ChatGPT

Review of the Snowflake Intelligence Claims Analyst Demo and Learning Series

Overview of the Demo Repository and Structure

The **Snowflake Intelligence Claims Analyst** repository is a comprehensive demo solution that integrates **Snowflake Cortex Analyst**, **Cortex Search**, **AI SQL functions**, and **Snowflake Intelligence Agents** to tackle an insurance claims analysis scenario. It centralizes structured claims data and unstructured evidence (notes, guidelines, images, audio) in Snowflake, allowing natural language Q&A across both types of data. The repository is well-organized into functional directories:

- `sql/` – SQL setup scripts for creating databases, tables, stages, and initializing AI services.
- `models/` – Semantic model definitions (in YAML) mapping the database schema and business terms for Cortex Analyst.
- `docs/` – Documentation files (likely including architecture overviews, "How It Works" explanations, and use-case descriptions).
- `medium/claude/` – Draft Medium articles (tutorial write-ups intended to accompany the demo).

Additional folders (e.g. a `streamlit/` app) are present for an optional UI, but the core learning content resides in the above directories. The repository's README itself provides an excellent high-level summary of the agent's capabilities and setup steps. For instance, it lists all Snowflake tools leveraged – **Cortex Analyst, Cortex Search, AISQL, and Agents** – and even highlights multimodal analysis of images and audio transcription features [1] [2]. Overall, the project structure is logical, grouping related resources in a way that's easy to navigate for learners.

End-to-End Learning Showcase with Snowflake Cortex

This hands-on demo does a strong job showcasing an end-to-end Snowflake Intelligence solution. It walks the user through every phase: setting up database objects, ingesting unstructured files, configuring AI services, and deploying an agent. The demo scenario is realistic – an "AI Claims Agent" that can answer complex questions about claim payments, reserves, guideline compliance, and even perform fraud detection tasks by analyzing notes and images. By covering **both** the structured data querying via Cortex Analyst *and* unstructured document search via Cortex Search, the demo illustrates how these components work in tandem. The agent is orchestrated to decide which tool to use for a given question, enabling a complete answer (for example, using semantic SQL for numeric analysis and vector search for textual justifications). This full-stack approach (structured + unstructured + multimodal) effectively demonstrates Snowflake's unique value proposition of bringing all data into one platform for AI-driven analysis.

Notably, the repository's content reflects many advanced AI capabilities of Snowflake Cortex. For example: the SQL scripts configure **AI document parsing** (using `CORTEX.PARSE_DOCUMENT` to OCR PDFs of claim notes and guidelines) and build **vector search indexes** on the text chunks [3] [4]. The semantic model YAML in `models/` defines dimensions, facts, and synonyms for all relevant tables (Claims, Claim Lines, Financials, etc.), which teaches the Cortex Analyst how to understand domain terms. The agent specification goes even further – it includes custom instructions and multi-step sample questions to guide the AI's reasoning (e.g. instructing the agent how to decide if a claim is "complete" by checking data presence) [5] [6]. It also registers multiple tools: one text-to-SQL tool tied to the semantic model and two search tools for guidelines and notes [7] [8]. This means the demo doesn't stop at a trivial use-case; it delves into orchestrating **hybrid querying**. Such depth is excellent for learning because it exposes users to real-world complexities – for instance, how to join insights from a PDF guideline with SQL results. In summary, the demo achieves an end-to-end scope: from data ingestion all the way to an AI agent answering questions, touching on each critical component of Snowflake Intelligence in a coherent workflow.

Documentation Clarity and Code Quality

The documentation is **clear, structured, and effective** in guiding users through setup and understanding the solution. The README (and likely extended docs in the `docs/` folder) starts with a succinct overview and key features list, then provides step-by-step **Setup Instructions**. These steps are easy to follow: create database objects (via `setup.sql`), upload the provided evidence files to a Snowflake stage, then configure Cortex AI components (via `setup_cortex_ai.sql`) [9] [10]. Each step is explained in plain language, and important object names (like the database `INSURANCE_CLAIMS_DEMO` and stage `LOSS_EVIDENCE`) are highlighted in code style for clarity. It's evident the authors took care to make the instructions newbie-friendly – for example, listing out all tables that will be created and what they contain [11]. The README even suggests **sample questions** the agent can answer (12 example queries about payments, reserves, compliance, etc.) which both prove the solution's capabilities and give users ideas to try [12] [13]. This is a nice touch to drive engagement after setup.

Moreover, the documentation touches on **"Who Should Care?"** – mapping the solution's value to different roles (Adjusters, Managers, SIU, IT, Compliance, etc.) [14]. This table of target audience vs. value proposition shows excellent organization; it not only teaches *how* to use the demo but *why* it matters, reinforcing learning with context. The `docs/` folder likely contains further narrative (perhaps an architecture deep-dive similar to Snowflake's developer guide). If it includes sections like "How It Works" or diagrams of the agent workflow, those would bolster understanding. The Snowflake Developer Blog content for this use-case (included in `docs/` or referenced) was very detailed about the architecture and business impact – integrating that ensures the documentation is not just a "how-to" but also a conceptual learning resource.

The **code quality** is generally high. The SQL scripts are well-commented and logically ordered (create roles, then databases, then tables, etc.) [15] [16]. They even handle permissions and role setup for least privilege (creating a specific role for the demo agent and granting only needed access) [17] [18]. This is excellent practice and educative for users unfamiliar with Snowflake's security model. The semantic model YAML is thorough – it defines columns with descriptions and synonyms, which exemplifies best practices for semantic modeling (each field like `CLAIM_STATUS` or `LOSS_STATE` has multiple synonyms listed, teaching how to make the model robust to different phrasings [19] [20]). Having this in a `models/` file is great for clarity, since users can read the YAML and learn how a well-designed semantic model is structured (dimensions vs. facts, sample values, primary keys, etc.).

The Python Streamlit app (if included) is also clearly written and commented. Right at the top there's a docstring explaining it's an "App for Insurance Claim Audits" and what it does [21]. Throughout the code, functions are documented (e.g. `get_claim_details` explains it fetches comprehensive details and even constructs predefined audit questions for the UI [22]). For a learner, seeing these inline comments and docstrings is very helpful to understand how the application ties into Snowflake (like using Snowpark to query tables, and calling the Cortex Analyst API via `_snowflake.send_snow_api_request`). The code is segmented into logical parts (Snowflake session init, data retrieval functions, AI chat logic, UI layout), improving readability.

One small suggestion: ensure that all key code sections in the SQL or Python have accompanying explanations either in docs or code comments. For instance, the agent creation JSON spec is quite dense [5] [23]. A beginner might not immediately grasp everything the JSON is doing. The Medium articles or docs could walk through this spec, explaining each part (tools, instructions, etc.), which I suspect is planned. Overall, however, the combination of clear docs and clean code makes the repository approachable. Even complex aspects (like the custom document classification function in SQL, or the AI_SIMILARITY image comparison) are documented in the code comments or commit messages. This clarity will help users **learn by example**.

Learning Value for Newcomers vs. Experienced Users

This demo provides significant learning value to both **Snowflake Cortex beginners** and more **experienced users**, though their perspectives will differ:

- **For newcomers to Snowflake Intelligence/Cortex:** The project offers a guided, hands-on introduction to many features. The stepwise setup and provided sample data mean a newbie can get something working without deep prior knowledge. They will learn fundamental concepts by doing – e.g. what a *semantic model YAML* looks like, how to *upload data to a stage*, how to *call an AI function* in SQL, etc. The documentation avoids assuming too much background; it defines the scenario in plain terms and even includes the business context. Also, the use of a concrete scenario (insurance claims) grounds abstract concepts in a story, which is pedagogically effective. That said, the breadth of the demo (covering structured, unstructured, multimodal *all at once*) could be overwhelming for some new users. There are many moving parts, so a true beginner might not fully understand each component on first pass. However, since the Medium article series is intended to break down these concepts, a newcomer who reads the articles *and* runs the demo will likely find the experience rewarding. They'll gradually connect "this is how I create a table" to "this is how the agent answers questions from that table."

- **For experienced Snowflake or AI practitioners:** They will appreciate the depth and realistic complexity. Experienced users often look for non-trivial examples that demonstrate best practices – this demo delivers on that. It goes beyond a "hello world" agent; it deals with multi-table joins, uses AI functions for OCR, sets up vector indexes, and even integrates everything into an application interface. An experienced user will find the repository well-organized, so they can jump to the part of interest (e.g., a Snowflake developer might go straight to `loss_claims.yaml` to inspect the semantic model design, or to the agent spec to see how multi-tool orchestration is defined). The comments and structure are concise enough that experienced folks won't be bored by hand-holding, yet they can still glean new insights (for instance, seeing how to use `FLATTEN(EXTRACT_ANSWER(...))` to auto-associate documents with claims [24] [25] might be a clever trick even a seasoned user hadn't tried). Additionally, experienced users may use this demo as a template or reference for their own projects, so the clear separation of concerns (data prep vs. AI config vs. app) is beneficial.

One area to consider is **onboarding prerequisites** for novices. The docs could explicitly mention prerequisites like "You need access to a Snowflake account (with the ability to create databases and the Snowflake Intelligence feature enabled) and the AccountAdmin role initially." The `USE ROLE ACCOUNTADMIN` at the top of setup.sql [26] hints at this requirement. New users might not realize they need to be an admin to create the SNOWFLAKE_INTELLIGENCE object. Making this clear up front in documentation would prevent confusion. Experienced users would likely infer this, but beginners might not. Minor guidance like this can improve accessibility without diluting the rich content.

Evaluation of Draft Medium Articles (`medium/claude/` Series Drafts)

The draft Medium articles in the `medium/claude/` folder seem intended to form a **learning series that parallels the demo.** Although we don't have the full text here, the prompts suggest the series is broken into multiple parts or topics (given the reference to a *"series structure"* and possibly a hub-and-spoke approach). Based on typical structure and the hints from the repo, the series likely includes: an introductory article about Snowflake Intelligence and the claims scenario, followed by deep-dives into semantic modeling, unstructured search, agent orchestration, etc.

From what we can infer, the **current series structure** might be linear (Part 1, Part 2, Part 3, …) covering the following:

- *Introduction to Snowflake Intelligence with an Insurance Use-Case* – Why AI-driven claims analysis matters, overview of Cortex Analyst and Cortex Search in plain language, and what the demo will achieve. This sets the stage.
- *Building the Semantic Layer (Cortex Analyst)* – Explaining semantic views, how the YAML model is constructed for the Claims data (perhaps walking through an example of defining a metric or dimension). Possibly referencing the "Designing Semantic Models" best practices.
- *Integrating Unstructured Data (Cortex Search & AI Functions)* – Discussing how claim notes and guidelines are parsed and indexed. Likely teaching what vector search is and how Snowflake's AI functions parse documents. Might include code snippets from the `setup_cortex_ai.sql` (like the `PARSE_DOCUMENT` and `SPLIT_TEXT` steps).
- *Orchestrating the AI Agent* – Showing how the Agent is configured to use both the Analyst and Search tools together. Possibly covering the agent JSON spec and custom instructions, and how queries flow through the agent.
- (Possibly) *Building the Streamlit Dashboard or Advanced Topics* – If included, this could be a bonus part about using the agent in a UI, or tips on extending the solution (like adding your own data or other tools).

Clarity and effectiveness: If the drafts mirror the quality of the repository docs, they are likely informative but may need a stronger narrative thread. For instance, do the articles simply walk through what was done (which can risk reading like documentation), or do they teach concepts with this demo as the example? The best Medium learning articles usually balance **concept explanation** with **practical example.** It will be important that each article doesn't feel like "Step 1 do this, Step 2 do that" in a vacuum – instead, it should explain *why* each step matters. Given the user mentions possibly using a "Claude" (AI assistant) for drafting, the content might be dense. I would check that the tone is consistent and engaging (sometimes AI-generated text can be formal or repetitive, so a human edit for flow and excitement is key).

One positive sign: the repository and README provided a wealth of material to draw on – such as the business impact and use-case scenarios – which, if woven into the Medium articles, will enhance their appeal. For example, a Medium draft that begins with a compelling problem statement ("Claims adjusters juggle spreadsheets and PDF reports, causing delays…") and then introduces Snowflake's AI solution will hook readers better than a dry technical start. The drafts should leverage those real-world hooks (like faster decisions, fraud detection, compliance benefits) that are mentioned in the repo's docs.

Without the exact text it's hard to critique specifics, but a potential issue could be **scope overlap**: If each article tries to cover too much, readers might get lost. Ensuring each piece has a clear focus (e.g., one article purely on semantic modeling best practices with this demo as an example) will make them more effective. Also, the series should introduce terms gradually – e.g., the first article should define "Cortex Analyst" and "Cortex Search" in simple terms, so that later articles can use those terms freely. From the Medium snippet by Lily Loyer, we see an example of introducing semantic models in context: *"In part 1 of this series, we explored Cortex Analyst… the quality of your queries is only as good as the semantic model behind them."* [27] . Emulating this style – where each part builds on the previous – will increase clarity.

Comparing Series Structure to Successful Medium Learning Series

There are already a few Medium series and articles on Snowflake Intelligence (and similar data/AI platform how-tos) that we can compare against:

- **Lily Loyer's Cortex Analyst series** (2025): As noted, Lily wrote a multi-part series where Part 1 introduced Cortex Analyst conceptually, Part 2 focused on designing semantic models, etc. This is a classic approach of progressively deepening the topic. Notably, Lily's articles are concise (~5 min read each) and narrowly focused. For instance, in her semantic model article she clearly defines what a semantic model is and why it matters before diving into YAML specifics [27] [28] . She also references prior parts, creating a linear flow. Readers praised this because each piece felt digestible and practical.

- **Justin Grimme's "Demystifying Snowflake Cortex AI"**: Justin took a single long-form approach, walking through building an agent in ~10 minutes read. He started with a strong TL;DR that succinctly explained Snowflake Intelligence as *"an orchestration layer that brings powerful tools like Cortex Analyst, Cortex Search and Custom Tools together… data never leaves your platform… smart enough not to hallucinate."* [29] . This kind of upfront summary of key points is very effective at orienting readers. Justin's article then used a narrative style, showing an example question and how the agent reasons step by step (with screenshots of the agent's chain-of-thought). It's engaging and less tutorial-like, but still educates on what the platform does. The strength here is in **storytelling** – it reads almost like a story of an AI agent solving a problem, with explanatory commentary.

- **Rahul Sahay's "Snowflake Intelligence solved X in 3 days"**: Rahul's Medium post took a business-first angle, highlighting ROI and time-savings (400+ hours saved, etc.), then outlined the technical steps as a sequence of four tasks (data setup, semantic model, indexing, agent creation) [30] [31] . Each step was mapped to an estimate of time (e.g., "Semantic Model: 5 minutes") and key code snippets. This quasi-"tutorial with a stopwatch" format made it feel very achievable. For example, he shows the exact SQL to create the semantic model from a YAML in one command [32] and immediately states the payoff: *"Now 'show loss ratio' auto-generates perfect SQL."* [32] . This style is compelling to technical readers because it promises quick results and shows tangible outcomes.

In comparison, the **current series draft** might not yet exploit some of these techniques fully. It sounds like the series is structured as a typical tutorial (which is good), but to reach the level of the successful examples:

- **Introduce a unifying narrative or persona:** For instance, speaking from the perspective of a claims analyst who gets "superpowers" from this solution can make it relatable. Justin's piece personified Cortex as an assistant that "does this and that – it's addicting to watch!" [33] [34], which brought excitement. Our series could similarly personify the Claims AI Agent or use a running example query that evolves through the parts.

- **Use visuals and diagrams:** The repository likely has an architecture diagram (perhaps in `docs/`). Including that in the Medium articles (hub or spokes) will help readers visually parse the system. Many Medium readers love flow diagrams or before/after visuals. For example, Rahul's article references a flow diagram and an ERD from his repo [35] [36]. If our series includes a clear diagram of how data flows from user question -> Analyst -> Search -> answer (the "agent orchestration"), that would be valuable in the hub or orchestration-focused article.

- **Ensure each article has a clear focus but all tie together:** A hub-and-spoke model is ideal here. The **"hub" article** can serve as an overview (similar to Justin's TL;DR but expanded) – explaining what the full solution will achieve and summarizing each component at a high level. It should excite the reader about the possibilities (perhaps with a compelling title and an image of the working demo). Then it can say, for example: "In this series, we'll learn to build this step by step: first by creating a semantic model, then enabling document intelligence, and finally deploying an agent." This hub piece would link out to each "spoke" article for details. Each **"spoke" article** then dives into one segment (semantic model, or search, or agent config).

The advantage of hub-and-spoke is twofold: (1) Readers can choose the depth they need – an executive could read just the overview for concept, a developer could click into the specific how-to articles. (2) It boosts SEO/visibility on Medium – one central article can funnel readers to the others, and vice versa (through "Part 1, Part 2" links or a series list). Successful series like those by Lily or the Snowflake Blog often have an intro that aggregates the parts.

In our case, the **current series can be improved** by adopting this model. Right now, if the drafts are just sequential posts titled "Part 1, Part 2…", consider renaming/refocusing them as standalone topics under a common theme. For example:

- Hub: *"Building an AI Claims Analyst with Snowflake – Overview"* – containing the why, what, and links to parts.
- Spoke 1: *"Setting Up Your Data and Semantic Model (Snowflake Cortex Analyst Guide)"* – hands-on with semantic model creation, including why semantic layer is important (with perhaps a nod to reliable text-to-SQL as Lily mentions).
- Spoke 2: *"Enriching the Agent with Unstructured Data (Cortex Search Guide)"* – covering document parsing, chunking, search indexing, and how that answers those "why" questions from claim notes/guidelines. Possibly discuss embeddings briefly but focus on usage.
- Spoke 3: *"Orchestrating an Insurance AI Agent in Snowflake Intelligence"* – putting it together: how to configure the agent JSON, use both tools, handle multimodal queries. Could show an example query flowing through.

- (Optional Spoke 4: *"Interactive Analysis: Building a Streamlit Claims Audit App"*) – if the audience is interested in front-end integration. This could be more for experienced users as a bonus.

Each should reference the repository and encourage running the demo for practice. Comparatively, *other series* often include code snippets and then link to a GitHub for the full project – we already have the full project, so we should leverage it. For instance, Rahul's Medium article explicitly pointed to SQL scripts in his repo and a wiki guide [31] [37]. We can do similarly: within our Medium tutorials, include snippets from our repo (with explanation) and say "see the full file in the GitHub repo for more context." This makes the Medium articles a funnel to the repo, and the repo a sandbox for readers – a very effective learning loop.

The Demo as a Hands-On Complement to the Articles

The **integration of the demo with the Medium articles is crucial** and, in my view, will be very effective provided they are properly aligned. Hands-on practice greatly reinforces written learning. The current demo gives readers something tangible to do: after reading about semantic models, they can open Snowflake and actually create one using the provided YAML; after reading about vector search, they can run the script to parse and index documents and then **try asking questions** in Snowflake Intelligence UI. This active component will transform a passive Medium reader into an engaged learner.

To maximize this complementarity, a few recommendations:

- **Explicit Callouts in Articles to "Try it Yourself":** For example, in the semantic model article, after explaining a concept, say "Now, open the `models/loss_claims.yaml` from the repo (or your fork) – notice how we define dimensions like CLAIM_STATUS with synonyms. Go ahead and upload this model in your Snowflake instance as shown." By guiding readers to actually use the repo content at each stage, you ensure they aren't just reading but doing. Each part of the series could have a "Hands-On Exercise" section referring to the repo scripts (e.g., "Run the `sql/setup.sql` script up to the point of creating tables, and verify in Snowflake that tables were created"). The README already enumerates steps, so the articles can simply reference those steps at the appropriate times.

- **Synchronization of Medium content with repository updates:** If any adjustments are made to the code or scripts, reflect that in the articles. For instance, if the Medium text says "upload files to a stage named X", ensure the repo uses the same name X. Consistency prevents user confusion. The good news is that the repository readme and Snowflake guide text are very consistent (the Medium draft likely borrowed from them), so we just need to keep them in lockstep.

- **Addressing common pitfalls in the write-up:** When users follow along hands-on, they might hit snags (e.g., forgetting to grant a role or enabling the Intelligence preview). The articles can proactively mention these. A tiny "Troubleshooting" callout or footnote could save a learner frustration – for example: "*Note:* Ensure the Snowflake Intelligence feature is enabled for your account (it's GA as of 2025). If `CREATE SNOWFLAKE INTELLIGENCE` fails, you may need to have your account admin enable it." Such tips increase the effectiveness of the demo as a complement, because they smooth the path for the user to actually see results.

In essence, the demo is a rich playground, and the Medium articles are the guided tour of that playground. Used together, a reader can both **understand the concepts** and **immediately apply them**. This greatly enhances educational impact. Many successful learning series (in tech blogs or Medium) use this formula:

explain a concept, then tell the user to go run something or observe something. Our content should do the same. Given the polish of the demo, I anticipate readers will get a "wow" moment when they run the agent and ask, say, *"Was a payment made according to state guidelines?"* and actually get an answer with citations. That kind of hands-on reward is memorable and will cement their learning.

One more note: the Medium articles should clearly link to the GitHub repo (and vice-versa the repo README can link out to the articles once published). This cross-reference ensures users are aware of both resources. For example, at the end of the Medium posts, include: "**Next Steps:** Try the full demo yourself by visiting the GitHub repository [12] and following the instructions." The README already has a line "Navigate to Snowflake Intelligence and start asking questions!" [38] – adding "(you can read the accompanying tutorial series on Medium for detailed guidance)" would close the loop nicely.

Recommendations and Improvements

1. Refine the Medium series structure with a Hub & Spoke approach: Create a short **hub article** that serves as the entry point, summarizing the problem (claims processing pain points) and the solution (Snowflake-powered AI agent) in an engaging way. Use this article to provide an outline of the series and links to each part. This will help readers grasp the big picture before diving in. Each subsequent "spoke" article should focus on a specific stage (semantic modeling, document search, agent orchestration, etc.), as discussed. Ensure the first few lines of each article remind the reader how it fits into the overall journey ("This is part 2 of the series on building an AI Claims Analyst…"). This structure improves clarity and allows readers to choose content based on their interest or level [27].

2. Emulate engaging storytelling and emphasis from successful articles: Incorporate a **TL;DR or key takeaway box** at the start of the hub article (and possibly each part). For example: "**TL;DR:** Snowflake Intelligence lets us build a claims analysis agent that can answer questions by combining SQL queries on structured data and semantic search on documents – all inside Snowflake [29]. In this series, we'll show you how to do it step by step, using a real insurance scenario." This primes the reader with excitement. Similarly, don't shy away from highlighting Snowflake's unique advantages (no hallucination, data governance, etc.) just as other authors did in one-liners [39]. Those points help experienced readers see the value proposition immediately.

3. Integrate comparisons or best-practice callouts in the narrative: Since you're aware of other approaches (like Rahul's quick-build method [40]), you can include little asides in the Medium text like, "Note: We're taking a slightly more detailed approach here than some quickstart guides – for example, you could create a semantic model in one command using a YAML file [32], but we'll walk through defining it to understand the components." Such commentary shows the reader you're knowledgeable about alternatives and are intentionally teaching the longer-form way for educational benefit. It also validates the methods used in the demo against known patterns.

4. Ensure consistency and completeness in documentation: As mentioned, clarify prerequisites in the docs (both README and Medium). For instance, mention that the demo uses Snowflake's **Enterprise Edition (or higher)** features (if Intelligence requires it) and an appropriate role. Small details like the Snowflake Region (if using an internal function like ARCTIC embedding model, region could matter) might be worth noting if relevant. In the code, double-check for any typos or minor bugs (the README had "claim photo eveidens" – a tiny typo for "evidence" [41]). Fixing such typos in documentation will enhance professionalism. Also, verify that all file names and paths in instructions match the repo (e.g., if the repo moved `files/loss_claims.yaml` to `models/`, update instructions accordingly).

5. Expand the educational commentary in the semantic model YAML (if not already in docs): The `loss_claims.yaml` is quite large, but adding a brief top comment or a section in docs explaining its layout (like "we define 5 tables in the model: here's how we link Claim Lines to Claims, etc.") would help learners. Even an annotated excerpt in Medium could be useful to illustrate how a metric or dimension is defined. For example, highlight a snippet defining a metric or a primary key and explain why those are important (e.g., *"Setting a primary_key in the semantic model helps Analyst avoid improper join duplications"* – tidbits like that add educational value beyond just following steps).

6. Leverage images and possibly short clips: If possible, include a screenshot of the Snowflake Intelligence UI showing the agent in action (the README references a screenshot [42]). A picture of an example Q&A result or the Streamlit app's interface can motivate readers. It makes the outcome more concrete ("here's what you'll get by the end"). If Medium supports it and you have time, even a quick gif or video of asking a question and getting an answer would be gold. It demonstrates the "wow factor" of the live agent.

7. Balance depth with not overwhelming the newbie: While the demo is comprehensive, in the articles be careful to introduce complexity gradually. Perhaps in the first article, focus on a **simpler question** the agent can answer with just structured data (e.g., "What's the average payment delay?") – basically Cortex Analyst alone. Then in a later article show a question that needs unstructured data ("Was the reserve rationale documented in notes?") requiring Cortex Search. This progression shows how each added component expands the agent's ability. It's a pedagogical approach to layer features. It prevents a new reader from being dropped in the deep end all at once. The demo itself has everything at once, but the writing can sequentially build it up.

8. Solicit and incorporate feedback: Once one or two articles are drafted, consider sharing them (even internally or with a small audience) along with the repo to get fresh eyes feedback. They might point out if any steps were unclear or if they got stuck following along. Use that to refine the documentation. Since this is an educational project, that feedback loop can significantly boost its effectiveness.

By implementing these improvements, the learning experience will be more structured, engaging, and accessible. The combination of a clearly narrated Medium series and a robust hands-on repository will cater to multiple learning styles. Readers will not only **understand *what* to do**, but also **why they are doing it** and **how each piece works**, which is the ultimate goal of an educational demo.

In conclusion, the hands-on demo is already a powerful teaching tool that showcases Snowflake Cortex's capabilities in a realistic scenario. With some reorganization of the accompanying Medium articles into a hub-and-spoke format and a few enhancements in presentation, this package will effectively guide both newcomers and seasoned users through the process of building an AI-driven analyst on Snowflake. It hits the mark on end-to-end learning; now it's about polishing the delivery so learners come away informed and inspired to apply these concepts in their own domain. Each part of the solution – from structured semantic modeling to unstructured search integration – provides valuable insights, and together they demonstrate how Snowflake Intelligence can transform a complex business process. By comparing the series with proven examples and adopting their best elements (clear structure, narrative engagement, practical focus), we can ensure that this demo and tutorial series not only matches but possibly sets a new bar for effective learning content in the Snowflake community.

Sources:

- Snowflake-Labs Insurance Claims Agent README (demo overview and features) [1] [10]
- Snowflake-Labs setup.sql and setup_cortex_ai.sql (schema, AI configuration, and agent spec from the repository) [43] [5]
- Lily Loyer – *Designing Semantic Models for Cortex Analyst* (Medium, part of series) [27]
- Justin Grimme – *Demystifying Snowflake Cortex AI* (Medium article, overview and TL;DR) [29]
- Rahul Sahay – *Snowflake Intelligence Just Solved Claims Chaos…* (Medium article with stepwise build approach) [40] [32]

---

[1] [2] [9] [10] [11] [12] [13] [14] [38] [41] [42] README.md
https://github.com/Snowflake-Labs/sfguide-insurance-claims-agent/blob/c48b6b862c2580f4b542e248548d12a64603c4a6/README.md

[3] [4] [5] [6] [7] [8] [23] [24] [25] setup_cortex_ai.sql
https://github.com/Snowflake-Labs/sfguide-insurance-claims-agent/blob/c48b6b862c2580f4b542e248548d12a64603c4a6/scripts/setup_cortex_ai.sql

[15] [16] [17] [18] [26] [43] setup.sql
https://github.com/Snowflake-Labs/sfguide-insurance-claims-agent/blob/c48b6b862c2580f4b542e248548d12a64603c4a6/scripts/setup.sql

[19] [20] loss_claims.yaml
https://github.com/Snowflake-Labs/sfguide-insurance-claims-agent/blob/c48b6b862c2580f4b542e248548d12a64603c4a6/files/loss_claims.yaml

[21] [22] app.py
https://github.com/Snowflake-Labs/sfguide-insurance-claims-agent/blob/c48b6b862c2580f4b542e248548d12a64603c4a6/streamlit/app.py

[27] [28] Designing Semantic Models for Snowflake Cortex Analyst: A Guide to Reliable Text-to-SQL | by Lily Loyer | Medium
https://medium.com/@lily.loyer/designing-semantic-models-for-snowflake-cortex-analyst-a-guide-to-reliable-text-to-sql-bc951bf5f801

[29] [33] [34] [39] Demystifying Snowflake Cortex AI: A Practical Guide to Building with Snowflake Intelligence | by Justin Grimme | Medium
https://medium.com/@grimmej/demystifying-snowflake-cortex-ai-a-practical-guide-to-building-with-snowflake-intelligence-aef542e9e727

[30] [31] [32] [35] [36] [37] [40] Snowflake Intelligence Just Solved Claims Chaos in 3 Days | by Rahul Sahay | Dec, 2025 | Medium
https://medium.com/@rahulsahay123/snowflake-intelligence-just-solved-claims-chaos-in-3-days-21417192c0d3