ChatGPT

Rebuilding the Timeline and Show Logic (20-Minute TOKN News Episode)

1. Timeline Rebuild with Latest Corrections

We have overhauled the timeline-building code to incorporate all in-progress fixes, aiming for a cleaner, fully functional structure. The new `timeline_builder` ("Full Rebuild, Clean Stack" version) implements a clear sequence for each news segment, reducing duplication and potential bugs. Key improvements include:

- **Intro Sequence at Start:** The show now **always begins** with the Vega intro followed by Chip's greeting, as requested. In the updated `build_timeline`, if it's the first segment of the show (`show_intro` and segment type "headline"), we prepend a **Vega welcome line** and **Chip's time-of-day greeting** to the timeline [1]. This ensures every episode consistently starts with Vega's booth announcer intro and then Chip's on-camera welcome.

- **Daytime & Holiday Awareness:** Chip's greeting remains time-of-day aware ("Good morning/afternoon/evening"), and we will reintroduce **holiday mentions** to preserve that personal touch. In the previous version, Chip's GPT-generated intro explicitly checked for major holidays like New Year's, Independence Day, Halloween, etc., appending a line like *"Today is Thanksgiving,"* if applicable [2] [3]. In the new static `_chip_opening_line`, holiday call-outs were omitted [4] [5]. We can address this by adding a quick holiday check in `_chip_opening_line()` so Chip can say, for example, *"Happy New Year's Day"* before the welcome when appropriate. This way we keep the intro **holiday-aware** as you requested, without needing a complex GPT prompt.

- **Primary Story Introduction:** After the Vega and Chip intro lines, Chip now delivers a **GPT-generated "story intro"** to frame the main headline [6]. This uses `gpt_chip_story_intro` to succinctly explain **why the headline matters today** in 1-2 sentences [7] [8]. Importantly, the prompt instructs GPT *not* to toss to the anchor (Chip doesn't name the next speaker here) [9] – Chip simply sets context and urgency for the story. This extra framing replaces what used to be a part of Chip's greeting; it makes the handoff to the anchor smoother and more informative.

2. Streamlined Anchor Toss Logic

The **toss logic** has been simplified and made more robust to eliminate redundancy and bugs. Previously, we had separate conditions for Chip tossing to the primary anchor depending on whether it was the show start or a mid-show segment [10] [11]. This led to complex branches and occasionally double-tosses or missed tosses. We've unified this into one clear step:

- **Single Toss Point:** In the new timeline builder, Chip's toss is generated once, right after the story intro. We use a new helper `chip_toss_line(next_anchor)` that provides a short handoff phrase

depending on who the next anchor is (and the show mode) [12] [13]. For example, if Reef is the primary anchor, Chip might say *"Reef, what's the DeFi angle here?"* [14]. This is appended as a `"chip_toss"` entry in the timeline [15]. Because we consolidate toss generation in one place, we avoid the redundant code paths that existed before.

- **No Toss to Self:** We also guard against the edge-case bug of Chip tossing to himself. In cases where Chip *is* the primary anchor for the story (meaning Chip will continue as the analyst), we **skip the toss**. The new code will check `if primary_anchor != "chip"` before adding a toss. (In the current branch, we noticed `chip_toss_line` would return a default string even for Chip, e.g. *"chip, what's your read?"*, which we will suppress in this scenario.) If Chip remains lead for a segment, he simply proceeds with reaction and analysis lines without an unnecessary toss. This preserves logical flow and fixes the prior bug where Chip might awkwardly address himself.

By centralizing the toss logic and adding the self-toss check, the handoff between Chip and the chosen anchor is now handled **cleanly and only once** per segment. This reduces code bloat and prevents the duplicated "toss" lines that were causing confusion.

3. Duo-Anchor Crosstalk Simplification

We have refactored the **duo anchor crosstalk** into a dedicated routine to eliminate redundancy. If a secondary anchor is present (for a two-anchor panel discussion), the new `timeline_builder` calls a single function `_build_duo_crosstalk(primary, duo, ...)` to generate the entire back-and-forth sequence [16] [17]. Improvements in this area:

- **Unified Duo Exchange:** The `_build_duo_crosstalk` function encapsulates the logic for a **multi-turn dialogue** between the primary and secondary anchors [18] [19]. It uses GPT (`gpt_duo_line`) for each turn, alternating speakers and modes (reaction, analysis, transition, close) to create a natural conversation [20] [21]. This is far cleaner than having bits of duo logic scattered in multiple places. All the nuance – turn-taking, persona styles, contextual references – is handled inside this one routine.

- **Redundancy Removed:** Previously, we attempted duo interactions in an ad-hoc way (and even discussed a possible "round 2" of duo crosstalk in comments). Now, with one function generating a full exchange, we avoid any duplicate or inconsistent duo handling. The code also filters out repeated jargon between turns (e.g. if "volatility" or "liquidity" was already mentioned, it skips lines that repeat those terms) [22] [23]. This stops the anchors from sounding repetitive and ensures each contribution adds something new.

- **Consistent Tone and Roles:** The crosstalk builder also checks if the same anchor spoke last in the solo portion and swaps roles if needed [24] – ensuring that the conversation starts with the other anchor to keep it dynamic. By consolidating these rules, the duo segment is both more **maintainable** and more engaging, without the bugs we saw from earlier redundant code.

All duo lines are then appended in one block, and the function returns a list of timeline entries for easy insertion [25]. In short, duo anchor handling is now **modular and robust**, eliminating prior duplicated logic

that caused confusion (for instance, conflicting “tosses” or misordered reactions when two anchors were present).

4. Code Simplification without Quality Loss

We have made the `pd_controller` and `timeline_builder` logic leaner while **enhancing output quality**. Large, monolithic functions were broken down, and GPT is leveraged more broadly so we can simplify static logic. Here’s how we achieved this balance:

- **Lean PD Controller:** The PD controller (`run_pd`) is now primarily focused on **high-level segment decisions** and state tracking, rather than constructing dialogue. It determines segment type (`headline` vs `breaking` vs `show_intro`), selects anchors (including duo if appropriate), and sets flags like `allow_bitsy/allow_vega` based on daypart or breaking news [26] [27]. We’ve kept this logic but trimmed any heavy lifting that didn’t belong there. For example, intro logic and detailed tone adjustments have been moved out of PD into the timeline builder. PD now just sets `show_intro=True` for the first segment and flips it off thereafter [28], and it no longer tries to handle things like **Chip’s follow-up lines or tosses** – those are handled in the timeline assembly. This separation of concerns makes each part of the pipeline easier to manage.

- **Smaller, Focused Functions:** In `timeline_builder`, we broke the process into discrete steps (intro, toss, reaction, analysis, etc.) with clear responsibilities. Helper functions like `_vega_ident_line()` and `_chip_opening_line()` provide fallback text for intros [4] [5], and similarly `chip_toss_line()` centralizes toss phrasing [29] [13]. By having these bite-sized helpers, the main `build_timeline` reads like a straightforward recipe for one segment, rather than a tangled 300-line function. This dramatically improves readability and maintainability.

- **Full GPT Writing & Analysis:** Wherever possible, we now let GPT generate the content to keep the quality high without manual scripting. The system will use the OpenAI writer for *analysis*, *transitions*, and *reactions* whenever the `USE_OPENAI_WRITER` flag is enabled [30] [31]. In practice:

- **Analysis lines** for the primary anchor are GPT-derived: `build_analysis_line` now calls `gpt_analysis` which produces up to 3 sentences of analysis tailored to that persona [32] [33]. If GPT fails or is off, it falls back to the deterministic template, but in normal operation this means more insightful explanations of the news.
- **Transition lines** between topics or to cameos are also generated via GPT (with persona-appropriate tone), ensuring smoother segues [34] [35].
- **Anchor reactions and quips** (the quick follow-up one-liners) likewise use GPT now (`gpt_anchor_react`) to inject a bit of personality or emotion consistent with the anchor’s character [36] [37].

Essentially, by **unlocking GPT across the board**, we offloaded a lot of hardcoded phrasing logic. The code is simpler (less manual string construction) while the output remains rich. The few areas we keep static are mainly for guaranteed critical lines like the initial Vega “Welcome to Token News” ident – which we want consistent each time for branding [4] [5]. Everything else, we trust GPT to handle under our guidelines.

This dramatically reduces the previous bloat of stored phrases and makes it easier to tweak output style via prompts rather than code changes.

- **Example of Simplification:** The earlier line builder had a complex deterministic method to form an analysis sentence by piecing together fragments and removing banned words [38] [39]. Now, if GPT is on, we simply pass the headline and synthesis to the model and get a fluent summary. The code path for GPT is one function call, with a fallback to the old method only if needed [40] [41]. This kind of toggle means we maintain quality without maintaining loads of custom logic – the best of both worlds.

Overall, we've **reduced redundancy** (no more duplicate intro or toss logic in multiple places) and cut down on "bloat" by letting the AI handle content generation. Yet, by carefully structuring the code and using fallbacks, we haven't compromised reliability – if the AI fails to return text at any point, we have static lines to fill the gap [42] [43]. The result is cleaner code and consistent output.

5. 20-Minute Show Structure

As discussed, we are targeting a ~20 minute show format. Given typical speaking rates, this corresponds to roughly **4-5 segments** of content (assuming ~4-5 minutes per story including intros, analysis, and banter). Here's how the new logic helps achieve that and what steps to take:

- **Segment Chaining:** We will treat each major headline as one segment in a continuous show, using the **persistent state** to link them. The first headline of the show triggers `segment_type = "show_intro"` (so we get Vega+Chip intro). Subsequent headlines default to `segment_type = "headline"` segments with no intro [44] [45]. The `director_state` keeps track of what's been played and increments a `cycle_index` for each segment (ensuring things like ads or special inserts trigger at the right intervals) [46] [47]. We will verify that `cycle_index` is incremented each cycle (and implement that if missing) so that by ~4 segments we know 20 minutes have passed.

- **Anchor Rotation & Continuity:** The PD controller's logic (unchanged) already rotates anchors based on story domains and prevents the same anchor from appearing back-to-back too often (using the `last_anchor_used` in state and domain-based selection) [48] [49]. This means across 4-5 segments, viewers will hear from different expert personas, keeping the show lively and covering various angles. Chip remains the constant host tying segments together. Chip's **follow-up toss** at the end of each segment is now handled outside the timeline builder – we removed the old static "Next up..." line to avoid clutter [50] [51]. Instead, Chip's closing toss or handoff to the next story can be generated when starting the next segment (or we can simply rely on the intro of the next segment to serve that purpose). In testing, we should ensure the flow from one story's end to the next story's start feels natural.

- **Duration Control:** To explicitly enforce ~20 minutes, we have a couple of options. One is to set a fixed number of segments per show (e.g. 4 segments + 1 ad break after segment 2 or 3). The `should_insert_ad` function is configured to return True every 5 segments [52], but since 5 segments might overshoot 20 minutes, we could adjust that or trigger a short ad around midway for pacing. Another approach is to monitor the cumulative length of the audio files generated for each segment – since we render each segment's audio via `render_audio_blocks` – and stop when the total nears 20 minutes. In practice, the simpler method is just picking the top N headlines (N≈4) for

each show. We can automate the pipeline to fetch the latest top 4 headlines, then call the script engine for each in sequence. Chip's intro covers the scope ("today's top stories"), and each story plays out. After the last story, we might add a brief Chip sign-off if desired (we can generate one via GPT or have Chip simply say "That's all for today on TOKN News. Thanks for joining us." as a static line).

By structuring the show as multiple segments, we ensure we hit the ~20 minute runtime without a single segment running overly long. The new code is designed to seamlessly stitch segments: after the first, `show_intro` is False and Chip will either toss to the next anchor or just let the next segment start (depending on how we call the generator). We'll test this sequencing with actual run-throughs. The persistent state file `director_state.json` will preserve continuity (so Vega or Bitsy cameos don't repeat too frequently, etc.) across the segments of one show [53] [54].

6. Deployment: New PM2 Process and Directory

To safely roll out these changes, we will **deploy the updated code in a fresh environment** rather than overwriting the old one. This means:

- **New Directory:** Create a new code directory (for example, `/var/www/toknnews_v2` or a similarly shortened path) and clone the GitHub repository's **source branch** there. Using a shorter path (and a clean folder) avoids any conflicts with cached bytecode or leftover files in the old `/var/www/toknnews-repo` directory. It also lets us keep the old version running in parallel until the new one is confirmed stable, which is a good precaution.

- **Update Configs:** In the new directory, update any path references or environment configs if needed. Notably, the old script had a manual `sys.path.append("/var/www/toknnews-repo")` [55] [56] to ensure imports worked. In the new setup, we can adjust that to the new path or, better, remove the need for it by installing the package properly (e.g. using `pip install -e .` if we convert it to a package, or simply ensuring we launch the script from the correct working directory so relative imports resolve). The goal is to make the runtime as clean as possible.

- **PM2 Process:** Register a new PM2 process for the updated script (e.g. `pm2 start new_tokruntime.json` or the relevant entry script for the news engine). This new process will run alongside the old one initially. We'll have it listen on a different port or output location as appropriate. Once it's up, we can test end-to-end: generate a full 20-min show, check logs for any errors (especially around the new timeline logic or any missing pieces), and verify the audio output and transcripts are as expected (intros present, no duplicate toss, etc.).

- **Cut Over:** After validation, we can gracefully shut down the old PM2 process and switch any incoming requests or cron triggers to point to the new process. Because we used a new directory and process name, rollback is simply a matter of toggling processes if needed – giving us a safe deployment path.

- **Shorter Domain Path Rationale:** By using a fresh, shorter path, we reduce any risk related to path length or misreferenced directories. It's mostly a precaution – some systems have limits or odd behaviors with very long paths, and a cleaner name makes commands easier. More importantly, it

psychologically assures us we're not mixing old and new code. The old `/var/www/toknnews-repo` may contain redundant files and backups that we won't copy over, so the new directory will have only the necessary, up-to-date code.

In short, **starting a new PM2 process in a new directory** isolates the new version, making it easier to debug without affecting the live show. Once we're confident, we'll retire the old one.

7. Cleaning Up GitHub Redundancies

Finally, regarding the "disgusting" redundancies in the GitHub repo: we will perform a cleanup to streamline the codebase. Over the course of development, multiple backup folders (like `sync_sets/v1.0/dev/backup_*`) and parallel versions (such as `persona/timeline_builder.py` vs `persona/timeline_builder.bak` or the `sync_sets/v1.0/current/processing` copy) have accumulated. This can be confusing and error-prone. Here's the plan:

- **Merge Branches:** We'll consolidate the `source` **branch** (which contains the latest "clean stack" rebuild) into the main codebase. That likely means making the `sync_sets/v1.0/current/processing/...` the primary implementation and removing the old `backend/script_engine/...` versions. For example, the new `timeline_builder` under `sync_sets/.../processing/` will replace the older one under `backend/script_engine/persona/` to avoid divergence. The same goes for any other modules that have dual locations.

- **Remove Obsolete Files:** All those dated backup directories and "OLD_BACKUP" files will be pruned from the repository. Since we use Git for version control, we don't need to keep zip-style backups in the repo itself – that's what commit history is for. We'll keep maybe the most recent stable tag, but delete the rest of the clutter. This will reduce repository size and make it clear which files are active. (For instance, we see `timeline_builder.bak` sitting next to the main file – that can be safely removed once we confirm the new code works.)

- **Update References:** After cleaning, we'll scan through imports to ensure nothing still points to a deprecated path. For example, the code references `script_engine.persona.timeline_builder` in several places – once we finalize the location of `timeline_builder.py` (likely keeping it in `script_engine/persona/`), we'll ensure all import lines use that single source. The `sys.path.append` hack used to load from `/var/www/toknnews-repo` [55] can be dropped if our module structure is consistent and installed properly.

- **Repository Organization:** We might also take this opportunity to organize the repo into a more standard Python package format (with a setup or at least a clear `toknnews_engine` module), but at minimum we will have one copy of each component. Redundancy in GitHub not only confuses developers but can also confuse the runtime if paths are not managed carefully (e.g., two different `pd_controller.py` could load unpredictably if both are on the path). Cleaning it up ensures the PM2 process is definitely running the intended code.

By doing "whatever is best" for these redundancies, I recommend a thorough prune and commit of the cleaned structure. We'll document this in the repo's README so future devs know which directories to use.

The end result will be a leaner GitHub project that reflects the simplified codebase – easier to navigate and less error-prone when making updates.

Sources: The implementation details above are based on the latest committed code in the source branch and our planned modifications. Key code excerpts include the new intro logic in `timeline_builder` [1], the unified toss with `chip_toss_line` [15], the GPT-driven duo crosstalk routine [18] [21], and the updated line builder enabling full GPT content generation [31] [35]. These confirm that the new design simplifies previous complexities while aligning with the show's creative goals. All changes will be tested under the new PM2 process to ensure a smooth 20-minute broadcast with no regressions in character or quality.

---

[1] [4] [5] [6] [12] [13] [14] [15] [18] [19] [20] [21] [22] [23] [24] [29] timeline_builder.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/sync_sets/v1.0/current/processing/script_engine/persona/timeline_builder.py

[2] [3] [10] [11] [16] [17] [25] [42] [43] [50] [51] timeline_builder.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/backend/script_engine/persona/timeline_builder.py

[7] [8] [9] [32] [33] openai_writer.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/sync_sets/v1.0/current/processing/script_engine/openai_writer.py

[26] [27] [28] [48] [49] [53] [54] pd_controller.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/backend/script_engine/director/pd_controller.py

[30] [31] [34] [35] [36] [37] [40] [41] line_builder.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/sync_sets/v1.0/current/processing/script_engine/persona/line_builder.py

[38] [39] line_builder.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/backend/script_engine/persona/line_builder.py

[44] [45] segment_router.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/backend/script_engine/director/segment_router.py

[46] [47] director_state.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/backend/script_engine/director/director_state.py

[52] ad_logic.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/backend/script_engine/director/ad_logic.py

[55] [56] script_engine_v3.py
https://github.com/TokNNews/toknnews/blob/2078288ffe3b230429a5636c6e351031efbb7b64/sync_sets/v1.0/current/processing/script_engine/script_engine_v3.py