ChatGPT

Novelist Context Loss Audit

End-to-end Phase Review

Premise Expansion Phase

Input: The initial user-provided project file supplies a brief **story premise**, along with metadata like genre, target audience, desired total words, chapter count, etc. [1] [2]. At this stage, the outline JSON is created with the raw premise and basic info (word count, chapter count), and marked as `Init` [3].

Prompt design: The premise expansion prompt takes the **original premise text** and asks the LLM to expand it into a longer paragraph focusing on conflict, stakes, and trajectory [4]. It appends the story **genre** and optionally the **target audience** as guidance, if those were provided, defaulting to "General fiction" if genre is missing [5]. Notably, no character or plot details beyond the original premise are included – the prompt explicitly says *"Preserve the premise's core ideas... Genre: {genre}. ORIGINAL PREMISE: {premise}"* [4].

Output: The result is a single expanded premise paragraph (150–350 words) written to the outline's `premiseExpanded` field [6]. For example, the simple premise of *"a carpenter discovers a mysterious door in a seniors' building..."* was expanded into a richer paragraph covering the enchantment and dangers of revisiting the past [7] [8]. The original premise remains untouched in the outline.

Context continuity: At this early phase, **little to no drift** occurs. The expansion stays true to the core idea, by design. However, one subtle omission is that the expanded premise is not explicitly used in subsequent phase prompts (some later prompts still reference the original premise). This could limit the benefit of the added detail. Still, at this stage the story's context is intact and actually enriched, with no contradictions introduced.

Story Arc Definition Phase

Input: Now working from a `PremiseExpanded` outline, the generator knows the overall story concept. The **expanded premise** is available in the outline, but the Arc Definer service actually still uses the original `premise` text as input for the prompt [9]. No characters or subplot information exists yet. The outline state is `PremiseExpanded` and contains total chapters count, which will guide the arc structure (e.g. 20 chapters → 4 acts by a rule) [10].

Prompt design: The arc prompt asks for a **multi-act structure**. It decides the number of acts based on chapter count (e.g. 20 chapters yields 4 acts) and instructs the LLM: *"Create exactly X ACT paragraphs... No headings, no markdown."* It provides the **genre**, any audience note, and the **premise** as context [11]. **Notably, it does not supply any character names or details** – at this point the model only knows the premise description of a protagonist like "a carpenter" and might invent specifics to tell a coherent arc. For example, the model was free to name characters or add backstory to connect the acts because the prompt did not restrict character identity beyond what the premise implied.

Output: The LLM returns X paragraphs (one per act) describing the story's progression. These are inserted as the outline's `storyArc` array with each act's summary in a `"definition"` field [12]. For instance, Act 1 of the *"The Door"* story came back as a paragraph introducing the carpenter in the building and hinting at the mysterious resident – and the model **invented a name** for the protagonist: *"Ian Holloway, a skilled but unassuming carpenter..."* [13]. It also named the eerie old resident *"Mr. Porter"*, who was not explicitly named in the premise. This output advanced the outline to `ArcDefined`.

Context drift: This phase is the **first point where continuity can drift**, due to the model extrapolating details without guidance. In our case, the model chose the name **Ian Holloway** for the protagonist and **Mr. Porter** for the enigmatic resident [14]. Since no official names existed yet, these choices aren't errors per se – however, they set up potential conflict with the forthcoming character outline phase. **The architectural decision to define the arc before defining characters meant the model had to assume or create character identities**, which introduces a **mismatch** if the next phase defines different names. Indeed, as we'll see, the protagonist's name "Ian" was not locked in; it drifted when characters were outlined. This is a key continuity gap emerging directly from prompt design: the arc generator omitted any placeholder or reference for the main character's intended identity, allowing the LLM to pick its own. Apart from naming, the arc paragraphs also added hints of plot not in the premise (e.g. the old photograph, the idea that the building "had an ageless aura") – enriching details, but again possibly introducing elements the later steps might not anticipate. There's no immediate contradiction yet, but the **seeds of derailment** (uncoordinated details) are planted here.

Character Outlining Phase

Input: With the story arc defined (outline now `ArcDefined`), the process moves to detailing characters. The **arc definitions** exist but are not directly used as input to the character generator. Instead, the Character Outliner pulls from the project settings: it knows how many characters to create in each category (1 protagonist, e.g. 2 supporting, 3 minor, etc as specified in the project) [15]. The **premise** is again used as the core context for character creation [16]. Importantly, the character prompt does **not** incorporate the story arc content or any names generated in the previous step. It does have access to author style presets and audience if specified, which can flavor the output (e.g. it can instruct to emulate Stephen King's tone if the preset flag is on) [17] [18].

Prompt design: The prompt instructs: *"Create exactly N characters for the premise below: • 1 protagonist... • X supporting... • Y minor... Each object MUST contain: name, role, traits, arc."* It explicitly demands a JSON array of character objects with those fields [19]. It appends the **premise text** (original premise) at the end of the prompt as context for the characters, along with any audience line or style line (for example, *"Target audience: Adults."* or *"Emulate the tonal style of Stephen King."*) [20] [21]. **Crucially, it does not mention anything from the story arc or previous outputs** – the model is operating only with the premise description of the scenario and general guidance about roles. There is no awareness that the arc already named the protagonist or introduced "Mr. Porter." The character prompt is effectively working in parallel, not sequentially building on the arc's specifics.

Output: The LLM returns an array of character profiles. The first object is the Protagonist, followed by the specified number of Supporting and Minor characters. Each profile includes a **name**, a role designation (e.g. "Protagonist: " prefix), a list of a few key **traits**, and a short **arc** describing that character's personal journey or role in the story [19]. In our example, the character list came back with a protagonist named **"Ethan Caldwell"** – a completely different name than the arc's "Ian Holloway" – and several residents such as

Martha Jennings, Henry Whitaker, Alice Morgan, Tommy Lee, Gloria Fields, each with traits and a brief arc [22] [23]. Notably, one of the supporting characters, **Henry Whitaker**, was described as an old resident who claims he's "been here forever," which is clearly aligning with the mysterious elder role (the same role Mr. Porter filled in the arc) [24]. In other words, the character outline *did* capture the archetype from the premise (an ageless resident) but gave him a different name (**Henry** instead of **Mr. Porter**). The outline is now marked `CharactersOutlined` and contains this roster in a `characters` array.

Continuity drift: Here the **identity drift fully manifests**. Because the arc and character steps were siloed, we ended up with two separate names for what is clearly the same story role: the protagonist is **Ian** in the arc summary but **Ethan** in the character list, and the enigmatic old man is **Mr. Porter** in the arc but **Henry Whitaker** in the characters [14] [24]. There's now an internal inconsistency in the outline. The root cause is that the **pipeline architecture did not enforce consistency across phases** – the arc phase invented names, and the character phase invented names, and the two were never reconciled. No mechanism exists to retroactively align the arc's narrative with the newly defined character identities. A reader of the outline at this stage would be confused: the act summaries talk about Ian and Mr. Porter, but the character list doesn't have those names at all (it has Ethan and Henry fulfilling those roles). This is a major continuity break.

Beyond naming, another gap emerges: the character profiles include rich **traits and personal arcs** (e.g. Ethan's introspective nature and how the experience changes him, Martha's role as a guide, etc. [23] [25]). However, these details exist only in the character section of the outline. Subsequent phases (beats, draft) do not explicitly use these trait descriptors or individual arc notes. If not consciously integrated later, things like *"curious, introverted"* or *"traumatized by childhood"* might never surface in the actual story. In short, character depth is defined here but risks being lost downstream due to lack of propagation. This isn't an immediate "drift" (nothing contradictory has happened with traits yet), but it's a **context omission** that can reduce the fidelity of the final narrative.

Subplot Definition Phase

Input: At this point the outline has a story arc (with acts and high-level beats, albeit some naming issues) and a set of characters. Next, the SubPlot Definer runs, but only if the project specified a subplot depth > 0. Assuming we do want subplots (the project had `subPlotDepth`: 2 for two subplots threads), the input includes the **premise** (this time the code uses the expanded premise if available) [26], the **story arc act summaries**, and the desired number of subplot threads (2). It also has access to the characters list and could theoretically use them, but instead it focuses on threading plot points. Importantly, the implementation actually reloads the original project file to get `subPlotDepth` and uses the **premiseExpanded** text if it exists [26]. Character names are not explicitly passed in the prompt, though the act definitions might mention some (like Ian, Mr. Porter) which indirectly introduces those names to the model.

Prompt design: The prompt task here is to create **subplot storylines** that span the acts. It tells the LLM to **"Create N subplot threads (IDs S1…S{N}). For EACH act, give the next stage of every thread."** The format required is a JSON object with keys `"Act 1", "Act 2", …` and each value an array of N strings (each string starting with `S1:`, `S2:` etc) [27]. It explicitly says *"Start each sentence with its ID followed by a colon"* and demands all arrays be exactly length N [28]. The prompt also provides context: it includes the **genre and premise/stakes**, and crucially a listing of the **Main story arc** by act: essentially a concatenation of *"Act 1: [act definition]"*, *"Act 2: [definition]"*, etc. [29]. This *acts block* means the model is aware of the act

summaries (containing any names or events introduced in the arc). There's an interesting design choice: the code concatenates all act definitions and subplots into one long prompt section, rather than breaking it act by act. That gives the model a holistic view of the story's flow while it generates subplots.

No direct mention of character profiles is given, but since subplots often involve characters, the model must infer them. The prompt does not forbid new names, but usually the model will refer to existing entities from the act summaries. Indeed, *the instructions in the code comment say each line is prefixed S1, S2 so that threads can be traced in later stages* [30] – implying these subplot IDs will serve as labels to connect related events across acts (a clever continuity device).

Output: The LLM returns a JSON with each act's subplot lines. For example, it might produce something like: S1 could be *"The residents begin whispering about a door that shouldn't exist"* in Act 1, continuing as *"The whispers grow into rumors of past disappearances"* by Act 2, etc., and S2 could be another thread like *"An old photograph is found that raises questions"* progressing through acts. In our running example, that's exactly what happened: **Subplot S1** dealt with the *strange behaviors and whispers among residents about the impossible door*, and **Subplot S2** introduced *the discovery of a mysterious old photograph of the protagonist (Ian/Ethan) looking much older, hinting at the building's secret* [31] [32]. The output ensures each act has the same number of subplot entries (2 in this case) and that each entry starts with the proper ID. The outline is updated to `SubPlotsDefined`, and the `storyArc` now contains these subplot lines in each act's `sub_plots` array [33].

Context continuity: The subplot content generally reinforces the main plot and often involves the supporting characters indirectly (e.g. the photograph might be found by some resident – later we see it was Henry in the beats). This phase did not introduce obvious new contradictions; rather, it provided connective tissue between acts. One structural decision stands out as a **guardrail**: labeling each subplot thread with S1, S2, etc. This was intended to maintain consistency – the IDs act like placeholders linking the same subplot across acts. This is effective within the outline: it's clear, for instance, that *"S2: A mysterious old photograph of Ian is found…"* in Act 1 is the beginning of a thread that continues as *"S2: Mr. Porter shares cryptic clues…"* in Act 2 and so on [32] [34].

However, these subplot IDs and lines are *internal outline logic*. The next stages need to propagate them for the benefit to carry over. We will see that while the outline preserves them through Structure, the actual drafting prompt does not leverage these IDs or explicitly mention the subplot threads to the language model, meaning this continuity device wasn't fully utilized where it counts (in writing the prose). Still, at the outline level, context is solid: the subplots align well with the main conflict and no new name drift occurred here. (Subplot lines used "Ian" and other names exactly as per the act definitions – so they carried forward the existing naming inconsistencies but didn't create new ones.)

Beats Expansion Phase

Input: Now the outline has a fleshed-out arc with subplots. It's time to break each act into fine-grained **story beats**. The Beats Expander takes each act's `definition` and its subplot lines, and will produce a list of beats (plot points or scenes). **Important inputs include**: the **premise** (original premise again, not even the expanded one in the current code) [35], the **genre and audience**, and crucially the list of **character names** from the outline. The code explicitly compiles all character names into a comma-separated list and adds a line in the prompt: *"Principal characters: [Name1, Name2, …]."* [36]. This is done to help the model maintain continuity with known characters. In our case, it would send *"Principal characters: Ethan Caldwell,*

Martha Jennings, Henry Whitaker, Alice Morgan, Tommy Lee, Gloria Fields." to remind the model of who is in the story. Note, however, that the story arc text the model sees (from act definitions) contains "Ian" and "Mr. Porter," which are *not* in that list. So already the model has two sets of names for possibly the same entities. The prompt will also include any **subplots for that act**, labeled as such.

Prompt design: For each act one by one, the service prompts: *"You are a pacing assistant. Create exactly M concise, chronological story beats (<25 words each) for Act X. Return ONLY a JSON array of strings."* It provides the number `M` which is calculated as `(total chapters / total acts) * 3` – essentially, it assumes each chapter will have 3 beats, so M is 3 times the number of chapters in that act [37]. In our 20-chapter, 4-act example, each act gets 15 beats (because 5 chapters per act * 3). The prompt then includes context lines: the **audience and principal characters line**, and if the act had any subplot threads, a line like *"Subplots in this act: [S1 details; S2 details]."* [38]. It also appends the **genre**, the **premise**, and the **act's definition** text [39]. So the model sees a wealth of information: high-level story synopsis, key characters, and specific subplot events that need to be threaded into the beats.

It's worth noting the model is not asked to explicitly tie each beat to a particular subplot or character, but the presence of those names and subplot summaries in the prompt strongly cues it to include them. The **characters line is a double-edged sword** here: it ensures the model knows "Ethan Caldwell" is the protagonist's name per the official list, but the act definition it just read might have primarily mentioned "Ian Holloway." The model has to reconcile that – which it attempted to do, as we'll see, by including both names in different beats as if they were separate people.

Output: The service expects an array of exactly M beat strings for each act. The model, following instructions, provided short, punchy beat sentences. For example, Act 1's beats (15 items) included entries like: **"Ian Holloway arrives at the senior apartment building for a routine renovation job."**, **"Ian meets Mr. Porter, a resident who cryptically claims, 'I've been here forever.'"**, **"Henry Whitaker finds an old photograph of Ian looking much older in the common room."**, **"Martha Jennings shares strange observations about other tenants."**, **"Ian enters the door, leading to moments from his past, enchanting him initially."**, and eventually **"The door vanishes, leaving Ian changed and the cycle ready to reset."** [40] [41]. These beats clearly weave in the subplot threads (whispers about the door, the mysterious photograph) and mention many characters by name – notably **both** Ian and Ethan show up in the beat lists. Indeed, by Act 2's beats, one entry reads: *"Ethan warns Ian about spending too much time in the past."* and another says *"Ethan notices Ian's behavior change after losing access to the door."* [42]. This is a striking sign of the model's confusion: it treated Ethan and Ian as two distinct characters interacting. In reality, those were supposed to be the same person, but the model doesn't know that. It saw Ethan's name in the character list (likely interpreting him as some other character, since Ian was the protagonist in the arc narrative it saw) and invented a scenario where Ethan is perhaps a friend or a new arrival cautioning Ian. Similarly, Mr. Porter and Henry Whitaker both appear as if they're separate—e.g. one beat has Mr. Porter giving cryptic clues and another beat in the same act has Henry sharing tales, whereas originally those were one role.

All acts get their beats filled in this manner. The outline's `storyArc` now has each act's `beats` array populated with these strings, and the state becomes `BeatsExpanded` [43].

Continuity drift: This phase is where the earlier inconsistencies fully bloom into actual plot incoherence. The model, trying to honor all inputs, effectively split one character into two. The protagonist's two names are both used, leading to beats that imply the presence of two characters (Ian and

Ethan) in the story when in fact there should be one. Likewise, the mysterious old man figure appears as both Mr. Porter and Henry in parallel. This is a direct result of **cross-phase context mismatch** – the arc provided one set of names, the characters list provided another, and the beats generation prompt gave the model *both sets* without clarifying they were the same entities. The guardrail of listing principal characters did ensure all key names were included in the beats, but without additional instruction, the LLM treated them as distinct. In essence, the pipeline's **lack of a reconciliation step or unified reference** caused a divergence in the story line at the beat level: we now have a story that unintentionally has an "Ethan vs Ian" subplot that was never intended. This is a severe continuity error introduced at the outline level.

Aside from naming issues, the beats phase otherwise did a decent job integrating plot threads – e.g. every beat of Act 1 and Act 2 is logical and covers the intended events (door discovered, nostalgia, then dark turn, etc.) in order. It even foreshadows well (the photograph, the hints about previous carpenters). So the **story continuity** in terms of events is strong, but the **character continuity** is deeply flawed here. Another subtle point: the Beats Expander assumed 15 beats per act uniformly. This fixed number meant the model sometimes had to break or merge events to fit the count. It generally complied, but this rigidity can cause some pacing oddities (not all acts of a novel naturally have the same number of plot beats). For instance, Act 1 and Act 2 beats might feel a bit dense to ensure exactly 15 each. This is a structural decision that can indirectly affect narrative flow, although the model handled it by enumerating smaller beats.

Structure Outlining Phase

Input: With detailed beats for each act, the outline now has all the pieces needed to map out chapters. The Structure Outliner's job is to assign beats (and subplot threads) into a chapter-by-chapter breakdown. **Key inputs** are: the total number of chapters (from the project), the acts with their beats and subplots, and the premise/genre again. The service will distribute the beats roughly evenly across chapters. The code logic enforces exactly **3 beats per chapter** in the final output [44]. It knows how many chapters are supposed to be in each act (roughly chapters/acts, e.g. 5 chapters per act if 20/4) and will ask for that many chapter entries per act. It also knows the subplot depth (say 2) to require up to 2 subplot lines per chapter. The prompt will ensure at least 1 subplot appears in every chapter's plan.

Prompt design: This prompt is framed as: *"You are a seasoned development editor. The novel has N chapters across M acts. For each chapter, create: "number": int, "summary": 1–2 sentences, "beats": exactly 3 strings drawn from the act's beat list, "sub_plots": 1–D strings each beginning with its ID."* [45]. It provides context after that: the **genre, premise** (expanded if available) are given, and then a comprehensive **"ACT / BEAT / SUB_PLOT GRID"** – essentially a dump of each act's info: Act 1 definition, plus "SUB_PLOTS:" lines for that act, plus "BEATS:" listing all beat strings for that act [46]. This is a very detailed prompt; the model sees the entire structure of the story (act by act) including every beat we just generated, and the subplot threads labeled S1, S2, etc. With this, it's asked to package those beats into chapter groupings.

A couple of things to note: the prompt explicitly says "exactly 3 beats" per chapter (so the model must pull three specific beats from the list for each chapter) and to include 1–2 subplot lines (with labels) per chapter [45]. This implicitly means each beat in the final chapter outline *should be one of the beats from the act's list*. We rely on the model to correctly allocate them. There is also a validation in code that ensures the returned JSON has the correct number of chapters and each has exactly 3 beats and valid subplot lines [47] [48].

Output: The LLM produces a JSON array of chapter objects. Each chapter has a short summary, a list of 3 beats, and a list of subplot lines. The beats in each chapter correspond to some of the act beats (the

phrasing is usually identical or slightly trimmed, indicating the model took them from the provided list). For example, *Chapter 1* came out as: summary: *"Ian Holloway arrives at the apartment building… meeting the cryptic Mr. Porter who claims eternal residence."*, with beats: ["Ian Holloway arrives at the senior apartment building for a routine job.", "Ian meets Mr. Porter, who says 'I've been here forever.'", "Residents whisper about a mysterious door that shouldn't exist."] [49] [50], and sub_plots: ["S1: The residents… exhibit strange behaviors, whispering… about a door that shouldn't exist."] [51] [52]. We can see the chapter summary is basically a concise combination of those beats. Each subsequent chapter continues this pattern. By *Chapter 5*, the end of Act 1, the summary encapsulates the turning point: *"Ian enters the door… enchanting him initially, but soon dark memories surface."*, with beats including *"Ian enters the door…", "Dark memories surface…", "The door vanishes, leaving Ian changed…"* [53] [54]. This corresponds exactly to the last beats of Act 1.

When the outline transitions to Act 2 (Chapter 6 onwards), the naming confusion becomes apparent in the output. In the chapter breakdown, the first several chapters (1–9) stick with **Ian** as the active character in summaries and beats (since they're covering Act 1 and early Act 2 beats). However, later in Act 2's chapters, we see lines like: *"Ethan warns Ian about spending too much time in the past"* as a beat (Chapter 9) [55], which is directly from the Act 2 beats. Chapter 9's summary accordingly mentions Ethan and Martha: *"Ethan warns Ian… and Martha hints at knowing more about the door."* [56]. By *Chapter 10*, the summary says: *"The door suddenly vanishes… and Ethan notices Ian's behavior change…"* [57]. Then, starting in *Chapter 11*, it appears the model decided the viewpoint had shifted – Chapter 11's summary is *"Ethan steps through the door… confronting his first failure…"* [58]. From that point on, **Ethan becomes the protagonist name used in summaries and beats**, and Ian is referenced in third person (or not at all except in subplot notes). Essentially, the chapter outline **switched the main character** mid-story: chapters 1–10 treat Ian as the protagonist (with Ethan appearing as a side character who somehow was present), and chapters 11–20 treat Ethan as the protagonist (with Ian largely gone or referenced as someone who was replaced). This is a direct artifact of the earlier confusion – the model is trying to reconcile two "protagonists" by handing off the narrative from one to the other. In fact, Chapter 16 explicitly says *"Ethan Caldwell arrives, replacing Ian as the new carpenter, while Martha notices the familiar yet unfamiliar presence of Ethan."* [59]. The outline has inadvertently created a **duplication/transition** where none was intended: originally Ethan *was* Ian. Now the story outline implies Ian was one person and Ethan is a new individual who takes over after Act 2. The mysterious cycle in the premise (where the protagonist is replaced by another man with his name) got totally misinterpreted: it should have been that *Ian/Ethan* is replaced by an older version of himself, but instead it reads like *Ian is replaced by Ethan Caldwell (a new character)*, and then later Ethan sees echoes of Ian. This is a dramatic continuity failure – effectively a derailment of the original story logic due to name mix-ups.

On a smaller scale, the old-resident confusion persists too. For example, Mr. Porter appears in early chapters, and Henry Whitaker in later ones as if they are different people: Chapter 1 mentions Mr. Porter [49], while by Chapter 3 and 4 we see beats with Henry Whitaker and Gloria etc., and Chapter 8's summary is about Mr. Porter again giving clues [60], then Chapter 12's beats have Henry sharing an experience [61]. A reader of this outline would likely be perplexed why there are two elderly sages in the building both fulfilling similar roles.

Continuity drift: The Structure phase itself did not introduce brand-new inconsistencies; rather it **amplified and locked in the inconsistencies that came from the beats**. By distributing the beats into chapters, it canonized the existence of both "Ian and Ethan" in the outline's chapter-by-chapter plan. The damage from the beats phase is now formalized: each chapter summary and beat list is internally consistent (the model

did a decent job selecting coherent groupings of beats), but across chapters the protagonist's identity is inconsistent. The *outline as a whole is derailed* with respect to character continuity. The **guardrails present** (requiring exactly 3 beats, at least 1 subplot per chapter) ensured a uniform structure and that subplots weren't forgotten entirely, which is good – e.g. every chapter includes at least one `S1` or `S2` line so that side threads like the "whispers" or "photograph" are continuously addressed. In fact, the chapter outline shows subplot S1 and S2 threads being weaved through most chapters (often alternating) which is a positive outcome of that rule. But what's missing is any guardrail for character or plot consistency beyond those structural elements.

Another point of note: because the model had to fill every chapter with a summary and beats, some beats in the act lists might have been left unused or combined. It appears the model managed to use all beats by evenly dividing them (15 beats per act / 5 chapters = 3 beats each, exactly). If there had been any remainder or if a beat didn't cleanly fit, it might have been dropped. Without a check to ensure *every* beat from the act list got allocated, there's a risk a plot point could be lost here (though our example doesn't show glaring omissions – the model was compliant in using exactly 3 per chapter and likely took them in order).

At this stage, the outline is essentially final (`StructureOutlined`). It contains a detailed plan. But as we see, it suffers major continuity issues (protagonist identity swap, duplicate elder characters). A human writer following this outline would likely be confused or have to do significant mental editing to reconcile who is who. The next phase will use this outline to actually draft prose, and we'll examine how those issues might propagate or even worsen if not corrected.

Draft Generation Phase

Input: The final outline (with chapters, beats, subplots, etc.) is the blueprint for drafting the actual novel. In the drafting phase, each **chapter is written out in full prose**, typically broken into a few pieces to manage length. The drafting code takes as input the outline file and the specified output directory. It also loads the chosen **author style preset** (e.g., Stephen King's stylistic preferences) so it can prompt the model to write in that voice [62]. Before writing, the system sets up a **running summary** (initially "None yet" for chapter 1) and a cost tracker for API usage [63].

For each chapter from 1 to N, it will generate the text piece by piece. The primary inputs to the model at each step are: - The **chapter context** (chapter number, piece number out of total, etc.), - The **author style persona** (the prompt literally says "You are {AuthorName}." e.g. Stephen King), - The **target length** guidance (rough word target and piece count), - A count of **remaining beats and remaining subplots** for the chapter, - The **running summary** of previous chapters, and - The **previous piece text** (for pieces after the first in a chapter) as continuity context [64] [65].

It's critical to note what is *not* included: the prompt does **not explicitly list the chapter's planned beats or subplot details by content** – it only provides how many are left to cover (as numbers) [65]. For example, at the start of Chapter 1, the model sees something like: *"Write only piece 1/P of Chapter 1. Start with:* `--- piece 1/P ---`*. Target ~350 words. Remaining beats: 3. Remaining subplots: 1. RUNNING SUMMARY: None yet. PREVIOUS TEXT: (none)"* [65]. It is not explicitly told what those 3 beats are (aside from whatever it might recall from the outline if it's in the prompt memory, but here they did **not pass the outline text itself**). The assumption is that the model, having written previous pieces or having the running summary (for later chapters), will implicitly know the story direction. For Chapter 1 piece 1, the model only has the premise

knowledge it internalized and perhaps the author style – essentially it's starting the story fresh, guided loosely by "remaining beats: 3" which just indicates this chapter should hit three plot points.

Prompt design: The piece-writing prompt has a YAML-like front matter (not actual YAML, but similar to Prototype 2 style) with the chapter and piece info, then says: *"Continue the story from the previous text. Write ONLY the next piece."* It stresses the format: *"Each piece must start with the specified tag... Only write one piece at a time when prompted."* (These instructions were used in prototypes; in the current code it's a bit simplified but effectively similar). The code adds special retry instructions if needed – e.g. if the tag was wrong, or if the model seems to skip ahead to later pieces, it will insert lines like *"The tag was wrong. Use the exact tag shown."* or *"Do not skip ahead. Write piece X only."* on retries [66] . This helps enforce the technical structure. But notably **there is no reminder of specific content to include** beyond structural correctness. There is no checklist of beats here (unlike in the outline prompts). The model is mostly on its own to generate story text that logically follows from prior text.

As the model writes each piece, the system checks and retries if: - The piece doesn't start with the correct `--- piece X/Y ---` tag (formatting issue), - The piece is too short or too long (outside a 250–500 word range roughly) [67] , - The piece doesn't end in a proper sentence (no trailing open sentence) [68] , - The piece is too similar to prior text (duplicate content) [69] .

These are **technical guardrails** to ensure a smooth output format and avoid obvious repetition/cliffhangers. They do not directly ensure story events are included or correct; they only ensure the text is well-formed as a chapter part. The system also logs each prompt and completion to a cost log (including token counts and cost) [70] [71] , carrying over the prototype's cost-tracking feature (we see lines logged per model call) [72] .

Output: The final output is a markdown file per chapter (e.g., `chapter_01.md`) containing the pieces merged, with the piece markers in place as scene breaks. If we examine a sample generated chapter (from a different test story in the repository, since the "Door" story draft wasn't provided, we have one about a character named Claire), it's formatted like:

```
--- piece 1/9 ---
[Prose of scene 1...]
--- piece 1/9 ---

--- piece 2/9 ---
[Prose of scene 2...]
--- piece 2/9 ---

... and so on ...
```

From the example chapter text, we can see the model writes rich, descriptive prose that *feels* coherent on a local scale (scene by scene). However, since the model was not explicitly fed the beats or outline details, it generates events and dialogue somewhat freely, based only on what it remembers and the general direction of the story so far.

Context drift: This final phase is where any **missing context propagation really translates into missing or altered story content**. Because the drafting prompt omitted the explicit **beats list**, the model might not include some crucial plot points from the outline, or might introduce new ones to fill the narrative logically. In our running example, if Chapter 1's beats were (a) arriving at the job, (b) meeting Mr. Porter, (c) hearing whispers about a door – the model would only include those if it infers them from the premise and perhaps the expanded premise (which it might have internalized during outline generation, but it's not guaranteed to recall specifics without being told). Given the outline, Chapter 1's summary did mention those points, but the draft prompt didn't reiterate them. The model might cover (a) and (b) because that's a natural opening scenario (arrive at job, meet an odd resident), but it could easily omit (c) or push it later, especially if "whispers about a door" doesn't come naturally to mind in the first scene. In fact, if we inspect a draft output, it may have subtle or major deviations. The provided example chapter (with Claire, Mark, Alex from another storyline) suggests that in at least that test, the draft text can introduce characters and backstory that were not part of a simple outline – it's writing a story that makes sense to it, which might not perfectly align with the outline structure.

For the "Door" story, a likely outcome is that the draft might continue referring to the protagonist by one name consistently (depending on what name was used at end of outline or in the running summary). If the running summary at chapter 11 onward refers to "Ethan", then chapter 12's prompt context would talk about Ethan's recent experiences, etc. The model might then drop "Ian" entirely and treat Ethan as the sole protagonist henceforth, effectively rewriting the story to have two different main characters in the first half vs second half. This kind of drift would **seriously derail the story's logic** – readers would wonder where the original protagonist went or who this new Ethan is, unless the text somehow explains it (which in the outline it doesn't, because it wasn't the intended plot). Similarly, the model could confuse Mr. Porter and Henry – perhaps merging them or randomly using one. Without the outline explicitly in front of it, the model may default to using the principal character names that were listed (since those were fed in beats stage, but not in draft stage). If it leans on the running summary, whichever name the summary uses for the old man (likely it might mention Mr. Porter from early chapters), the model might stick with that and effectively ignore that "Henry" ever existed as a separate person in the outline plan. Or vice versa. In any case, there's a discontinuity between what the outline intended and what the draft will likely emphasize.

Additionally, **character traits and arcs defined earlier are at high risk of being lost** in the draft. The model isn't explicitly reminded that Ethan is "nostalgic and introverted" or that Martha is "wise and mysterious" from the character sheet. It would only pick those up if it gleaned them from how the outline described events (e.g., maybe the beats or arc imply some traits). For instance, the outline said Ethan was curious and nostalgic; the model might show him being curious simply because the plot has him investigating the door. But more nuanced traits (like "has childhood trauma" or "moral ambiguity") may never appear because the model wasn't cued to incorporate them. The same goes for **themes**: The project's genre and any suggested themes (Stephen King's preset includes themes of guilt, memory, etc.) are not strongly reinforced beyond the initial premise. The draft model might include some thematic elements organically, but there's no guarantee it will hit the intended themes like *"can one truly change the past or just become its prisoner,"* which was eloquently stated in the expanded premise [8] but could get lost in translation to scenes.

Missing beats and subplots: Because the model wasn't told "you must include X event," some beats could be skipped. The prototype solutions had a step to check if a required beat or subplot was not covered in a piece and then prompt a retry to cover it [73] . The current pipeline doesn't do that. If, say, the subplot about the old photograph (S2) is supposed to appear in Chapter 3 according to the outline, but the model's

narrative flow in drafting Chapter 3 doesn't mention any photograph, nothing in the pipeline catches that omission. The story will simply proceed without that element, meaning a plot thread set up in the outline quietly disappears. Similarly, a late subplot like residents vanishing (S1 in Act 4) might not be fully realized if the model doesn't focus on it. The only nudge the model has is the running summary which might mention prior subplot events, but if it doesn't deem them important, it might not resolve them. Essentially, **the draft phase has no direct awareness of the outline's checklist of beats or threads** – it relies on the model's memory and initiative, which is a fragile approach.

In summary, by the end of the draft generation, we expect the **story context to have drifted in several ways**: - **Character identity and relationships** could be inconsistent (Ethan vs Ian confusion might lead to actual narrative inconsistency unless the model "chooses" one and implicitly writes out the other). - **Some planned plot points might be omitted or altered**, especially those that were only in the outline but not naturally reinforced (the draft might introduce new scenes or omit outline scenes). - **Themes/traits** that were not continuously reinforced likely won't be emphasized, potentially flattening the story's depth compared to the outline's promise. - On the positive side, the draft will be divided into well-structured pieces with proper scene breaks, and will maintain general coherence with preceding text due to the running summary, preventing total non sequiturs from chapter to chapter. Technical continuity (tense, perspective, style) should be fairly consistent since the prompt keeps the same "You are [Author]" persona throughout and uses the running summary to maintain memory.

Where continuity truly **broke down** is in the *logical consistency with the outline*. The guardrails ensured format and prevented outright model errors (like incomplete sentences or wildly repeated text), but **did not ensure adherence to the outline's specifics**. As a result, the story that emerges can deviate in important ways from the planned structure, as seen with the character confusion and any dropped beats.

Major Cross-Phase Issues

Character Identity Drift Across Phases

One of the most glaring issues was the **inconsistent handling of character identities between phases**, leading to the Ethan/Ian situation. This stemmed from an architectural choice: defining the high-level plot (arc) *before* defining the characters, combined with prompt designs that lacked references to previous outputs. The Arc Definer phase invented placeholder names (e.g., *"Ian Holloway"* for the unnamed carpenter) [14], not knowing the Character Outliner would later provide an official name (*"Ethan Caldwell"*) [22]. Because the arc prompt didn't reserve or use a consistent identifier for the protagonist (like calling him just "the Carpenter"), the LLM filled in a name on its own. When the Characters phase ran, it had no knowledge of "Ian Holloway" – it generated a protagonist name independently. This **misalignment was never corrected**. There was no step to reconcile or unify the arc's narrative with the character list. In an ideal flow, either characters (names) should be established first and fed into arc generation, or the arc should avoid specific names until characters are known. The lack of such synchronization is a flaw in the outline **schema/progress design**: it assumes each phase builds on the last, but here Phase 2 (Arc) and Phase 3 (Characters) produced conflicting facts.

The ripple effect of this drift was felt in later phases: Beats expansion included both names in the prompt, causing the model to think Ethan and Ian were two people interacting [42]. Structure then locked in chapters where the protagonist inexplicably changes name partway [59]. By the drafting stage, the model either has to silently choose one identity or continue the error, confusing the narrative. For the reader or

writer, this is a catastrophic lapse – the protagonist's identity is the backbone of the story. **Impact:** Very high. It renders the outline internally inconsistent and would force a writer to manually rewrite either the outline or the draft to fix it.

Similarly, the case of **Mr. Porter vs Henry Whitaker** (two names for the old eternal resident) is another identity drift. It's slightly less central than the protagonist, but still very problematic. It arose for the same reasons – the arc named a character in lieu of an official definition, and the character outline named presumably the same figure differently. The model then kept both. This indicates a structural issue: the outline schema had a place for characters, but the arc text had already "hardcoded" a character that wasn't linked to that list. **Root cause:** a combination of out-of-order phase sequencing and prompt omissions (not informing the arc generator about character names, and not informing the character generator about arc content).

In both cases, a guardrail could have been: after character generation, perform a pass to replace any placeholder names in the arc/beats with the official names (a simple find-and-replace of "Ian" -> "Ethan" in outline text, for instance). There was even a utility in code (`SlugHelper` perhaps intended for consistent naming), but it appears not applied in this scenario. The **guardrail breakdown** here is that the system relied on the LLM to be consistent rather than enforcing consistency through the data pipeline. The expectation that the LLM would either not introduce names or magically use the same invented name twice was misplaced. Without explicit instructions, the LLM had no reason to stick to one name across phases. Essentially, **the continuity of character identity was left to chance**.

Lack of Cross-Phase Context Propagation

Another broad issue is that **each phase's prompt often omitted crucial context produced by previous phases**, leading to information loss or underutilization. Some examples: - The **Story Arc prompt** didn't utilize the richer *expanded premise*. It fell back to the original premise text [9], meaning any nuance or stakes added in the expanded premise were not leveraged to shape the arc. This is a missed opportunity – e.g., the expanded premise had specific language about *"the stakes… the question can one change the past or become its prisoner"*, but the arc might not emphasize that theme because it never saw it. *Guardrail issue:* The system should carry forward the best version of the premise to all subsequent phases; not doing so dilutes thematic context. - The **Character Outliner prompt** had no knowledge of what happened in the arc. It created characters in a vacuum of plot context (just using the premise) [16]. This can lead to characters that don't quite fit the evolving story. In our case, it actually aligned decently on some aspects (likely because the premise itself contained enough clues). But imagine if the arc had introduced, say, a major plot device or twist – the characters might not reflect any readiness for that because it wasn't in the premise. For instance, if the arc established an antagonist in Act 2, the character list might fail to include a proper villain if the premise didn't mention one. *Guardrail issue:* The character generation should possibly consider the main story arc or at least be fed the act summaries to ground characters in actual story events. - The **Beats Expansion prompt** did try to include cross-phase context: it injected character names and subplot lines into the prompt [74] [75]. This is one phase that recognizes previous outputs (characters and subplots). However, it did not explicitly clarify relationships (as noted, it didn't say "Ethan = protagonist = previously called Ian"), so while it included context, it didn't resolve conflicting context. It also still relied on the original premise instead of expanded premise text for some reason [35], another minor inconsistency. - The **Structure Outliner prompt** used the outputs of all prior phases in a big combined form (all beats and subplots) [46]. This is good in theory, but the sheer volume of information could be overwhelming for the model. We essentially dumped the entire outline-in-progress back into the model. While this ensures no beat or

subplot is forgotten during chapter planning, it also **perpetuates any errors** (like both names). The model wasn't guided to fix them, just to use them. - Finally, the **Draft generation prompt** is where context propagation largely **breaks down**. After spending so much effort to create a detailed outline, the actual drafting step doesn't explicitly feed most of that detail into the model. It gives only high-level numbers and previous text. The carefully constructed beats and subplot lines from the outline are **not directly mentioned**. For example, the outline might say Chapter 7's beats are A, B, C, but the draft prompt for Chapter 7 will not list A, B, C – it only says "Remaining beats: 3" [65]. The model has to rely on the prior chapter's content and its general understanding of story to guess what A, B, C might be. This is arguably the biggest context loss in the pipeline: the structured outline is not explicitly used to guide the narrative generation. It's like having a detailed roadmap and then asking someone to drive to the destination from memory without looking at the map.

The result of this omission is that **important story elements can vanish**. If the model doesn't naturally include something and there's nothing in the prompt to remind it, that piece of context is lost in the draft. For instance, maybe the outline specified a character's internal conflict to resolve in a particular chapter (one of the beats), but if the model doesn't recall that, the scene might skip the resolution, leaving a dangling thread. The pipeline provides no automatic feedback loop to catch such omissions. In prototypes, as mentioned, they had at least a check for missing beats or explicit markers in the prompt that forced the model's attention to them [76] [77]. The current system dropped those markers, so the model's only compass is the narrative so far. This is a design regression in terms of ensuring outline fidelity.

Themes and traits are also victims of this. They were largely contained in the initial premise and character bios and never reinforced. By the time we get to drafting, nothing in the prompt says "remember, this is a story about memory and guilt" or "this character is haunted by childhood trauma" – unless those were clearly woven into the outline's beats, they'll be gone. For example, Stephen King's preset listed *"themes of addiction, guilt, redemption"* [78] as typical, but unless the model subconsciously follows the King style, the final story might not explicitly touch those themes that were intended, because the system never mentions them again after preset loading. Even the style presets: the prompt only says "You are Stephen King" but doesn't reiterate, say, *preferred tone: ominous* or *hallmarks: deep introspection*. It trusts the model to emulate King's voice just from name. That might yield general stylistic similarity, but finer points could be missed (e.g., the model might not automatically include *"use of the supernatural as metaphor"* or specific motifs like *"storms/doors as motifs"* unless it does so by its own training bias for King).

In short, **the pipeline fails to propagate a lot of the contextual data from earlier planning into the actual writing**. Each phase tends to "reset" the context to just the basics (premise, maybe genre) and whatever the phase produced just before, rather than accumulating context. The result is cumulative loss: the further along we go, the more of the original outline detail falls away. By the final draft, only the most salient plot points survive (and some not even, as we've seen with missing beats). The guardrails to prevent this would have been explicit prompt injections of outline details (like listing beats to cover) or post-generation validation. Those guardrails were **present in prototypes but broke down in the current design** – they were not carried into the new system, leaving a gap between outline and draft.

Structural Rigidities and Derailment Points

Some architectural or schema decisions, while well-intentioned, led to potential story derailment or at least awkwardness: - The insistence on **exactly 3 beats per chapter** (and by extension 3 scenes per chapter). This uniform pacing doesn't fit every story naturally. It could cause the model to compress or stretch

content unnaturally. For example, if Act 1 had 13 beats instead of 15, the structure module would still expect 5 chapters * 3 beats = 15 beats, so the model might fabricate 2 filler beats to reach the count. Conversely, if there were more beats, it might have to merge some into one or drop a couple (though the code tries to calculate beatsPerAct to match exactly). This rigidity can derail story flow subtly by prioritizing structure over story logic. - The **Act and chapter mapping** in the code is a bit opaque (there's a function to determine which act a chapter belongs to by counting chapters) [79] . If that mapping were to go wrong (say an off-by-one error), the model might get a wrong sense of context, like thinking Chapter X is in Act 3 when it's actually Act 2, and thus mixing up plot progress. While I didn't see a clear bug there, it's a fragile area – any schema mismatch (if chapters per act aren't equal, for instance) could confuse the context fed to the model (e.g., it might feed Act 3 beats to a chapter that's actually Act 2). - **No final validation stage**: After drafting, there is no mechanism to compare the draft against the outline to see if all outline beats were covered or if any new major elements appeared. If such a validation existed, it could flag, for instance, "The outline said Ethan's old identity should show up, but the draft never mentioned it" or "The draft has a character named Claire that is nowhere in the outline" (just hypothetically). Without it, the output could be far off-track and the system wouldn't know. Essentially, any derailment that happened is left for the human to discover. From a writer's perspective, this reduces trust in the tool – one might have to manually check the draft against the outline line by line, defeating the purpose of automation. - **Subplot resolution**: The outline ensures each subplot appears throughout, but it doesn't ensure they conclude satisfyingly. For example, in the outline, S2 (the photograph mystery) appears in acts 1, 2, maybe Act 3 in a journal discovery, etc. The final chapters still list S2 events up to chapter 19 [80] . But does the draft actually resolve what the photo meant or what Mr. Porter's secret was? Possibly not, if the model wasn't guided to do so. There was no explicit instruction like "make sure to resolve subplot threads by the final chapters." It's implied by beats perhaps, but not guaranteed in prose. This is a narrative guardrail missing – checking that subplots reach a conclusion (Prototype 1 had a notion of checking scene endings and story logic, albeit minimal). - **Perspective and voice consistency**: The outline doesn't specify POV, but the style preset does ("third-person limited" for King, presumably applied). If the model were to drift into a different perspective mid-way (say, accidentally start a first-person narration or head-hop between characters), the current system wouldn't catch it. A possible guardrail could be instructions or checks on POV consistency. They mentioned POV consistency as a gap in prototypes [81] and indeed it's not addressed here either. If a writer finds the draft suddenly not following the intended POV or tense, that's another editing headache.

In summary, multiple cross-phase issues boil down to one core problem: **the phases are too disjointed in terms of data flow**. Each phase does its job in isolation, and the connective tissue (ensuring continuity of names, facts, and emphasis) is weak or absent. Where there were attempts (the principal characters list in Beats, the unified grid in Structure), they ended up surfacing conflicts rather than resolving them, because no logic was built to reconcile differences. The schema enumerated phases (`Init -> PremiseExpanded -> ArcDefined -> ... -> StructureOutlined -> Finalized`) but did not enforce coherence of content across those milestones. Thus, architectural decisions like ordering and the choice to not loop back for corrections significantly influenced these failures.

Comparison to Prototypes

The current Novelist project (C# implementation) drew inspiration from two prior Python prototypes, but not all of their lessons were integrated. Below we compare key areas:

- **Prompting with Beat Scaffolds:** Prototype 1 provided the LLM with a **visual checklist of beats** (using checkboxes "☒ Beat 1, Beat 2, …") in the chapter prompt [82] , and Prototype 2 passed a

structured list of `remaining_beats` and `remaining_subplots` in a YAML front-matter for each piece [83]. This ensured the model knew exactly which plot points to cover before finishing the chapter. In the current implementation, this strategy was **not carried over**. The draft prompt only indicates the number of beats remaining, not their content [65]. Consequently, the LLM isn't explicitly reminded of each beat's details during writing. This is a regression in guided prompting – the prototypes' method of explicitly surfacing outline elements was a strong guardrail for story coherence. The absence of a beat-by-beat scaffold in the prompt is a primary reason context got lost in the final drafts.

- **Retry and Validation Logic:** Both prototypes had robust validation loops for each chunk of text. For instance, Prototype 1 would retry if a piece was too short or ended improperly, up to 3 times [84] [85]. Prototype 2 expanded this with up to 5 retries and more conditions: detecting duplicate paragraphs, missing required beats, etc., each time appending specific corrective instructions (e.g., "expand this section", "finish the scene", "remove repetition", "cover remaining beats") [86] [73]. The current system does implement **some of these improvements**: it allows up to 5 retries and checks for a proper ending punctuation and sufficient length, issuing targeted retry prompts (like instructing to fix the tag or not to skip ahead) [87] [66]. It also includes a **duplicate text detector** with a similarity threshold (Jaccard 85%) to avoid the model rehashing paragraphs [68], which mirrors Prototype 2's fuzzy duplicate check (they both set ~85% similarity) [86]. These show that **technical guardrails** from prototypes were largely adopted – indeed, the piece outputs in the test draft adhere to the tagging and are free of obvious mid-sentence breaks or repeated paragraphs, thanks to these checks.

However, one area of divergence is **content validation**: Prototype 2 specifically checked if a required beat or subplot was missing from a piece and would prompt the model to include it [73]. The current implementation does *not* have a check like "did the piece cover at least one beat or use up the beats?". It only tracks length/format. Thus, if the model skipped a beat, the system wouldn't notice, whereas Prototype 2 would explicitly catch that and tell the model to "cover remaining beats". This is a notable prototype feature that did not make it into the new system, contributing to outline deviations.

- **Cost Tracking:** Prototype 1 introduced a `CostTracker` to tally token usage and cost, and Prototype 2 expanded it to log each API call to a CSV [88] [89]. The current project did implement this idea: there is a `CostLogger` that writes a CSV line for every LLM call with model name, prompt tokens, completion tokens, and USD cost [72]. In the provided run logs, we can see each piece generation and outline call being logged with token counts. This is a positive carry-over; it helps developers or users analyze the cost and possibly optimize prompts. It doesn't affect story continuity, but it's a successful adoption of a prototype improvement.

- **Dynamic Piece Sizing:** Prototype 1 fixed 8 pieces per chapter, but Prototype 2 calculated pieces based on desired piece word length (~350 words each) and target chapter length [90] [91]. The current C# version follows the dynamic approach: it sets `PieceSizeWords = 350` and computes `pieceCount = ceil(targetChapterWords / 350)` [92]. For example, with ~3000 words target per chapter, it might use 9 pieces (as seen in the example "piece 1/9" markers). This shows the project learned from Prototype 2 to be more adaptive in chunking. The result is more flexibility and potentially less risk of hitting token limits or having awkwardly short/long pieces. This improvement was implemented without issues.

- **Piece Tagging and Formatting:** Prototype 1 and 2 used distinctive markers to label pieces (Proto 1: `[C##-P#]` tags, Proto 2: YAML `--- piece x/y ---` headers) [93] [94]. The current system chose the `--- piece x/y ---` style, aligning closely with Prototype 2. It successfully ensures pieces can be concatenated unambiguously. The model is generally compliant in repeating the tag at start (the code even auto-corrects it after 3 tries if needed) [87]. This is another area where prototype guidance was followed, yielding a stable output format.

- **Coverage of Outline in Prompts:** In Prototype 2, each piece's prompt explicitly listed *remaining beats and subplots by content* (not just count) as we discussed [77]. The current system only provides counts. This is a **key difference** that has story implications. Essentially, the current model is writing somewhat **open-endedly** within each chapter, whereas Prototype 2 was writing with a checklist in hand. Writers comparing the two would notice that the prototype's output tended to explicitly tick off plot points (it even had the model change ☒ to ✔ as it covered beats in Proto 1) [95]. The new system's output might feel more fluid or creative, but also more prone to drifting from the outline since the model isn't anchored to a to-do list. The absence of this checklist is arguably the biggest "missed lesson" from the prototypes.

- **Single POV and Style Consistency:** Prototype analyses noted some gaps remained, such as enforcing POV consistency or adapting context window size [81]. The current project doesn't explicitly address POV either, so that gap remains. One thing the current system did maintain from recent updates (perhaps from internal iterations) is a **"single-heading guard"** – essentially making sure the model doesn't start adding its own chapter titles or duplicating headings. By controlling the format tightly (the model must start with the piece tag and nothing else), it prevents issues like the model generating extra headings. This wasn't explicitly in prototype docs, but it is something that the current code's tag validation covers (if a piece started with "Chapter 1:" or something instead of the tag, it would fail the regex and retry). So format guardrails are strong.

- **Duplicate Content Handling:** Prototype 1 lacked it, Prototype 2 introduced paragraph-level duplicate detection [96] [86]. The current system implemented a similar check using Jaccard similarity on word sets [97]. This was clearly a learned lesson to avoid repetition. In practice, this means if the model tries to recapitulate a previous paragraph (which GPT sometimes does to reorient itself), the system will notice (>85% similarity) and trigger a retry with an instruction to avoid repetition. This improves the reading quality and coherence (no deja vu paragraphs). This feature from Proto 2 was effectively ported (the `DuplicateDetector.AreSimilar` function).

To sum up, **technical enhancements (piece sizing, cost logging, tagging, end-of-sentence enforcement, anti-duplication)** from the prototypes were largely adopted, improving reliability and efficiency. Where the current project falls short is in **content adherence and narrative consistency** – prototypes had mechanisms to enforce or at least heavily hint at covering outline points (beats/subplots) in each piece, which the new system omitted. They also did not carry over any notion of checking the final product against the outline or preventing the kind of identity drift we saw.

One could say the developers focused on infrastructure (format, tokens, chunking) improvements from the prototypes, but **overlooked some of the storytelling safeguards** that prototypes (especially Proto 2) experimented with, like dynamic prompts that keep track of remaining story obligations. The result is a system that technically runs smoothly but at times **does not fulfill the narrative intent** as faithfully as the prototypes did.

Summary

In reviewing the Novelist pipeline, we identified several gaps and failure points that lead to story context being lost or distorted. Below is a ranked summary of these issues by their impact on the writing outcome, along with notes on how the system's guardrails failed to prevent them:

1. **Protagonist Identity Confusion (Ethan vs Ian)** – *Impact: Highest.* The main character's name and identity diverged between outline phases, causing the final story outline (and likely the draft) to treat what should be one person as two characters. This is a fundamental breakdown in continuity. The guardrail that failed here was the **lack of cross-phase reference**: the system provided no mechanism to carry a chosen name forward or reconcile different names. Once "Ian" was introduced in the arc without later correction, all subsequent steps propagated the error. This kind of slip-up can derail the entire narrative, as the protagonist is central to every scene. A writer encountering this would have to do significant rewriting to fix the narrative perspective. In essence, the pipeline's design of independent phase outputs broke the story's spine – a critical guardrail (character consistency) was missing.

2. **Mismatched Character Roles (Mr. Porter vs Henry, etc.)** – *Impact: Very High.* Similar to the protagonist, a key supporting character was doubled up under two names. The intended mysterious resident was called Mr. Porter in arc/beats and Henry Whitaker in the character list/beats, making the outline's cast confusing and the story internally contradictory. This not only affects character continuity but also muddles plot points (e.g., who found the photograph – was it "Henry" or "Mr. Porter"? The outline says Henry, but the arc implies Mr. Porter knew about it). This occurred because the **schema allowed the arc to introduce characters outside the character roster**, and no guardrail brought them into alignment. For a writer, this means the outline cannot be trusted to have one consistent set of characters – a major flaw. The guardrail solution would have been a unification step or at least guidelines that arc text shouldn't use specific names for characters that aren't defined. Its absence was a significant oversight.

3. **Outline Plans Not Fully Reflected in Draft** – *Impact: High.* Many carefully outlined beats, subplot details, and thematic elements did not make it into the draft due to prompt omissions. For example, some beats were effectively dropped because the drafting prompt never mentioned them, and the model didn't organically incorporate them. Subplot threads that were present in the outline could fade in the draft (e.g., the "old photograph" subplot might get scant mention in the actual prose beyond the outline summary). The **guardrail that broke down** here was the **explicit guidance of the model by the outline**. Earlier prototypes used checklists or remaining-beat lists to tether the draft to the outline, but the current system removed those tethers. Consequently, the model had freedom to wander or focus on certain things and ignore others. The impact on the writer is that the draft can come out missing scenes or payoffs that the outline promised, requiring the writer to fill gaps or rewrite sections to insert those plot points.

4. **Character Depth and Traits Diluted** – *Impact: High.* The rich character traits and arcs defined in the outline weren't explicitly used when writing scenes. For instance, if a character was described as having *"a childhood trauma and a tendency for internal monologue"* in their profile, the draft might not reflect this at all, making the character on-page flatter than intended. Themes like *redemption or guilt* that were part of the planning could also diminish. The pipeline lacked guardrails to carry these nuances into the prose (no prompt reminders or constraints to steer characterization). For a writer,

this means the AI draft may feel generic or "off" in portrayal, not matching the character sheets. They would then have to revise dialogue, internal thoughts, and descriptions to realign with the intended traits – essentially rewriting character moments that the outline already determined. This gap reduces the usefulness of the outline: why define traits if they don't show up in story? The failure here was in **context propagation** – the system didn't guard against loss of these details between outline and draft.

5. **Unresolved or Weak Subplot Conclusions** – *Impact: Medium.* While subplots were at least present in each chapter outline, there was no guarantee the final chapters properly resolved them. For example, the outline indicated residents vanishing and identity rewriting as threads, but the draft might end without fully explaining these if the model wasn't nudged to do so. This is partly due to the lack of explicit instructions to *resolve threads by the end*, and partly due to the identity confusion which made the resolution (Ethan becoming the new resident "Ian") very muddled. The guardrails for subplot continuity (the S1/S2 labels) were clever in the outline, but a guardrail to ensure **subplot resolution** was absent. The impact is that a writer might find the ending unsatisfying or confusing and have to add closure scenes or clarifying dialogue to tie up loose ends. It's slightly lower impact than main plot issues, but still important for narrative completeness.

6. **Over-Rigid Chapter Structure** – *Impact: Medium.* The enforcement of exactly three beats per chapter and evenly distributing chapters across acts could lead to pacing issues. While this didn't cause a logical inconsistency, it can cause narrative issues like rushed scenes or filler content. The model might stretch a minor moment into a full beat just to have three, or it might condense two important moments into one beat to not exceed three. This rigidity is a design choice that acts as a double-edged guardrail: it keeps chapters uniform (perhaps easier to edit and predict) but at the cost of natural story flow. The impact on the writer is scenes that might feel artificially segmented; they might choose to rearrange or merge chapters afterward. It's a moderate issue – not a blatant error, but a potential quality concern. In this case, the guardrail (structure uniformity) itself might be too strict, meaning the solution is not that it broke, but rather it might need loosening.

7. **Stylistic Inconsistencies or Tone Shifts** – *Impact: Low to Medium.* Since the draft heavily relies on the model's innate style emulation ("You are Stephen King"), it generally stayed in a consistent narrative voice. However, subtle style guide elements (like avoiding certain phrases or using specific motifs) from presets were not actively enforced beyond that. If the model deviated (say it started using bold text or something against the preset rules, or it didn't include an expected motif), the system wouldn't catch it. This is a minor issue as GPT-4 is usually quite good at maintaining style once given a persona. We didn't observe a major style break in outputs. Still, any instance of voice drift (e.g., suddenly getting overly flowery or changing perspective) would be on the writer to fix. Prototypes didn't solve this either, and it appears relatively minor compared to the structural problems above.

In conclusion, the **most impactful gaps are those in continuity and adherence to the outline's intent**. The places where **guardrails "broke down"** were often where none existed: the design did not anticipate needing them. Character identity continuity, for example, had no safeguard and thus failed. Outline-to-draft consistency was assumed rather than enforced, leading to lost context. The system excelled at low-level consistency (format, tagging, no duplication) but lacked high-level consistency checks.

For the developers, the priority should be fixing the cross-phase data flow: for instance, using the same canonical names throughout (perhaps linking arc and character phases or merging them), and feeding the outline's beats and character traits into the drafting prompts. For the writers, understanding these gaps explains why a draft may veer off or require heavy edits even if the outline looked solid. By addressing these issues – reintroducing content-focused guardrails from the prototypes and enforcing continuity rules – the Novelist system can better maintain story context from the first idea to the final manuscript.

Sources: The analysis references the Novelist code and outputs, including outline JSON snapshots and prototype design documents, to illustrate where context was lost or preserved. Key examples include the protagonist naming inconsistency [13] [98], the prompt templates lacking outline details [99], and lessons from prototypes about maintaining beat checklists [82] [77]. These demonstrate the points above and underscore the recommendations for improvement.

---

1 2 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/samples/thedoor.project.json

3 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/OutlineBuilderService.cs

4 5 6 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/PremiseExpanderService.cs

7 8 13 14 22 23 24 25 98 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/docs/test_artifacts/outline_20250708064841_CharactersOutlined.json

9 10 11 12 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/ArcDefinerService.cs

15 16 17 18 19 20 21 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/CharactersOutlinerService.cs

26 27 28 29 30 33 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/SubPlotDefinerService.cs

31 32 34 40 41 42 49 50 51 52 53 54 55 56 57 58 59 60 61 80 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/docs/test_artifacts/outline_20250708064841_StructureOutlined.json

35 36 37 38 39 43 74 75 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/BeatsExpanderService.cs

44 45 46 47 48 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.OutlineBuilder/StructureOutlinerService.cs

62 63 64 65 66 67 68 69 70 71 79 87 92 99 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.DraftBuilder/DraftBuilderService.cs

72 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/drafts/cost_log.csv

73 77 81 83 86 89 91 94 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/docs/analysis/Summary_B.md

76 82 84 85 88 90 93 95 96 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/docs/analysis/Summary_A.md

78 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/author_presets/StephenKing.json

97 raw.githubusercontent.com
https://raw.githubusercontent.com/bilols/thenovelist/refs/heads/main/src/Novelist.DraftBuilder/DuplicateDetector.cs