Optimizing Span-Level Semantic Extraction for Generative Video Prompts: A Comprehensive Technical Analysis and Architectural Roadmap

Executive Summary

The rapid evolution of generative Artificial Intelligence has precipitated a fundamental paradigm shift in the domain of video synthesis. The transition from static text-to-image generation to dynamic text-to-video (T2V) synthesis—exemplified by state-of-the-art models such as OpenAI's Sora, Google's Veo, and RunwayML's Gen-3—has fundamentally altered the requirements for user interaction and prompt engineering. Unlike their text-based predecessors, where ambiguity can occasionally yield serendipitous results, T2V models operate under a stricter modality that necessitates a transition from loose, conversational requests to structured, cinematic briefings. These models function less like chatbots and more like virtual production crews, requiring precise, technical, and visually grounded instructions regarding shot composition, subject dynamics, lighting conditions, and camera behavior to maintain spatio-temporal coherence and avoid hallucinations.[1]

This report presents an exhaustive technical analysis of a span labeling system designed to mediate this interaction by extracting structured semantic metadata from user inputs. The system serves as a critical middleware layer, translating raw user intent into a taxonomy-compliant JSON payload that informs downstream video generation policies. However, our analysis reveals that the current architecture—reliant on dynamic prompt construction, regex-based validation, and post-hoc repair—is inherently fragile. It faces significant challenges related to prompt-policy drift, taxonomy misalignment, and the "Visual-Semantic Gap"—the discrepancy between what a word signifies in a linguistic context and how a diffusion model interprets it in a visual latent space.[1]

Specifically, the reliance on generative models for direct character offset prediction introduces a high risk of hallucination and misalignment, undermining the integrity of the

extracted indices. Furthermore, the system struggles with disambiguation conflicts where linguistic synonyms (e.g., "pan") map to distinct visual operations (e.g., *Subject Action* vs. *Camera Movement*), often violating the "One Clip, One Action" principle critical for temporal coherence.[1]

To address these failure modes, this report proposes a multi-tiered architectural evolution. We recommend transitioning from the current "generate-then-repair" pipeline to a **Schema-First, Disambiguation-Aware Architecture**. This involves adopting **Grammar-Constrained Decoding (GCD)** to enforce valid JSON outputs via strict schema adherence, replacing naive offset prediction with **Anchored Substring Extraction** coupled with fuzzy alignment, and implementing a **Dynamic Few-Shot Retrieval System** to resolve domain-specific ambiguities. This roadmap prioritizes low-effort, high-impact interventions to stabilize the current system while laying the groundwork for advanced, agentic extraction workflows capable of navigating the complexities of the "Director's Lexicon".[1]

---

1. Contextual Analysis: The Deterministic Imperative in Generative Video

1.1 The Paradigm Shift: From Chatbots to Virtual Directors

To understand the specific engineering challenges of the span labeling system, one must first contextualize the unique demands of the Text-to-Video (T2V) domain. We are witnessing a transition from a regime of manual rendering, where visual fidelity was the primary constraint, to a regime of prompt-mediated synthesis, where the limiting factor is the user's ability to articulate high-dimensional visual concepts in natural language.[1]

In text-to-text or even text-to-image generation, the model often acts as a creative partner, filling in unstated details with plausible defaults. However, video introduces the dimension of *time*, which exponentially increases the complexity of coherence. T2V models are deterministic simulation engines that must maintain object identity, physics, and narrative logic across hundreds of frames. Ambiguity in the prompt—such as failing to specify whether the "camera" or the "subject" is moving—can lead to "morphing" artifacts, temporal flickering, or a complete breakdown of the scene geometry.[1]

Therefore, the "Prompt Builder" tool and its underlying span extraction logic are not merely

syntactic parsers; they are **semantic enforcement engines**. They must bridge the gap between the user's vague intent (e.g., "make it look cool") and the model's need for explicit technical parameters (e.g., "Shot Type: Wide, Lighting: Volumetric, Style: Cyberpunk"). This requires a deep integration of **computational linguistics** (to parse intent) and **computer vision semantics** (to map text to visual attributes).[2]

1.2 System Deconstruction: The Current Architecture

The system under analysis operates as a middleware service within a Node.js/TypeScript environment. Its primary function is to accept a user's raw text input and return a structured list of "spans"—segments of text categorized into specific roles (Shot, Subject, Action, etc.) with precise start and end character indices.

The current architecture follows a **Dynamic Prompt Construction** pattern:

1. **Prompt Assembly:** The backend dynamically concatenates a base system prompt (span-labeling-prompt.md), a taxonomy definition (shared/taxonomy.js), and runtime policy variables (e.g., maxSpans).
2. **Probabilistic Extraction:** This monolithic prompt is sent to the aiService, instructing the Large Language Model (LLM) to identify spans and calculate their character offsets.
3. **Heuristic Validation:** The output JSON is passed through a SpanValidator pipeline. This component acts as a defensive layer, attempting to sanitize the probabilistic output using deterministic rules: normalizing data types, deduplicating overlapping spans, filtering low-confidence predictions, and enforcing word count limits.
4. **Repair Loop:** If validation fails (e.g., malformed JSON), the system triggers a retry mechanism, appending error feedback to the original prompt.

While this architecture is functional for simple tasks, it relies heavily on the assumption that an LLM can reliably perform arithmetic tasks (counting character indices) and strictly adhere to complex structural constraints within a single inference pass. Research suggests this is a flawed assumption for Transformer-based models, which are fundamentally probabilistic token predictors, not deterministic symbol manipulation engines.[3]

1.3 The Core Friction: The Visual-Semantic Gap

The central friction point in this system is the **Visual-Semantic Gap**.[1] This term describes the discrepancy between the linguistic meaning of a word and its visual manifestation in a

diffusion model's latent space.

In natural language, words are polysemous and context-dependent. The word "light" can refer to weight (Subject attribute), illumination (Lighting), or a color tone (Style). In the rigid taxonomy of video generation, however, these concepts must be orthogonal.

- **Subject:** "A feather."
- **Lighting:** "Soft, high-key lighting."
- **Style:** "Light, pastel color palette."

The current system attempts to map fluid natural language onto this rigid hierarchy without sufficient disambiguation logic. It often conflates "Shot Type" with "Camera Angle" or "Action" with "Camera Movement" because the linguistic markers (verbs like "moves," "pans," "zooms") overlap significantly. This leads to **Disambiguation Conflicts**, where a single phrase could validly belong to multiple categories depending on the interpretive lens, resulting in inconsistent downstream behavior.[1]

---

2. Failure Mode Phenomenology: Anatomy of Extraction Errors

A rigorous analysis of the system's operational constraints reveals five primary failure modes. These are not merely "bugs" but structural weaknesses inherent to the current architectural approach.

2.1 Failure Mode 1: Index Drift and Offset Hallucination

The most pervasive technical failure is **Index Drift**. The system requires the LLM to return start and end character indices for every extracted span. This requirement fundamentally misunderstands the internal architecture of Large Language Models.

Mechanism: LLMs do not "see" text as a sequence of characters; they process text as a sequence of **tokens**. A single token can represent a whole word (camera), a sub-word (ing), or even a concept. To return a character index, the model must internally:

1. Generate the token.
2. Recall the exact character length of all preceding tokens in the prompt.
3. Perform arithmetic addition to calculate the cumulative offset.

Transformers are notoriously poor at this type of internal arithmetic state tracking.[3] Consequently, they frequently hallucinate indices that are "approximately" correct but off by a few characters.

Manifestation:

- **Token Boundary Mismatch:** A span might start in the middle of a token. If the model tokenizes "sunlight" as ["sun", "light"], it might correctly identify "light" but fail to calculate the offset if "sunlight" was tokenized as a single unit in a different context.
- **Unicode Variance:** The presence of multi-byte characters (emojis, non-Latin scripts) exacerbates this issue. If the backend calculates string length using UTF-16 code units (common in JavaScript) while the model counts bytes or unicode code points, the indices will drift progressively further apart as the string gets longer.
- **The "Off-By-One" Hallucination:** The model might return indices `` for a 6-character word, leading to backend validation errors or frontend highlighting glitches (e.g., highlighting "the ca" instead of "car").

Impact: The SpanValidator relies on these indices to extract the substring for validation. If the indices are wrong, the validator checks the wrong text, leading to false negatives (valid spans being rejected) or false positives (nonsense spans being accepted).

2.2 Failure Mode 2: Taxonomy Ambiguity and Category Confusion

The second critical failure mode is the system's inability to reliably map user text to the correct semantic category within the Universal Prompt Framework.1 This framework establishes a strict hierarchy:
$$ \text{Shot} > \text{Subject} > \text{Action} > \text{Setting} > \text{Camera} > \text{Lighting} > \text{Style} $$
Mechanism: The current prompt likely lists these categories as flat options. Without explicit **Semantic Role Labeling (SRL)** instructions, the model defaults to surface-level keyword matching.[2]

- **Shot vs. Camera Conflict:** The term "Close-up" is a **Shot Type** (Tier 1) because it defines the spatial boundary of the scene. However, "Zoom in" is a **Camera Movement** (Tier 5) or Lens operation. Users—and models—often conflate these. If the system labels "Close-up" as a Camera Movement, the video model might attempt to *move* the camera into a close-up during the shot, rather than *starting* in a close-up, fundamentally altering the composition.[1]
- **Action vs. Movement Conflict:** The verb "runs" usually denotes a **Subject Action**. However, in the phrase "The camera runs along the track," "runs" describes the **Camera Behavior**. A naive extractor sees the verb "runs" and labels it as *Action*, potentially

assigning the camera's motion to the subject or vice versa. This is a failure of *Agent detection*—identifying *who* is performing the action.

- **Subject vs. Environment:** In "A man standing in a crowded room," "crowded room" is the **Environment**. In "A crowd cheering," "crowd" is the **Subject**. Failure to distinguish the active agent from the passive setting leads to "Attribute Bleed," where background elements are rendered with the focus and detail reserved for the main subject.

2.3 Failure Mode 3: Prompt-Policy Drift

The decoupling of the extraction logic (code) from the extraction instruction (prompt) creates a persistent risk of **Prompt-Policy Drift**.

Mechanism: The system policies (e.g., maxSpans=60, allowOverlap=false) are defined in TypeScript, but the prompt template is a static Markdown file.

- **Instruction Divergence:** If a developer updates the TypeScript policy to allow overlaps for advanced users but forgets to update the hardcoded "Do not allow overlapping spans" instruction in the Markdown template, the LLM receives contradictory signals. Research indicates LLMs often prioritize instructions found in few-shot examples or explicit system messages over implicit constraints, leading to unpredictable behavior.[5]
- **Taxonomy Lag:** If taxonomy.js is updated with a new category (e.g., "Physics" or "Audio"), but the few-shot examples in the prompt are not updated to demonstrate this new category, the model will likely ignore it. Zero-shot performance on novel categories is significantly lower than few-shot performance.[6]

2.4 Failure Mode 4: The "One Clip, One Action" Violation

A specific and critical requirement for high-fidelity video generation is the **"One Clip, One Action"** principle.[1] Video diffusion models degrade rapidly when prompts describe complex, sequential narratives (e.g., "The man runs **and then** jumps") within a single generation window.

Mechanism: The current extractor treats "runs" and "jumps" as two valid *Action* spans. It successfully extracts them, but in doing so, it validates a prompt that is destined to fail at the generation stage. The extractor acts as a passive tagger rather than an active critic.

- **Temporal Blindness:** The system lacks the logic to identify temporal markers like "and then," "after," or "subsequently." It fails to flag these as violations of the "One Clip"

constraint.
- **Impact:** This results in "morphing" artifacts where the subject inexplicably transforms or the motion becomes incoherent as the model attempts to superimpose two distinct temporal states.[1]

2.5 Failure Mode 5: Context Fragmentation via Chunking

For prompts exceeding the token limit (e.g., >400 words), the system likely employs a naive chunking strategy (splitting text into blocks).

Mechanism: Simple splitting ignores semantic boundaries.

- **Boundary Artifacts:** If the phrase "high contrast lighting" is split across two chunks (e.g., "high contrast" | "lighting"), the system will extract two partial, potentially meaningless spans. "High contrast" might be labeled *Style*, and "lighting" might be discarded or labeled *Lighting*, losing the specific attribute binding.
- **Context Loss (Coreference Resolution):** An entity defined in Chunk 1 (e.g., "The astronaut") might be referred to as "she" in Chunk 2. Without access to the context of Chunk 1, the model processing Chunk 2 labels "she" as a generic Subject, losing the specific visual attributes (spacesuit, helmet) associated with "astronaut." This leads to character inconsistency across the generated video segments.[7]

---

3. Theoretical Framework & Deep Research

To address these failure modes, we must ground our solution in the latest research from Information Extraction (IE), Natural Language Processing (NLP), and Computer Vision.

3.1 Structured Extraction: From Regular Expressions to Grammar Constraints

The era of parsing raw LLM output with Regular Expressions is ending. The current state-of-the-art relies on **Grammar-Constrained Decoding (GCD)** or **Native Function Calling**.

- **Function Calling (Tool Use):** Models like GPT-4o and Gemini 1.5 have been fine-tuned to detect when to "call a function." By defining the extraction task as a tool—e.g., extract_spans(spans: List)—we shift the burden of formatting from the prompt (which is probabilistic) to the model's specialized "agentic" head.[8] This significantly reduces syntax errors and hallucinated fields.
- **Structured Outputs (JSON Schema):** Newer API features (e.g., OpenAI Structured Outputs) enforce strict schema adherence at the decoding level. This works by masking invalid tokens during the generation process. If the schema expects an Enum for Category (e.g., ``), the model is mathematically prevented from generating any token that does not fit one of those strings.[10] This provides a guarantee of type safety that post-hoc validation cannot match.

Implication: Moving to Schema-Enforced generation eliminates the "JSON/format failures" and the need for complex, fragile repair loops.

3.2 The Pivot to Anchored Extraction

Research into **Named Entity Recognition (NER)** with LLMs [3] strongly suggests that asking for character offsets is an anti-pattern.

- **The "Anchor" Method:** Instead of predicting indices ``, the robust approach is to ask the model to return the **exact substring** ("quote") from the text. The system then locates this substring in the original text using deterministic string matching algorithms.[11]
- **Fuzzy Alignment:** Because LLMs are generative, they may introduce minor alterations to the quote (e.g., "don't" vs. "dont", or skipping a comma). To handle this, we must employ **Fuzzy Matching** algorithms (like Levenshtein distance, implemented in libraries such as rapidfuzz) to align the generated quote with the source text. This decouples the *semantic* task (identifying the span) from the *mechanical* task (counting characters), playing to the LLM's strengths while mitigating its arithmetic weaknesses.[12]

3.3 Semantic Role Labeling and the Director's Lexicon

To solve the disambiguation conflicts, we must leverage **Semantic Role Labeling (SRL)** theory. In linguistic terms, every action has an *Agent* (the doer) and a *Patient* (the receiver).

- **Agent-Based Disambiguation:** In the phrase "Camera follows the subject," the *Camera* is the Agent. In "Subject follows the path," the *Subject* is the Agent. The extraction logic

must be sensitive to these roles.

- **The Director's Lexicon:** Video prompting requires a specialized vocabulary. Research confirms that T2V models respond best to technical cinematography terms.[1]
  - **Camera Movement:** *Truck, Dolly, Pan, Tilt, Roll, Pedestal, Rack Focus*.
  - **Lighting:** *Chiaroscuro, Volumetric, High-key, Low-key, Rim lighting*.
  - **Lens:** *85mm (Portrait), 24mm (Wide), Anamorphic, Bokeh*.
  - **Shot Type:** *Extreme Close-Up (ECU), Medium Shot (MS), Cowboy Shot, Establishing Shot*.

The system must be explicitly trained (via few-shot examples) to recognize and categorize these terms correctly. If a user writes "zoom," the system should identify it as a *Camera/Lens* operation, not a *Subject Action*.

3.4 Evaluation Metrics for Imprecise Spans

Standard "Exact Match" metrics (Precision/Recall based on perfect index overlap) are too harsh for span extraction.

- **Relaxed Span Matching:** Literature on IE evaluation suggests using **Relaxed Match** or **Soft F1** scores.[14] If the Ground Truth span is "the red car" and the model extracts "red car", an Exact Match metric gives a score of 0. A Relaxed Match metric, however, recognizes the semantic overlap.
- Intersection over Union (IoU): We can quantify this overlap using IoU.
  $$ IoU = \frac{\text{Intersection}(\text{Predicted}, \text{Truth})}{\text{Union}(\text{Predicted}, \text{Truth})} $$
  An IoU threshold of > 0.5 is typically considered a true positive in computer vision and can be adapted here to credit the model for identifying the correct concept even if the boundaries are slightly off.

3.5 Handling Long Contexts: Sliding Windows

To address chunking artifacts, we look to **Sliding Window** techniques used in RAG (Retrieval-Augmented Generation).[16]

- **Overlap Strategy:** Instead of disjoint chunks, we use overlapping windows (e.g., 500 tokens with a 100-token overlap).
- **Merge Logic:** Spans detected in the overlap region are candidates for merging. If "neon sign" is detected in Window A and Window B, they are deduplicated. If "neon" is in

Window A and "sign" is in Window B (split by the boundary), the overlap ensures a third extraction opportunity covers "neon sign" intact. This preserves semantic continuity across boundaries.

---

4. Concrete Architectural Solutions

Based on this theoretical framework, we propose three distinct designs. These are not mutually exclusive but represent a maturity model, moving from stabilization to advanced agentic capabilities.

4.1 Design A: The "Aligned Baseline" (Immediate Stabilization)

Objective: Resolve Index Drift and Prompt-Policy Drift with minimal refactoring. This design focuses on stabilizing the current "generate-then-repair" loop without introducing complex new dependencies.

Implementation Details:

1. **Switch to Quote-Based Extraction:**
   - **Prompt Change:** Modify span-labeling-prompt.md to instruct the LLM: *"Do not calculate indices. Return the exact substring text from the prompt."*
   - **Backend Logic:** Implement a SpanAligner service.
     - Receive {"text": "red car", "label": "Subject"}.
     - Execute prompt.indexOf("red car") to find the start index.
     - Calculate end = start + "red car".length.
     - **Fallback:** If indexOf returns -1 (due to hallucinated punctuation), use rapidfuzz.process.extractOne() to find the closest fuzzy match in the source text with a high score threshold (e.g., >90).
2. **Dynamic Template Rendering:**
   - **Template Engine:** Replace simple string concatenation with a logic-aware template engine like **Handlebars** or **Mustache**.
   - **Injection: Pass the policy object directly into the template at runtime.**

Constraints

- You must extract no more than {{policy.maxSpans}} spans.
  {{#if policy.allowOverlap}}
- Overlapping spans are permitted.
  {{else}}
- Do NOT generate overlapping spans.
  {{/if}}

  - **Benefit:** This ensures the system instructions *always* perfectly mirror the validation logic, eliminating drift.
3. **Simplified Taxonomy Injection:**
   - Flatten the nested taxonomy.js into a simplified list of "Category: Definition" pairs for the prompt context. This reduces token usage and cognitive load on the LLM compared to dumping a massive JSON object.

4.2 Design B: The "Disambiguation-Aware" Architect (Precision Focus)

Objective: Address the Visual-Semantic Gap and reduce Category Confusion. This design introduces "Semantic Routing" and "Two-Pass Logic."

Implementation Details:

1. **Semantic Role-Based Few-Shot Retrieval:**
   - **Vector Store:** Create a curated database of ~50 "hard case" examples that specifically address ambiguity (e.g., "Camera pans" vs. "Frying pan", "Tracking shot" vs. "Tracking a package").
   - **Retrieval Mechanism:** At runtime, generate an embedding of the user's prompt. Query the vector store to find the 3 most semantically similar examples.
   - **Dynamic Injection:** Inject these specific examples into the "Few-Shot" section of the system prompt. This provides the model with **Dynamic Context** relevant to the specific ambiguity present in the user's input.[18]
2. **Hierarchical Classification (CoT) Prompt:**
   - **Concept:** Instead of asking for a flat list, structure the prompt to force a logical dependency check, mimicking the Universal Prompt Framework.
   - **Instruction:**"Step 1: Identify the Primary Subject and their Actions.
     Step 2: Identify the Environment (background) surrounding the subject.
     Step 3: Identify Cinematic Controls (Camera, Lighting, Style) that apply to the frame."
   - **Benefit:** This effectively forces the model to distinguish between the *content* of the

scene (Subject/Action) and the *presentation* of the scene (Camera/Lighting), reducing Attribute Bleed.

3. **Negative Constraints:**
    - Explicitly list "Anti-Patterns" in the prompt based on common failure modes.
        - *Rule:* "IF a span describes a lens (e.g., '35mm', 'Anamorphic'), label as TECHNICAL, NOT STYLE."
        - *Rule:* "IF a span describes a time of day (e.g., 'Golden Hour'), label as LIGHTING or ENVIRONMENT, NOT SUBJECT."

4.3 Design C: Schema-First Agentic Extraction (State-of-the-Art)

Objective: Maximize reliability and handle complex/long prompts using agentic workflows. This represents the target architecture for a production-grade system.

Implementation Details:

1. **Native Structured Outputs (Zod/Pydantic):**
    - **Mechanism:** Use the LLM provider's native "Structured Output" or "JSON Mode."
    - **Schema Integration:** Use a library like zod-to-json-schema [20] to convert the TypeScript Taxonomy interface directly into the JSON Schema passed to the LLM API.
    - **Benefit:** This guarantees that the LLM's output strictly conforms to the TypeScript types defined in the codebase. It eliminates 100% of JSON syntax errors and type mismatches, rendering the "Repair Loop" obsolete for format issues.
2. **Overlap-Aware Sliding Window:**
    - **Chunking:** Implement a sliding window of 500 tokens with a 100-token overlap.
    - **Deduplication Strategy:**
        - If a span is detected entirely within the overlap region of two windows, merge them.
        - If the labels conflict (e.g., Window A says "Subject", Window B says "Action"), prioritize the label from the window where the span is more centrally located (as context is better in the center of the window) or defer to a confidence score.[16]
3. **The "Critic" Loop (Semantic Repair):**
    - **Logic:** Instead of a generic repair, implement a targeted **Critic Agent**.
    - **Trigger:** If a span has low confidence (<0.7) or violates a semantic heuristic (e.g., identifying "Daylight" as a Subject).
    - **Critic Prompt:** *"The system labeled 'Daylight' as a Subject. This is likely an Environment or Lighting attribute. Please verify the Universal Prompt Framework rules and correct if necessary."*.[21]
    - **Benefit:** This moves validation from simple syntax checking to semantic consistency

checking.

4. **One-Clip Policy Enforcer:**
    - The agent is explicitly instructed to scan for temporal markers ("and then," "later," "after").
    - **Output:** The JSON schema includes a temporal_violation boolean flag. If true, the frontend can immediately warn the user: *"Your prompt describes a sequence of events. Video models work best with a single continuous action. Consider splitting this into two clips."*

---

5. Evaluation and Experimentation Plan

To transition from "vibes-based" engineering to data-driven optimization, we must establish a rigorous evaluation harness.

5.1 The Golden Set Protocol

We cannot rely on live user data for evaluation due to privacy concerns and the lack of ground truth. We must curate a **Golden Set** of 100-200 representative prompts.

- **Dataset Composition:**
    - **Core (50%):** Standard prompts following the Subject + Action + Setting structure.
    - **Technical (20%):** Prompts heavy on the "Director's Lexicon" (focal lengths, lighting ratios, camera moves) to test Disambiguation accuracy.
    - **Edge Case - Overload (15%):** Prompts with >60 spans or >400 words to stress-test Chunking and Truncation logic.
    - **Adversarial (15%):** Prompts with intentional semantic ambiguity (e.g., "The camera flies," "A light feeling," "Pan the cooking pan").
- **Annotation Methodology:** Utilize a **Human-in-the-Loop** approach. Run the current best model (Design C) to pre-label the dataset, then employ domain experts (videographers or prompt engineers) to manually correct the labels and boundaries. This creates the Ground Truth.[23]

5.2 Metrics Definition

We define a set of metrics designed to capture the nuance of span extraction performance [14]:

Metric	Definition	Target Goal
Relaxed Span F1	A match is counted if the Intersection over Union (IoU) > 0.5 AND the extracted Label matches the Ground Truth. This penalizes missed spans and wrong labels but forgives minor boundary drift.	> 0.85
Boundary Precision	The average character distance between the Predicted Start index and the True Start index (for matched spans).	< 3 chars
Category Accuracy	For spans that match spatially, what percentage have the correct Category? This specifically measures Disambiguation performance (e.g., distinguishing Shot vs. Camera).	$> 90\%$
Hallucination Rate	The percentage of predicted spans that have NO overlap (IoU < 0.1) with any ground truth span.	$< 5\%$
JSON Validity Rate	The percentage of API responses that parse successfully without triggering the "Repair Path."	$> 99\%$

5.3 A/B Testing Methodology

1. **Offline Replay:** Before deploying any change to the prompt or logic, replay the full 200-item Golden Set against the new configuration. Automated scripts calculate the Relaxed F1 and Category Accuracy scores. Any regression >2% blocks deployment.
2. **Shadow Mode (Online):** Deploy the new system (e.g., Design B) alongside the old one in production.
    - The user sees the result from the **Live (Old)** model.
    - The **Shadow (New)** result is generated and logged silently.
    - **Metric:** Compare the **Validator Rejection Rate** (how often the SpanValidator rejects the output) between Shadow and Live. A lower rejection rate in Shadow indicates higher structural stability.
3. **Gradual Rollout:** If Shadow Mode metrics are superior, switch traffic to the new model for 5% of users, then 20%, monitoring for user edit rates (if users can manually correct spans) as a proxy for dissatisfaction.

---

6. Strategic Roadmap

This roadmap prioritizes actions that yield the highest reliability gain for the lowest engineering cost, moving towards the robust Design C architecture.

6.1 Horizon 1: Stabilization (Weeks 1-2)

- **Objective:** Kill the "Index Drift" failure mode immediately.
- **Action 1:** Switch extraction target from **Index Offsets** to **Substrings (Quotes)** in the prompt instructions.
- **Action 2:** Deploy the SpanAligner service with **Fuzzy Matching** (rapidfuzz) in the backend to map quotes back to indices.
- **Action 3:** Implement **Dynamic Template Rendering** (Handlebars/Mustache). Inject maxSpans and taxonomy definitions directly into the prompt template to eliminate Prompt-Policy Drift.

6.2 Horizon 2: Robustness & Disambiguation (Weeks 3-6)

- **Objective:** Solve the Visual-Semantic Gap.
- **Action 1:** Implement **Design C (Schema-First)**. Use zod-to-json-schema to enforce strict output formats and type safety.
- **Action 2:** Add **Disambiguation Rules** and **Negative Constraints** to the system prompt (e.g., "Shot Type is not Camera Angle").
- **Action 3:** Build the **Few-Shot Vector Store**. Index ~50 annotated examples and implement the retrieval logic to inject context-aware examples at runtime.

6.3 Horizon 3: Advanced & Agentic (Weeks 6+)

- **Objective:** High-fidelity control and cost optimization.
- **Action 1: Fine-Tuning (Distillation).** Once sufficient Golden Data and corrected user logs are available, fine-tune a smaller, faster model (e.g., GPT-4o-mini or a specialized Llama 3) specifically for this extraction task. This reduces latency and cost while "locking in" the specific "Director's Lexicon" behavior.[24]
- **Action 2:** Implement the **Critic Loop** for high-uncertainty spans or policy violations (e.g., "One Clip" checks).
- **Action 3: Synthetic Data Generation.** Use a strong model (GPT-4o) to generate thousands of synthetic video prompts with perfect labels to further train the fine-tuned model, specifically targeting under-represented categories like "Audio" or "Technical Specs."

Conclusion

The instability currently plaguing the span labeling system is not a mysterious property of "AI magic" but a solvable engineering problem. It stems from treating a deterministic data extraction task as a probabilistic generation task without adequate constraints. The system attempts to make a language model perform arithmetic (offset counting) and unstructured classification simultaneously.

By adopting **Anchored Substring Extraction**, we decouple the semantic task from the mechanical task. By enforcing **Schema-First Decoding**, we constrain the probabilistic output to deterministic structures. And by aligning the prompt hierarchy with the **Universal Video Semantics**, we bridge the gap between natural language and the rigorous demands of video

generation. Implementing this architectural roadmap will transform the system from a fragile, passive labeler into a robust, intelligent director's assistant, capable of guiding users toward high-fidelity, hallucination-free video generation.

Works cited

1. Prompt Engineering for Video Prompts.pdf
2. [1505.04474] Visual Semantic Role Labeling - arXiv, accessed November 22, 2025, https://arxiv.org/abs/1505.04474
3. 3 Building custom entity recognition models – The CLEANUP guide, accessed November 22, 2025, https://data.nav.no/fortelling/cleanup-guide/chapters/custommodels.html
4. Large Language Models as Span Annotators - arXiv, accessed November 22, 2025, https://arxiv.org/html/2504.08697v1
5. What Is a Prompt Injection Attack? - IBM, accessed November 22, 2025, https://www.ibm.com/think/topics/prompt-injection
6. Zero-Shot, One-Shot, and Few-Shot Prompting, accessed November 22, 2025, https://learnprompting.org/docs/basics/few_shot
7. Chunking Strategies for LLM Applications: The Complete Guide - GlobalNodes AI, accessed November 22, 2025, https://globalnodes.tech/blog/chunking-strategy-for-llm-application/
8. The guide to structured outputs and function calling with LLMs - Agenta, accessed November 22, 2025, https://agenta.ai/blog/the-guide-to-structured-outputs-and-function-calling-with-llms
9. Function calling - OpenAI API, accessed November 22, 2025, https://platform.openai.com/docs/guides/function-calling
10. Introducing Structured Outputs in the API - OpenAI, accessed November 22, 2025, https://openai.com/index/introducing-structured-outputs-in-the-api/
11. Introducing LangExtract: A Gemini powered information extraction library, accessed November 22, 2025, https://developers.googleblog.com/en/introducing-langextract-a-gemini-powered-information-extraction-library/
12. rapidfuzz/RapidFuzz: Rapid fuzzy string matching in Python using various string metrics - GitHub, accessed November 22, 2025, https://github.com/rapidfuzz/RapidFuzz
13. Rapidfuzz Explained. Rapidfuzz is a powerful Python library... | by Kasper Junge - Medium, accessed November 22, 2025, https://medium.com/@kasperjuunge/rapidfuzz-explained-c26e93b6012d
14. Span-Oriented Information Extraction A Unifying Perspective on Information Extraction - arXiv, accessed November 22, 2025, https://arxiv.org/html/2403.15453v1
15. Span-Oriented Information Extraction: A Unified Framework - SIGKDD, accessed November 22, 2025, https://kdd.org/exploration_files/p137-span_survey_cameraready.pdf

16. [2503.17952] SLIDE: Sliding Localized Information for Document Extraction - arXiv, accessed November 22, 2025, https://arxiv.org/abs/2503.17952
17. Sliding Window Technique — reduce the complexity of your algorithm | by Data Overload, accessed November 22, 2025, https://medium.com/@data-overload/sliding-window-technique-reduce-the-complexity-of-your-algorithm-5badb2cf432f
18. Few-Shot Example Retrieval Strategies - Emergent Mind, accessed November 22, 2025, https://www.emergentmind.com/topics/few-shot-example-retrieval-strategies
19. Dynamic Demonstrations Selection for Few-Shot Legal Argument Mining - CEUR-WS.org, accessed November 22, 2025, https://ceur-ws.org/Vol-4089/paper2.pdf
20. zod-to-json-schema - NPM, accessed November 22, 2025, https://www.npmjs.com/package/zod-to-json-schema
21. Revealing and Addressing the Self-Correction Blind Spot in LLMs - arXiv, accessed November 22, 2025, https://arxiv.org/html/2507.02778v1
22. Divide, Cache, Conquer: Dichotomic Prompting for Efficient Multi-Label LLM-Based Classification Acknowledgement - arXiv, accessed November 22, 2025, https://arxiv.org/html/2511.03830v1
23. How to: LLM-Automated Labeling for Legal Docs - Datasaur, accessed November 22, 2025, https://datasaur.ai/blog-posts/how-to-llm-automated-labeling-for-legal-docs
24. Overcoming Data Scarcity in Named Entity Recognition: Synthetic Data Generation with Large Language Models - ACL Anthology, accessed November 22, 2025, https://aclanthology.org/2025.bionlp-1.28/