ChatGPT

Overview of OA Agent and Model Configuration

The **Orchestrator Assistant (OA)** operates as the "thinking" layer, leveraging a set of local language models (LMs) assigned to specific functional *slots*. According to the updated **AGENTS.md** spec, the system uses a **per-slot model routing** architecture. Each logical slot – `local_agent` (general reasoning/tool use), `embedding` (vector embeddings), `reranker` (relevance reranking), and `theology` (Bible/theology expertise) – is wired to a designated model and can independently use one of the local inference providers (either **LM Studio** or **Ollama**) [1]. By default, the **Granite** model family (IBM Granite 4.0 series) is used for general-purpose slots, while a specialized **theology expert** model is reserved for the theology slot [1] [2]. In practice, this means:

- **Local Agent slot:** uses the *Granite4:tiny-h* model (a smaller Granite chat model) for general reasoning [2].
- **Embedding slot:** uses *Granite-embedding-278M* for generating embeddings [3] (with an optional **BGE m3** model available for multilingual/Bible-specific embeddings, though BGE is not the default) [4].
- **Reranker slot:** uses *Granite4:tiny-h* as well, to score and rerank retrievals [2] (a Qwen model exists as a fallback but not primary).
- **Theology slot:** uses *Christian-Bible-Expert-v2.0-12B*, a 12B-parameter specialist model for scriptural commentary and theology tasks [4]. This larger model runs via the **LM Studio** backend (due to its size and domain) and is **only** used for theology or Bible context queries.

All model inference calls are local – no external API – and go through providers on `127.0.0.1`. The system is configured such that **LM Studio** handles at least the theology model (and in the current setup, appears to be the primary provider for all slots), while **Ollama** can serve Granite models if enabled [1] [5]. In fact, environment toggles like `LM_STUDIO_ENABLED` / `OLLAMA_ENABLED` and per-slot provider env vars (`LOCAL_AGENT_PROVIDER`, `THEOLOGY_PROVIDER`, etc.) determine which service actually serves each slot [6]. The **LM Router** module in `agentpm` ensures that each query is routed to the correct slot/model based on its type: e.g. embedding requests go to the embedding slot, Bible queries route to the theology slot, math or tool-related queries go to local_agent, etc. [7]. This guarantees OA uses the **right model for the right task** under the hood.

At runtime, the **LM Studio** instance is indeed active and serving the models. The latest system snapshot shows the LM provider endpoint set to `http://127.0.0.1:9994/v1` (the LM Studio API) and reports the provider as "**lmstudio**," with the LM service in a healthy `lm_ready` state [8]. In summary, OA's mental model configuration matches the designed architecture: Granite models (via local providers) handle general reasoning, while the specialized Bible model (via LM Studio) is plugged in for theology, all coordinated by a routing layer.

Detected Discrepancies and Misconceptions

Our deep dive uncovered a few **discrepancies** between the documented configuration/assumptions and the runtime's recorded state, mostly minor and likely resolvable:

- **Reality Green Status Mismatch:** There is an inconsistency in how the system's "green status" (overall health) is recorded. The **Handoff Kernel** snapshot (`HANDOFF_KERNEL.json`) indicates `reality_green: true` with all critical health checks passing [9], and indeed the detailed `REALITY_GREEN_SUMMARY.json` shows **all checks passed** (no guard failures) implying a healthy state. However, the **PM bootstrap state** (`PM_BOOTSTRAP_STATE.json`), which the planning agent (PM) uses, appears to interpret the same data differently – it ended up labeling `reality_green` as **false**. This caused an "OA State Mismatch" warning in the reality summary: it notes that **PM_BOOTSTRAP's health reported true vs. REALITY_GREEN reported false**, or vice versa [10]. In other words, one part of the system believed everything was green, while another part flagged a degraded state. This is clearly a stale or corrupted assumption in one of the surfaces. The evidence suggests the **planning snapshot didn't get the updated green status** – possibly a minor bug or timing issue in how `generate_pm_bootstrap_state.py` computes overall health. This discrepancy is acknowledged by the system (surfaced as a failed check for OA state consistency) [10] and should be investigated so that OA and PM share the same understanding of the kernel health state.

- **DMS–Share Alignment Note vs Reality:** A similar stale note is visible in the Handoff Kernel's notes. The kernel JSON's `notes` section marks `dms_share_alignment`**: "BROKEN"** `with a mismatch count of 1` [11]`, implying that the documentation in the database (DMS) and the files in` **share/** `were out of sync by one item. In reality, the current alignment check shows **no mismatches** – the DMS and filesystem are fully aligned (the **DMS Alignment** check returned` **ok: true** `, with **` **dms_share_alignment** `: "OK"` **and no missing or extra files listed). All four alignment guards (DMS, bootstrap, share sync, backup) are passing. The "BROKEN" flag in the kernel notes is therefore outdated. It likely reflects a prior state (perhaps just before an ingestion or sync was performed) and wasn't cleared after the mismatch was resolved. In effect, the system is healthy but carrying a** stale flag** in the handoff notes. This could confuse OA if it trusted that note blindly, but fortunately OA's actual checks rely on the live summary, which shows alignment is fine.

- **Model Identity & Routing Assumptions:** Earlier OA runs (or older documentation) might have had blurred lines regarding which model handled which tasks – e.g. perhaps assuming **"Granite does everything"** or using a single default model for all queries. The current configuration corrects that: there is a distinct **theology expert model** for theological Q&A, and **BGE embeddings** are only used in specialized cases [4]. Any prior belief that the *BGE* model was the standard embedding model (or that it was active for general use) is now outdated – **Granite is the default** for embeddings, with BGE reserved for a "Bible" profile override [4]. Similarly, if any part of the system assumed that the theology questions would be answered by the same model as general chat, that is no longer true – they are explicitly routed to a separate 12B LM. Our review did not find evidence of OA making incorrect model choices in recent runs, so this shift in mental model has likely been internalized by the system. It's codified in the LM router rules (theology queries → theology slot, etc.) and in environment settings, so the **misconception risk is mainly historical**. Nonetheless, it's worth

ensuring all prompts and planning logic reflect these new roles (e.g. OA should be aware that for Bible commentary it has a dedicated expert model available, rather than trying to use the smaller Granite model).

- **Branch/Phase Tracking:** One minor point of potential confusion is that the kernel reports `current_phase`**: "24"** and **branch`: *feat/phase27l-agents-dms-contract*** [12] **. This means the repository is on a feature branch for Phase 27 work, even though the last completed phase is 23 and the current in-progress phase is 24. OA/PM might need to reconcile this: the planning agent likely knows from the MASTER_PLAN that Phase 24 tasks are underway, yet sees code for Phase 27 changes. This isn't exactly a "misconception" but rather an intentional concurrency (developing ahead). The** Orchestrator Assistant should not assume the project is already in Phase 27** just because of the branch name. There's no evidence it has done so – the bootstrap state correctly derives current_phase 24 – but it's a scenario to keep in mind to avoid planning confusion. The presence of Phase 27 features (like the AGENTS–DMS contract checks) while still officially in Phase 24 is by design, not an error.

In summary, the main discrepancies are **stale status artifacts** (the reality_green flag and the alignment note) and the need to ensure all components fully embrace the updated model-slot assignments. These are relatively small issues in an otherwise consistent picture, but they merit cleanup to prevent any lingering false alarms or confusion in OA's reasoning.

System Posture and Runtime Health (DB, LM, Exports, Hints)

The overall **system posture** right now is strong – essentially **healthy/green** across database, local models, and export surfaces, with all governance checks passing. The latest combined status snapshot (`pm.snapshot`) shows `overall_ok: true` and provides detail on each subsystem [13] :

- **Database (Postgres) Health: OK.** The primary databases are reachable and in `ready` mode [14] . All DB checks (connection, driver, required tables/stats) return true. There's no indication of partial outage or read-only mode; the DB is fully up. The kernel's health checks confirm this: e.g., the *DB Health* check reports "DB is reachable and healthy". No errors were found in the DB layer. In short, the **gematria and bible_db** databases are online, and the system's guard did not flag any integrity or connectivity issues. (For completeness, the `system_health.json` export lists `db.mode: "ready"` with no errors [15] .)

- **Local LM (Model) Health: OK.** The local inference stack is running properly. As noted, **LM Studio is active** on port 9994 and has the necessary models available – the snapshot shows `lm.ok: true` and `mode: "lm_ready"`, with the provider identified as **lmstudio** and no error messages [8] . That means OA can successfully call into LM Studio for both Granite and the 12B theology model. (Ollama is also installed, but it's either idle or not needed given LM Studio's coverage in the current configuration.) The **LM status** portion of the `/status` API would list each slot's model and service status; although we don't have that text in full here, we know from config that *Granite4:tiny-h* is the model ID for local_agent/reranker and *Christian-Bible-Expert-12B* for theology. Since **no LM errors** or offline flags appear, we infer all these models are loaded and responding. The **"LM slots summary"** in the UI would show each slot as healthy as well. Notably, the **system explanation** acknowledges

all four slots (local_agent, embedding, reranker, theology) and their statuses as part of the health snapshot [16] [17]. In sum, the **language model layer is fully operational**.

- **Kernel Mode: Normal (not degraded).** Because all critical guards passed, the system's mode remains NORMAL (i.e. not in a failsafe state). The Handoff Kernel JSON explicitly had `reality_green: true` and no failed checks [9]. For OA and Cursor, this means normal operations can proceed – no remediation mode is required. We see that none of the guard checks (DMS alignment, bootstrap consistency, share sync policy, backup system) failed in strict mode [9]. Each is marked `true` (pass) in the kernel health block, which is a prerequisite for **allowing forward progress**. If any had failed, the kernel would flip to degraded mode and OA would refuse to propose normal tasks [18] [19]. That is not the case here – **the kernel envelope is green**.

- **Guard Hints and Non-blocking Warnings:** The system posture includes a few **warnings (hints)** that do not stop operation but are worth noting. For example, the **"Repo Alignment (Layer 4)"** check detected a drift between planned and actual file locations (some files living in `scripts/code_ingest/` whereas the design expected them in `scripts/governance/`). The output shows a **FAIL** for this guard, but crucially it's in "⚠ HINT MODE: Warnings only (exit 0)". This means the issue (two integration files not in their forecasted paths) triggers a warning but **does not degrade the system**. The check even suggests updating the plan doc or moving the files, indicating this is a known low-priority discrepancy. Similarly, the **Control-Plane DB Health** check surfaced a hint about a materialized view refresh error (a PG hint about `mv_compliance_30d`), but still labeled control-plane health as passed. These kinds of hints are captured in the `REALITY_GREEN_SUMMARY.json`, and OA/PM can present them to the operator as non-critical issues. They do **not affect the "green" status**.

- **Exports and Surfaces:** All required JSON surfaces and exports appear to be present and up-to-date. The kernel notes list the expected surfaces it relies on – e.g. it references `share/PM_BOOTSTRAP_STATE.json` and `share/SSOT_SURFACE_V17.json` as required [20], and both are indeed present. The **"Share Sync & Exports"** check explicitly says *"All required exports present"*, indicating nothing is missing from the `share/` directory that should be there. Additionally, the **ledger verification** (discussed below) confirms that key exported files like `system_health.json`, `lm_indicator.json`, etc., are current. So the various pieces that OA or the Console might read (e.g. the reality summary, the PM snapshot, indicator data) are in place. This robust availability of **fresh data snapshots** means OA has the evidence it needs to explain the system state reliably.

- **Next-Step Proposals:** Because the system is not degraded, the OA is free to propose normal next actions. According to the OA design, on a new session it should read the kernel status and then propose tasks based on whether mode is NORMAL or DEGRADED [21] [22]. In NORMAL mode, that means proceeding with planned phase work. We haven't been given an example of OA's actual narrative output, but we can infer it will summarize Phase 24's context (since `current_phase: "24"`) and suggest the next unit of work from the MASTER_PLAN. The **planning agent (PM)**, which generates the 4-block instructions, also sees that all systems are go, so it won't be in a holding pattern. Essentially, **system health is not blocking any development tasks** – all lights are green or at worst yellow, not red.

In summary, the runtime health is **solid**: database is ready, local models are running, all mandatory guards are green, and only minor warnings exist. The Orchestrator can have confidence that the **OA's**

interpretations and PM's plans are based on a synchronized, healthy state. The presence of hints (like the repo path drift) is actually a positive sign of the system's observability – it catches these issues without stopping the pipeline. OA should (and likely will) mention such warnings in its state explanation, but it will also correctly report the overall status as OK. There is one exception: the aforementioned *reality_green flag confusion*, which is an internal inconsistency rather than an actual degradation. That is being flagged by the system itself, and we treat it as a quirk to fix rather than a true health problem.

Knowledge Base Registry and Document Sync Status

One major focus of recent phases (Phase 24 in particular) was to ensure the **Single Source of Truth (SSOT) documents, the knowledge base (DMS registry), and the** `share/` **filesystem** are all aligned. The evidence shows these efforts have been successful:

- **DMS ↔ Share Synchronization:** The **DMS (control-plane doc registry)** and the `share/` folder are now in sync with each other. The **DMS Alignment** check came back **OK** in strict mode – it found **no documents missing on either side and no unexpected extras**. This means every doc that the registry expects is present in `share/`, and every file in `share/orchestrator` (and other managed namespaces) is accounted for in the registry. Initially, after the Phase 18–23 restoration, there was concern that the registry vs filesystem could diverge (since `kb_registry.json` was a restored snapshot) [23]. Phase 24.B introduced an ingestion command and a guard (`guard_dms_share_alignment.py`) [24], which we now see in action. The outcome is that **not a single mismatch remains** (the earlier note of 1 mismatch has been cleared by ingestion). The kernel's SSOT surface was updated to report `dms_share_alignment: OK` as well, reflecting this alignment [25]. In practical terms, the **pmagent control-plane DMS is an authoritative index of docs, and our** `share/` **content exactly matches that index.** OA/PM can trust that there are no orphan docs floating around and no registry entries pointing to missing files.

- **AGENTS.md Consistency:** All copies/versions of **AGENTS.md** (the agent framework documentation) are consistent and up-to-date. There are multiple references to AGENTS.md in the repository (including in subdirectories and tests), and a snapshot in the knowledge base, so a special check verifies they are all in sync. The **"AGENTS.md Sync"** check passed with *"All AGENTS.md files are in sync"*. Additionally, a dedicated **AGENTS–DMS Contract** validation was run (part of the Phase 27.l work on agents metadata). This check reported **116 rows** and confirmed that *"All AGENTS.md rows satisfy pmagent control-plane DMS contract"* [26]. In other words, the content in `AGENTS.md` aligns with what the DMS expects for agent definitions and tags. There are no stray or outdated agent definitions. This is important because `AGENTS.md` is treated as a primary source; any divergence between it and the registry could cause confusion about agent roles or capabilities. Now they are explicitly kept in lockstep (likely via regeneration of the **AGENTS_REGISTRY_SNAPSHOT** and the table in AGENTS.md [27]). The ledger confirms the **root** `AGENTS.md` **file is current**, matching its recorded hash exactly.

- **Tracked Document Ledger:** A **document ledger** mechanism tracks key files (SSOT docs and critical JSON exports) to detect staleness. The **Ledger Verification** check reports *"All X tracked artifacts are current"* with no stale or missing entries. For example, it lists `AGENTS.md`, `MASTER_PLAN.md`, `DB_HEALTH.md`, etc., and for each shows the stored ledger hash vs current hash – in every case, the status is **"current"** (hashes match). This means none of the essential docs have been edited

without updating their references, and none of the expected outputs have been forgotten to regenerate. Notably, `MASTER_PLAN.md` (the high-level plan) and `RULES_INDEX.md` (governance rules) are up-to-date, which is vital for PM to make correct decisions. Similarly, the `system_health.json` and `lm_indicator.json` exports in `share/atlas/control_plane/` are current – these feed the UI and are also used in OA's reasoning about system status. The ledger did not flag any "stale" or "missing" status, and `missing: []` in its summary confirms nothing is lost. This **eliminates a whole class of errors** where OA might operate on an outdated picture of the system due to an old file.

- **Knowledge Base Hints & Coverage:** The **pmagent hint registry** is accessible and contains the expected hints. The reality summary includes a **"DMS Hint Registry"** item which passed, noting *"4 REQUIRED hints across 2 flow(s)"*. This indicates that the control-plane DMS's `hint_registry` table has four hints marked as required, belonging to two distinct flows. Likely these include hints like `ops.preflight.kernel_health` (ensuring kernel check before destructive ops) and possibly others for governance. The key point is that the **planning agent has these hints available** – PM will know, for instance, that before running migrations, Cursor must check kernel health (that's encoded as a required ops hint). This is part of the **DMS-first workflow** (Rule-053) where PM consults the DMS for known procedures and documentation before resorting to older methods [28]. The presence of the required hints confirms that integration. Also, the **DMS Metadata** check shows the registry's contents distribution: e.g., it has 144 critical docs, 262 high-importance, etc., and importantly `low_enabled=0` (no critical docs are accidentally disabled) [29]. The large number of "unknown" importance (783) just reflects many docs that are not categorized as critical/high – which is normal as the project has lots of auxiliary docs. There's no sign of something like a crucial doc being archived or missing – the **registry looks healthy** and properly configured.

- **Share Sync Policy:** The **Share Sync Policy** guard, introduced in Phase 24.C, ensures no untracked files linger in share and that we're not accidentally deleting needed files. This check passed in strict mode with *"No unknown or unsafe share/ docs"* [30]. The details did list two items under `missing_in_share` – `PM_HANDOFF_PROTOCOL.md` and `SHARE_FOLDER_ANALYSIS.md` [31] – but these are known planned documents that are not yet present. The guard logic likely treats missing tracked docs as an error only if they're critical; here it still reported overall OK (`"ok": true`) [32]. Essentially, the system knows these files *should* exist (perhaps from a plan or registry entry) but sees they aren't in `share/` – yet it's not considering it a failure at this time. This could be because we're on a dev branch or in a permissive mode for new docs. Other than that, there were no surprises: `extra_in_share: []` (no rogue files) and no policy violations. This suggests the new **allowlist derived from DMS** is working – only expected files remain in share, and anything not in the registry would be flagged. The two missing docs likely have DMS entries (hence expected) but just haven't been written yet; the team is aware and the guard passes with a note. From OA's perspective, this means it won't suddenly encounter an unknown file or lack an important one – the share folder's contents are exactly what the system of record (DMS) says they should be, with those two pending exceptions noted.

- **Bootstrap and OA State:** The **Bootstrap Consistency** check confirms that the generated `PM_BOOTSTRAP_STATE.json` is consistent with the SSOT phase info [33]. It shows current_phase "24" in both bootstrap and SSOT, with no mismatches [34]. This is important for OA/PM startup: it means the bootstrap envelope correctly reflects the project's phase and status (no stale phase pins, an issue that Phase 24.A addressed [35]). The **OA State** check (aside from the reality_green flag issue

already discussed) found no missing surfaces and no other mismatches – it specifically looked at OA's state versus kernel surfaces and apart from the green flag, everything lined up [36] [37]. So OA has the surfaces (like REALITY_GREEN_SUMMARY, PM_KERNEL, etc.) it expects and they contain matching data.

- **Backups and Other Infrastructure:** The **Backup System** check passed, indicating that a recent backup exists and the rotation policy is sane [38]. This means our periodic `make backup.surfaces` has run and older backups have been trimmed according to Phase 24.D rules. It's more of an ops concern than something OA actively uses, but it contributes to overall system confidence (and if the backup guard had failed, it would trigger degraded mode due to safety). Also, **WebUI shell** files are present (the simple sanity check for the web UI static files passed) [39], meaning the console interface has what it needs to run. These items show that housekeeping tasks have been performed properly in the background.

In summary, the **knowledge base and documentation synchronization is in excellent shape**. The project has effectively achieved a **"single source of truth"** alignment: the PM agent's internal registry, the `share/` content, and the various SSOT markdown files all reflect the same reality. OA and PM can rely on the fact that the docs they cite (governance rules, plans, agent instructions) are current and not duplicated in conflicting versions. Moreover, the system has built-in checks to immediately flag if any doc falls out of sync or if any expected knowledge base entry is missing – and currently all those checks are green. This greatly reduces the chance of OA harboring any **out-of-date beliefs** about the project. Even the AGENTS.md (which historically could have multiple copies) is now a single synchronized truth across the board.

Final Summary: Alignment of Mental Model with Reality

Overall, the **Orchestrator Assistant's mental model now aligns closely with the actual system state**. The configuration outlined in design documents (agents, models, phases, health guards) is **reflected in the runtime**:

- **Agent & Model Configuration:** The slot-wise model assignments in **AGENTS.md** (Granite for general tasks, a specialized 12B model for theology, etc.) are indeed what the system is running [1] [2]. The local model server (**LM Studio**) is deployed and serving those models, which matches the plan for an on-prem inference stack. Any previous ambiguity about which model handles which query has been resolved by the Phase 7F updates – OA uses the correct expert model for each context, via the LM router [7]. There's no evidence of OA using a wrong or outdated model; on the contrary, it has the infrastructure (env settings and router logic) to choose appropriately. This alignment ensures that, for example, when OA explains a Bible verse, it leverages the theology LM as intended, and when embedding text for retrieval, it uses the dedicated embedding model – yielding better results and consistency with our design expectations.

- **System State Awareness:** The OA and PM agents now boot up with a **shared, authoritative view of system state**. Through the **handoff kernel envelope** and associated surfaces, they know exactly which phase we're in (Phase 24) and what the last completed work was [12]. They also know the repository branch name (so PM is aware we're in a feature branch for Phase 27 prep). Crucially, all the **governance and health hints** (from the control-plane DMS) are available to OA/PM, meaning the agents are informed about checks like kernel preflight, DMS-first documentation usage, etc., per project rules [28]. This addresses earlier issues where Cursor or OA might "forget" to enforce a rule –

now the **PM agent has those hints in its registry** and OA has tools to query them [40] [28]. The result is that the **agents' behavior is much more tightly coupled to the actual system's needs**. For instance, OA will refuse to proceed if the kernel mode is degraded [41], and Cursor's contract ensures it always loads the kernel and runs `make reality.green` before destructive actions [42] [43]. These behaviors are documented and, as our checkups show, the system is in the correct state (NORMAL mode with all guards green) to exercise them.

- **Resolved Staleness:** Many of the "stale or corrupted assumptions" from earlier phases have been fixed. The knowledge base alignment was a big one – the Phase 23–24 boundary introduced some drift that is now corrected (we see DMS alignment = OK). The AGENTS.md content is freshly synced and even programmatically validated against the DMS contract [26], so OA is not operating on an outdated agent spec. Additionally, improvements like **phase-agnostic bootstrap generation** mean the PM_BOOTSTRAP_STATE is generated dynamically for the current phase rather than being pinned to an old phase [44]. This prevents the planning agent from using a misleading bootstrap snapshot. Indeed, the Bootstrap Consistency check confirms no mismatches in phase info [33]. Another example: the **share sync policy reform** eliminated the previous hardcoded whitelist of files – now everything is driven by the registry [45] [46]. This ensures no surprise deletions or missing files, which in the past could have confused OA if a needed doc vanished or an unknown file appeared. All these changes tighten the loop between **the system's reality and the assistant's understanding** of it.

- **Current Posture:** As of the latest data (Dec 8, 2025), the system is **running in a healthy, normal mode**. OA has likely explained to the orchestrator (human) that all systems are OK and is ready to assist with the next development tasks. The **Console/UI** would show green status for DB and LM, and list Phase 24 as current with no blockers. If we consider what OA's next steps proposal might be: since Phase 23 was completed and Phase 24 is in progress, OA (in conjunction with PM) should be suggesting tasks related to Phase 24's goals (SSOT alignment, share sync, etc. – some of which are already done, as indicated by on those sub-phases). It may even suggest moving to finalize Phase 24 or verify its completion, given that a lot of Phase 24's checks are now green. The planning agent's posture is proactive – it has no "firefighting" to do (no remediation needed), so it can focus on strategic work from the Master Plan. The **alignment of mental model and reality means OA/PM can operate confidently**: the decisions they make and advice they give are based on truthful, up-to-date information about the project.

In conclusion, the Orchestrator Assistant system has largely achieved the intended alignment between design and implementation. The agents (OA, PM, Cursor) are working with correct information about each other and the environment. The local models configured match the slots defined in documentation, the knowledge bases are synchronized with the code and data, and the runtime health is correctly reflected to the assistant. The minor inconsistencies that do exist (like the contradictory `reality_green` flag reading) are already identified by the system and can be corrected with a small fix. Going forward, ensuring those last small gaps are closed will give us a fully self-consistent OA. At this point, **the mental model (what OA believes about the system) is almost entirely in line with reality**, which sets a solid foundation for the next phases of the project.

Open Questions / Follow-Up Areas:

- **Reality Green Flag Discrepancy:** What is causing the `reality_green` status to be marked **false in PM_BOOTSTRAP_STATE** when all checks are passing? [10] This seems to be a minor bug – possibly

the bootstrap generator being overly strict (treating any hint like the repo drift as a fail). Clarifying and fixing this will ensure PM and OA absolutely agree on system mode. It's important to address so we don't get a false DEGRADED signal in any logic that trusts the bootstrap's `reality_green` field.

- **Missing Share Docs:** Two documents (`PM_HANDOFF_PROTOCOL.md` and `SHARE_FOLDER_ANALYSIS.md`) are noted as **missing in share/** (though the policy guard didn't fail) [31]. We should confirm their status: Are these to be created in an upcoming phase? If they are meant to exist now, adding them would clear the last "missing" indicator. If not, perhaps the registry entry is premature. Resolving this will tighten the share sync even further (ideally, no "missing_in_share" entries under strict mode).

- **OA Boot Sequence Automation:** The ORCHESTRATOR_REALITY doc and Phase 26 specs highlight that **OA should automatically load the kernel status on session start** and have tool access to do so [21] [47]. We need to verify if this has been fully implemented. Currently, OA has *methods* (`get_pm_boot_envelope()`, etc.) [40], but are they being invoked without human prompt at session init? If not, that integration is the next step – so OA truly never starts "cold." Ensuring OA's system prompt or bootstrap includes the kernel envelope (or that Cursor triggers it) would complete the vision of mandatory context. This is likely in progress (as the design is there), but we should track it to closure.

- **Inference Provider Configuration:** Given that LM Studio is being used as the primary model host (all signs point to `INFERENCE_PROVIDER=lmstudio` in the env), do we still need Ollama running in parallel? The current setup works, but it might simplify things to standardize on one unless needed for fallback. If we intend to use both (e.g., Ollama for smaller models), we should test that pathway. Otherwise, we might update documentation or `.env` defaults to reflect that **LM Studio is the default provider for all local models** now. Consistency between what's documented (Granite via Ollama by default vs what's actually happening) will prevent any confusion in operations. In short, **align the env defaults with the actual usage** (it appears LM Studio is effectively the default in practice, which is fine – we just note it).

- **Knowledge Base Coverage:** The DMS metadata shows a large number of docs with *unknown* importance (783) [48]. While not immediately problematic, an open question is whether we should categorize more of those (some might deserve "high" or "medium" importance for better oversight). Also, as new phases complete, we should continuously run the **KB course-correction** (there's a Phase 9.x item about KB registry health) to catch any *KB_DOC_STALE* or *KB_DOC_OUT_OF_SYNC* hints [49]. Currently, no such hints were present (all agent docs are synced, etc.), but as the project evolves it's wise to keep an eye on that.

- **Phase/Branch Forward-Pointers:** We're working on Phase 27 features while Phase 24 is the official current phase. This is fine, but we should be diligent in updating the phase index and SSOT surfaces when Phase 24 is completed, so that Phase 25+ can be registered. The bootstrap generator already discovers phases from `docs/SSOT/PHASE*.md` [50]. An open point is to ensure any partially implemented phase features (like the AGENTS–DMS contract checks from Phase 27.l) don't confuse the plan execution. Thus far it's under control, but we will want to **advance the phase in the kernel** when appropriate so that OA/PM know to officially incorporate those Phase 27 changes. Essentially, coordinate the **mental model's phase progression** with the actual development branch progression.

By addressing these follow-ups, we'll further solidify the alignment between the OA's internal model and the ground truth of the system – keeping the assistant and orchestrator firmly on the same page as the project moves ahead.

Sources:

- Current agent/model configuration and slot assignments – **AGENTS.md** 1 2
- LM router rules and provider setup – **AGENTS.md** 7 5
- Handoff kernel status (phase, branch, health checks) – **HANDOFF_KERNEL.json** 51 11
- Reality summary checks (DMS alignment, agents sync, etc.) – **REALITY_GREEN_SUMMARY.json**
- Detected OA state mismatch in summary – **REALITY_GREEN_SUMMARY.md** (excerpt) 10
- System health snapshot (DB ready, LM ready on LM Studio) – **pm_snapshot.md** 14 8
- Hint about repo drift (non-blocking) – **REALITY_GREEN_SUMMARY.json**
- Knowledge base registry and contract checks – **REALITY_GREEN_SUMMARY.md** 26 31
- Orchestrator design/boot references – **ORCHESTRATOR_REALITY.md** 40 52

---

1 2 3 4 5 6 7 27 28 AGENTS.md
file://file-C7rZGVaKG8tThWSNCQes2x

8 13 14 15 pm_snapshot.md
file://file-5nwMGEyqgNaRabjh87iCrE

9 11 12 20 51 HANDOFF_KERNEL.json
file://file-M5UCUKKUaQWBu2pAZEYw9p

10 26 29 30 31 32 33 34 36 37 38 39 48 REALITY_GREEN_SUMMARY.md
file://file-U2Xufuc25sih8RtjKqpeGy

16 17 18 19 21 22 40 41 47 52 ORCHESTRATOR_REALITY.md
file://file-DkgvAm5MrPnyv9medQMPsN

23 24 25 35 44 45 46 50 PHASE24_INDEX.md
file://file-NecM595TkQFY15UU7psFWr

42 43 PHASE26_OPS_BOOT_SPEC.md
file://file-AmmdEtw1FzU2wgthrJyWQ8

49 pm_system_introspection_evidence.md
file://file-8ooaoGSu6EFwBbLjEBV88y