sink phenomenon? Does it affect your models? In long sequences, attention models disproportionately attend to the first few tokens regardless of their importance. For my system, input lengths are short (sentence pairs, ~50 tokens) — attention sink is negligible. It would become relevant if processing full articles through the cross-encoder.

Q191. Why not use a graph neural network (GNN) for evidence aggregation? GNNs model relationships between nodes. Evidence sources could be nodes, with edges representing cross-references or contradictions. However: (1) the number of evidence sources is small (3-5), (2) inter-evidence relationships aren't currently modeled, (3) a weighted sum is simpler and interpretable. GNNs would be overkill for the current scale.

Q192. What is the information bottleneck theory? How does it apply to your embeddings? It posits that optimal representations compress input while preserving task-relevant information. SBERT's 384-dim embeddings are a compressed representation of sentence meaning — they discard syntax and style while preserving semantic content. The cosine similarity on these embeddings effectively measures how much task-relevant (semantic) information is shared.

Q193. How would you handle multi-modal claims (text + image)? Currently out of scope. To add: (1) image-based claims → CLIP or BLIP for image understanding, (2) reverse image search for provenance, (3) image-text consistency checking. The pipeline architecture supports this — add an image processing module before or parallel to text processing.

Q194. What is contrastive learning? How were SBERT embeddings trained? Contrastive learning trains by pulling similar pairs together and pushing dissimilar pairs apart in embedding space. SBERT uses a siamese network trained on NLI pairs: entailment pairs should be close, contradiction pairs should be far. The resulting embeddings capture semantic similarity by construction.

Q195. Explain the no-free-lunch theorem. How does it apply to your model choices? No single model is optimal for all problems. DeBERTa-v3 excels at NLI but fails at numerical reasoning. SBERT excels at similarity but not entity verification. My system uses multiple specialized models rather than one general model — each model is used where it's strongest: SBERT for similarity, DeBERTa for NLI, spaCy for NER.

Q196. What is explainable AI (XAI)? How does VeriFact achieve it? XAI makes AI decisions interpretable to humans. VeriFact achieves explainability through: (1) every evidence source is shown with URL, (2) stance labels are visible per source, (3) similarity scores show relevance, (4) credibility weights show source trustworthiness, (5) the 5-step explanation chain shows reasoning, (6) AI summary translates to plain language.

Q197. How would you evaluate your system against human fact-checkers? Design: (1) create a test set of 100+ claims with known verdicts (from PolitiFact/Snopes), (2) run VeriFact on each, (3) compare verdicts against human labels, (4) measure accuracy, precision, recall, F1 per verdict category. Expected: high accuracy on clear-cut claims, lower on nuanced/contextual claims. Human fact-checkers will outperform on claims requiring deep domain knowledge.

Q198. What is data poisoning? Could an attacker manipulate your system's evidence? Data poisoning involves corrupting training data to influence model behavior. VeriFact doesn't train, but an attacker could: (1) create many web pages with false information to dominate search results, (2) manipulate search engine rankings (SEO poisoning). VeriFact's source credibility scoring partially mitigates this by weighting authoritative sources higher.

Q199. What is the Turing test? Can your system pass it in the fact-checking domain? The Turing test asks if a human can distinguish AI from human output. VeriFact's AI summary is designed to sound like a journalist's explanation. For straightforward claims, it likely passes. For complex claims requiring nuance, human fact-checkers provide richer context. The system's verdicts are structured (not conversational), which would reveal its AI nature.

Q200. If you had unlimited resources, what would the ideal version of VeriFact look like? (1) GPU cluster for real-time inference (<1s), (2) multilingual support (50+ languages), (3) multimodal (text + image + video), (4) knowledge graph integration (Wikidata, DBpedia), (5) dynamic source credibility (learned, not static), (6) claim decomposition for compound claims, (7) temporal reasoning for time-sensitive claims, (8) adversarial robustness testing, (9) federated deployment for global coverage, (10) human-in-the-loop verification for high-stakes claims.

BONUS: Rapid-Fire Questions

Q201. What Python version? → 3.10+ **Q202.** What web framework? → Flask with Gunicorn (WSGI) **Q203.** What ORM? → None — no database **Q204.** How many Transformer models? → 3 (DeBERTa, MiniLM, spaCy's CNN) **Q205.** What is the embedding dimension? → 384 **Q206.** What is the NLI accuracy? → 88% on MNLI **Q207.** What is the similarity threshold? → 0.25 **Q208.** What is the verdict threshold? → ±0.35 **Q209.** How many search API tiers? → 3 (Tavily, Brave, DDG) **Q210.** How many credibility tiers? → 4 (1.5 → 1.3 → 1.0 → 0.3-0.6) **Q211.** What is the temperature scaling factor? → T=1.3 **Q212.** What is the MMR lambda? → λ=0.7 **Q213.** How many ThreadPool workers? → 3 **Q214.** What is the scraping timeout? → 8 seconds **Q215.** What is the Gunicorn timeout? → 120 seconds **Q216.** What is the rate limit? → 5/min per IP, 100/hour global **Q217.** What Docker base image? → python:3.10-slim **Q218.** What CI/CD? → GitHub Actions **Q219.** What cloud provider? → AWS EC2 **Q220.** What LLM? → Llama-3.3-70b-versatile via Groq

---

Document contains 220 questions covering NLP/NLI theory, Transformer architecture, system design, all pipeline stages, deployment, security, ML theory, ethics, performance optimization, and expert-level scenario analysis.