Training Objectives Inference for Advisor and Prompt-Architect Agents

Executive Summary

This report analyzes a dataset of 3,248 learning-focused conversational fragments, an accompanying methodology report, and a reference framework of learning objectives, to infer training and learning objectives for two agent roles: **(1) an Advisor-type conversational agent** (helps users clarify goals and structure task briefs) and **(2) a high-technical prompt-architect meta-agent** (designs, validates, and refines prompt architectures and system-level reasoning). The provided files each contribute unique insights: the fragment dataset with weights reveals cognitive patterns in advisor dialogues, the methodology report explains how these fragments were extracted and classified, and the reference JSON lists target **learning objectives** for advanced agent behavior.

Key Findings and Recommendations:

- **Data Insights:** The corpus of 3,248 fragments captures prevalent reasoning patterns – *reflective analysis, procedural planning, descriptive statements,* and minimal content – which form the basis of advisor cognition modeling. Each fragment is tagged with a weight (0.1–0.99) and rationale denoting its cognitive category (Reflective, Procedural, Descriptive, Minimal). This enriched data helps identify what skills the Advisor agent is exercising and hints at objectives the agent should learn (e.g. reflecting on constraints, planning next steps). It also surfaces references to **system-level principles** (like *"prompts as production assets requiring lifecycle management"*) that align with the meta-agent's domain.
- **Current Model Critique:** The existing rule-based weighting model classifies fragments by key "learning verbs" (e.g. *reflect, plan, verify*) into four cognitive tiers. While straightforward and transparent, this approach may **over-simplify** conversational nuance. We find it heavily skewed towards "Minimal" content (over 75% of fragments fell in the lowest tier), indicating the heuristic might be too strict or not capturing implicit reasoning. This raises concerns about missed signals – e.g. important principle statements without explicit keywords were down-weighted as *minimal*. A critique is that the weighting model, though consistent, might under-represent descriptive or contextual learning content, and conflates *presence of certain verbs* with cognitive importance.
- **Alternative Inference Approaches:** A broader landscape of methods can supplement or replace the current model. We discuss options including **semantic pattern mining, cognitive abstraction frameworks,** and **unsupervised clustering** to derive training objectives from conversation logs. These range from refined NLP pattern matching (to catch learning-oriented phrases beyond a fixed verb list) to **embedding-based clustering** of fragments (to discover emergent categories of advisor behavior). The goal is to capture nuanced learning behaviors (e.g. iterative questioning, summarization) that a simple keyword heuristic might miss. Each approach offers trade-offs in interpretability and coverage.

- **Interpretation Rubrics:** We propose multiple contrasting rubrics for interpreting fragments and inferring objectives, beyond the current four-tier weight model. One formulaic system extends the *rule-based categories* with multi-dimensional scoring (e.g. separate scores for reflection vs. planning content in a single utterance). Another rubric leverages a **taxonomic framework** (inspired by cognitive levels like Bloom's taxonomy), mapping each fragment to levels such as *Knowledge (recall), Application (procedure), Analysis (reflection),* etc. A third data-driven rubric uses **clustered themes** from the fragments (e.g. "clarify-goal inquiry", "plan-step outline", "verify-checkpoint") as categories. We examine these systems' pros and cons – for example, rule-based methods are transparent but brittle, whereas clustering reveals patterns organically but may produce less intuitive categories. The report details how each system would function and what biases or uncertainties come with each.
- **Agent-Specific Objectives:** By analyzing each fragment type, we infer distinct learning objectives for the Advisor vs. the Prompt-Architect agent. For instance, *reflective* fragments (e.g. "I need to understand the constraints before deciding.") suggest that an Advisor's training should emphasize **introspective reasoning** about user needs and assumptions, whereas a meta-agent's training would focus on **self-evaluative reasoning** about prompt designs and constraints. *Procedural* fragments (e.g. "Let's plan the next evaluation cycle.") indicate the Advisor should learn to guide users through structured planning, while the prompt-architect agent should learn to design and sequence complex workflows or experiments. We highlight such differences for all fragment categories (reflective, procedural, descriptive, minimal), showing how each type maps to role-specific skills. This comparative analysis ensures that training objectives are **tailored to each agent's function**: the Advisor's objectives center on conversational guidance and clarity, whereas the meta-agent's objectives center on technical prompt engineering and system oversight.
- **Alignment with Reference Framework:** We cross-reference the inferred objectives with the provided *learning.json* framework of meta-architect objectives. There is substantial alignment: many principles extracted in the conversations directly match the reference list (for example, the fragment *"Prompts must be treated as production assets requiring lifecycle management…"* appears in the logs and corresponds to a listed objective). This confirms that the conversation data is yielding relevant high-level goals. However, we also identify gaps – notably, the reference objectives file emphasizes system design principles (e.g. *defense-in-depth validation, multi-model orchestration*) and says little about the **conversational process skills** the Advisor needs (like clarifying user intent or summarizing trade-offs). Our analysis points out these missing pieces and suggests extensions: for example, adding objectives around *"clarifying ambiguous requirements"* or *"iteratively validating understanding with the user"* to cover Advisor-centric competencies. This synthesis uncovers opportunities to enrich the training blueprint so that it covers both the **technical meta-objectives** and the **interactive reasoning objectives** evident in the advisor logs.

- **Data Gaps & Signal Enhancement:** We note that certain metadata and signal layers are absent in the current dataset, limiting the fidelity of training-signal extraction. For instance, speaker roles in some logs are ambiguous (`unknown` entries) and conversation context isn't explicitly marked (no labels for question vs. answer, or which phase of the task the fragment comes from). There is also no explicit measure of outcome success or user feedback tied to fragments, which could otherwise help weigh which advisor behaviors are truly effective. To improve future analysis, richer annotations should be captured: e.g. tags for **dialogue act** (question, instruction, explanation), markers for **conversation stage** (onboarding, brainstorming, consolidation), and links to any **outcome metrics** (was the user satisfied? was the objective achieved?). Such metadata would provide stronger signals – allowing more precise filtering of what constitutes a "learning moment" and enabling supervised models to learn from success cases. We recommend an expansion of the logging and extraction process to include these layers, thereby improving the quality of inferred training objectives.
- **Proposed System Architecture:** Finally, we propose a reusable **system architecture** for ongoing inference of training objectives from future advisor and prompt-architect logs. The design is a pipeline that ingests raw dialogue data and produces updated training objective insights in a continuous loop. Key components include: automated fragment extraction (with an improved semantic parser), a classification and clustering module to categorize cognitive strategies, a mapping engine to align or expand the objectives taxonomy (like matching emerging patterns to known objectives or flagging new ones), and a knowledge base to store and version the evolving objectives. The architecture supports human-in-the-loop review for quality and integrates feedback to refine its inference rules over time. This ensures that as the agents interact in new scenarios, the system can **dynamically learn** and update what skills or knowledge the agents should be trained on. Overall, this approach moves the training objective inference from a one-off analysis to a **sustainable, scalable process** embedded in the deployment of these AI agents.

In summary, by combining data-driven analysis of advisor conversations with a structured reference framework, we derive a comprehensive set of training objectives for both the Advisor and the Prompt-Architect agents. We identify improvements to the current modeling approach and lay out methods and architecture for continually refining these objectives as more conversational data becomes available. The findings aim to guide the development of more effective meta-agents by ensuring their training targets the nuanced skills actually needed in practice, from high-level prompt engineering principles down to the dialogical techniques of clarifying and reasoning with users.

File Interpretation

This section describes the contribution of each provided file and how it informs the inference of training and learning objectives:

- `advisor_learning_weights.csv` – ***Fragment Dataset with Cognitive Weights*:** This CSV contains 3,248 entries of *learning-first conversational fragments* extracted from advisor dialogues. Each fragment is a short text snippet (often a single sentence or clause) that was identified as involving a learning or reasoning process (e.g. a statement of understanding, a plan, a check or reflection). Alongside each fragment, the file provides:
  - **Context identifiers:** an `id` (traceable to the source conversation turn), speaker role (`user` or the advisor agent, sometimes named "Kimi"), turn number, and source session (Advisor_1, Advisor_2, etc.), which help situate the fragment in the dialogue flow.
  - **Learning fragment text:** the content of the snippet, focusing on cognitively relevant language (for example, *"Understand the user's goals…"* or *"I will first validate my understanding…"*).
  - **Weight (0.10–0.99):** a numeric score representing the fragment's **relevance to advisor reasoning**, as determined by a rule-based model (higher means more cognitively rich).
  - **Rationale:** a short label or explanation of why that weight/category was assigned (e.g. "Reflective reasoning showing awareness or evaluation" or "Minimal learning content detected.").

  This file is the **primary data source** for discovering what mental processes the advisor engages in and at what frequency/intensity. By examining which fragments were tagged as reflective, procedural, etc., we can infer what kinds of cognitive skills or objectives the Advisor agent is practicing. For example, a high concentration of *reflective* fragments (weight ~0.8–0.99) in the data would indicate the advisor often engages in self-analysis or evaluation, which should be an explicit training objective for that agent. In contrast, fragments labeled *minimal* suggest areas where either the content is trivial (and maybe training should minimize such filler) or the current detection missed deeper meaning. Overall, `advisor_learning_weights.csv` provides the raw evidence of **advisor cognitive behavior**, which we will map into training objectives for both the Advisor and the meta-agent.
- `Advisor Learning Reports.pdf` – ***Methodology and Weighting Model Report*:** This PDF report documents the data processing and rationale behind the fragment dataset. It guides our understanding of how the fragments were extracted and classified:
  - It describes the **extraction methodology**: conversations were parsed for sentences containing certain *"learning verbs"* (like *define, clarify, design, reflect, verify, ensure, check, evaluate, plan, learn*, etc.). Sentences with these keywords were split and filtered to isolate meaningful *learning-oriented* statements. This clarifies that the dataset was intentionally biased toward segments where the advisor or user is discussing understanding, planning, or verifying – key moments for learning. Knowing this, we interpret the fragment dataset not as random dialogue, but as deliberately the *cognitive high points* of conversations.

- It details the **weight assignment model**: a rule-based heuristic that assigns each fragment a weight according to four cognitive categories. Specifically, *Reflective reasoning* fragments (deep insights, analysis) were given weights 0.80–0.99, *Procedural cognition* (planning and execution steps) 0.55–0.79, *Descriptive observation* (neutral statements or info sharing) 0.30–0.54, and *Minimal content* 0.10–0.29. Example verbs for each category are listed (e.g. *"reflect, analyze"* for reflective; *"plan, design"* for procedural) and example scored fragments are provided (ID 1001: *"I need to understand the constraints before deciding."* → 0.93, reflective). This helps us interpret the CSV: for instance, if a fragment was weighted 0.67, the rationale likely labeled it procedural (planning-related). Understanding these categories is crucial for inferring training objectives: each category corresponds to a type of cognitive skill the agent is exhibiting. The report also notes that a rationale sentence was auto-generated for each to explain the classification.
- It includes a brief **distribution analysis** (expected metrics) suggesting that in an ideal scenario, about 40% of fragments would be reflective, 35% procedural, 25% lower (descriptive/minimal). This expectation underscores that the training focus is on the higher-order reasoning content. In reality, our analysis found the actual output skewed more heavily to low-weight fragments, which the report indicates might still be under refinement (the weighting was a planned step). The methodology report's transparency about the process allows us to critically evaluate the dataset and consider improvements (e.g. potential underestimation of descriptive fragments, given only ~2% were classified as descriptive).
- It outlines *next steps* and a planned **analytical roadmap**, including clustering the fragments by semantic similarity and reviewing outliers. This hints at alternative methods (embeddings, thematic clustering) which we take into account in our method landscape. It also emphasizes principles like traceability and extensibility – meaning the system was designed to be built upon. We leverage these insights to propose our extended inference methods and system architecture.

In summary, the PDF provides context that transforms the raw CSV data into a meaningful story about advisor cognition. It helps us interpret what the fragment weights signify, warns us of the model's heuristic nature (so we remain cautious about biases), and inspires ideas for further analysis (like using clustering or embedding techniques beyond the initial approach).

- `learning.json` – ***Reference Learning Objectives Framework*:** This JSON file contains a curated list of high-level **learning objectives** aligned with a "master blueprint" for the meta-architecture (likely the system that these agents operate within). Each entry is a concise principle or practice for advanced AI system design, with references to sources (L1, L2, etc.) indicating their provenance. Examples of objectives listed include:
  - *"Prompts function as production assets requiring lifecycle management and performance observability..."*
  - *"Defense-in-depth validation architecture separates user content from system instructions"*
  - *"Retrieval-Augmented Generation (RAG) grounds outputs by retrieving relevant external knowledge"*

- *"Continuous performance monitoring tracks semantic drift and failure patterns"*, etc.

These objectives appear to be derived from previous learning (noted by sources L1–L4) and cover a range of *system-level and procedural best practices* for complex AI orchestration. They are especially pertinent to the **Prompt-Architect meta-agent**, who would be responsible for implementing and respecting such principles in prompt design and multi-agent workflows. This file serves as a **benchmark or target** for our inference: after analyzing the conversation fragments, we want to see how the emergent objectives (what the advisor is focusing on, what needs the meta-agent has) align with or diverge from this list. For example, if an objective in `learning.json` is *"Parallel exploration of alternatives enables comparison of methods"*, we can check if the advisor's conversation ever touched on encouraging multiple solution paths, or if the meta-agent's design would require that capability. We indeed find many direct overlaps, as the user's conversation with the advisor often explicitly enumerated these objectives (often referencing them as L1, L2, etc., indicating they were consolidating such principles during the dialogue). Thus, the JSON provides a *framework to validate against*: it ensures our inferred training objectives for the two agents include established ideas (no major principle is overlooked) and helps highlight any **gaps** – e.g. if the advisor logs reveal an important skill like "clarifying requirements" which is not in `learning.json`, that gap suggests an extension to the framework. In essence, `learning.json` ties our analysis to the broader blueprint of what the AI agents should ultimately learn, grounding our specific findings in the context of the system's goals.

By interpreting each file in this way, we set the stage for analysis: the CSV gives us empirical data on advisor behavior, the PDF gives us methodology and a lens for evaluation, and the JSON gives us the target learning outcomes especially for the meta-agent. Together, they enable a comprehensive inference of training objectives, as we explore next.

Method Landscape

Inferring training objectives from conversational data can be approached through multiple analytical lenses. The current project employed a **rule-based semantic filtering and weighting** approach. Here, we broaden the view and outline a landscape of alternative inference methods, each leveraging different techniques to extract meaning from conversation logs. These methods are not mutually exclusive – they can complement each other to build a richer understanding. Key approaches include:

- **1. Enhanced Semantic Pattern Mining:** This approach generalizes the current rule-based method by using more advanced NLP to detect learning-related content. Instead of relying solely on a fixed list of "learning verbs", we could use semantic parsing or dependency analysis to find statements where the agent is making an inference, drawing a conclusion, or setting a goal. For example, patterns like *"I need to [X] before [Y]"* or *"This will help [achieve Z]"* might indicate a learning objective or strategy being discussed even if specific verbs differ. Techniques like **keyword expansion** (using synonyms or word embeddings to catch variants of "clarify", "ensure", etc.) and **named entity or role recognition** (to detect when the agent references its own role or the user's goal) can improve recall of relevant fragments. The benefit of pattern mining is **transparency** and alignment with human intuition (we explicitly know what we're looking for), but the trade-off is that it may still miss less obvious learning moments or novel phrasing. It requires iterative tuning of patterns to improve coverage.
- **2. Clustering and Embedding-Based Inference:** Unsupervised learning techniques can reveal structure in the conversational fragments without predefined categories. By converting each fragment into a vector (using language model embeddings), we can apply **thematic clustering** to group similar fragments together (as hinted in the analytical roadmap). For example, clusters might naturally form around *"asking clarifying questions"*, *"laying out step-by-step plans"*, *"discussing system constraints"*, *"summarizing past results"*, etc. Once clusters are formed, we interpret each cluster to infer the underlying training objective. If one cluster consists of fragments like *"Let's break this task down…"*, *"First, we should do X then Y"*, that clearly points to an objective around **procedural planning skill**. Another cluster might feature self-checks (e.g. *"I want to double-check that I understand…"*) indicating a **self-validation or reflection** objective. Clustering is powerful in that it can surface **emergent patterns** that were not explicitly looked for. It also helps handle the nuance of multi-faceted utterances (some fragments might belong to two themes). The trade-off is interpretability: clusters need to be manually labeled or characterized, which can be subjective. Also, very context-dependent phrases might cluster oddly if vector representations aren't distinct. Nonetheless, this data-driven method ensures we don't narrowly focus only on what we expect – it may discover, say, a cluster of "meta" comments (agent talking about its own limitations) that could highlight a training need (e.g. *calibrating confidence or deferring when unsure*).

- **3. Abstraction via Cognitive Frameworks:** This method involves mapping fragments to an existing theoretical framework of cognitive or conversational skills. For example, using **Bloom's Taxonomy**, we could classify each statement into levels such as *Remember, Understand, Apply, Analyze, Evaluate, Create*. An advisor saying *"I need to understand the constraints…"* is at the Analysis/Evaluation level (identifying what needs evaluation), whereas *"Let's plan the next steps"* is Application (applying knowledge to a plan), and a simple factual statement would be Remember/Understand. Similarly, one could use a framework like **dialogue acts or conversational functions**: classify whether a fragment is a question, an instruction, a confirmation, a reflection, a suggestion, etc. Each category corresponds to a skill (e.g., *asking clarification questions* is a skill; *giving suggestions with rationale* is another). By abstracting fragments into such categories, we can formulate training objectives more systematically: e.g., "The Advisor should be able to Evaluate and Analyze user needs (Bloom level V)" or "The agent should master the dialogue act of Clarification Queries". The benefit of a framework-based approach is that it draws on established educational or linguistic theory, potentially covering the full range of cognitive processes. It ensures **completeness** (we check each level or category for representation in the data). The downside is that real conversational data may not fit neatly into one taxonomy; some fragments can span multiple categories, and some categories might not appear at all. There is also a risk of being too high-level – the mapping might gloss over domain-specific nuance (e.g., "Evaluate" could mean evaluating a design vs. evaluating understanding – very different in practice).
- **4. Supervised Learning for Intent Classification:** If we had a labeled dataset of what each fragment's "intent" or learning significance is, we could train a classifier (potentially using an ML model or fine-tuned language model) to automatically categorize new fragments. For instance, human annotators (or domain experts) could label a subset of fragments as "Goal clarification", "Solution brainstorming", "Verifying assumption", "User instruction interpretation", etc. A model could learn these and then predict labels for new fragments from logs. This is essentially treating **training objective inference as a classification problem** at the utterance level. The advantage is automation and potentially high accuracy in known categories – once trained, the model could quickly sift through large volumes of logs and flag instances of each training-relevant category. It might also capture subtle linguistic cues that rule-based methods miss. However, it requires an initial investment in labeling data and the performance is only as good as the consistency of those labels. It might struggle with edge cases or shifts in language usage over time unless continuously retrained. Moreover, the categories must be wisely chosen to correlate with meaningful training objectives (there's a risk of conflating surface-level intent with deeper learning goals).

- **5. Temporal or Conversational Sequence Analysis:** Another angle is to look at **when and how these fragments occur in the dialogue sequence**. By analyzing conversation transcripts in sequence (not just isolated fragments), we can infer objectives related to process and timing. For instance, if reflective fragments often appear after the user provides new information, it suggests the agent has a pattern of summarizing or evaluating input – a training objective could be "Agent should summarize user inputs to confirm understanding". If procedural planning fragments appear mostly at the conversation outset or conclusion, it indicates how the agent frames tasks – leading to objectives like "Agent should be able to outline a plan at the start of a task and recap next steps at the end." Methods to do this include **sequence pattern mining** (finding common n-grams of dialogue acts) or **Markov/state modeling** of conversation phases. While our current data is segmented and sorted by turn, a more sequential analysis could reveal *causal relationships* – e.g., a verification question from the agent leads to a user clarification, which leads to the agent adjusting a plan. Such patterns highlight training needs in **interactive responsiveness** and **adaptability**. The challenge here is needing the full dialogue context and aligning fragments with their context, which our current fragment list only partially provides. We might need to reconstruct or refer back to the full logs (the original JSONL) to perform this analysis. If done, it provides a dynamic view of training objectives (not just "what skill" but "when to use it").
- **6. Knowledge Graphs and Concept Extraction:** Particularly for the meta-agent's domain, we can extract key concepts and relations discussed in the logs (e.g. *lifecycle management, validation architecture, semantic drift, failure memory*). Building a simple knowledge graph of these concepts and seeing how the conversation connects them can inform objectives. If the advisor spends effort linking concept A to concept B (e.g. linking *"verification burden"* to *"system complexity"* as a principle), it suggests an objective like "The meta-agent should learn the relationship between verification effort and system complexity" – essentially domain knowledge that the agent should internalize. Concept frequency and co-occurrence can signal what the important domains of knowledge are for training. This is somewhat orthogonal to the cognitive process view, but it's complementary: it ensures that training objectives cover not only *how* the agent should think, but *what content* it should know well. Techniques for this include entity extraction and topic modeling. The drawback is that not every concept mentioned needs to become a training objective; we must filter to those relevant to agent performance. Also, focusing on concept-level could miss the behavioral aspect of objectives.

In practice, a robust inference of training objectives would **combine these approaches**. For example, one might start with pattern mining to capture obvious cases, use clustering to find hidden themes, apply a framework to ensure completeness, and then validate with sequence context. The methodology report itself hints at multi-step analysis (weight distribution, then clustering, then outlier analysis), which aligns with mixing quantitative and qualitative methods. By employing this broad toolkit, we can derive a comprehensive set of objectives that reflect both the **qualitative insights** of human analysis and the **quantitative patterns** present in data.

Formula / System Proposals

Building on the above methods, we now propose several concrete **rubric systems** for interpreting fragments and inferring training objectives. Each system provides a formulaic way to score or categorize fragments, offering distinct perspectives. We contrast these systems and discuss their trade-offs:

1. **Rule-Based Cognitive Scoring System (Extended):** This is an evolution of the current weighting model. It retains the intuitive four categories – *Reflective, Procedural, Descriptive, Minimal* – but refines the rules for assigning scores:

   - We broaden the keyword set and incorporate **contextual cues**. For example, beyond explicit verbs, patterns like question marks (?) could indicate a *clarifying question* (reflective intent), imperative tone could indicate *instruction or planning*, etc.
   - Introduce a **multi-facet score**: instead of one weight, each fragment gets a tuple like (ReflectionScore, PlanningScore, InfoScore). For instance, "I will first validate my understanding…" might score (0.9 Reflection, 0.5 Planning, 0.2 Info) indicating it's mostly reflective with some procedural aspect. The system could use simple rules: presence of analysis verbs adds to ReflectionScore, presence of sequence words (first, next) adds to PlanningScore, presence of factual statements adds to InfoScore, etc.
   - A final composite weight could still be computed, or the facets themselves become categories for objectives (e.g. a high ReflectionScore fragment contributes to reflective training objective).

   Trade-offs: This system is transparent and traceable – each score is based on known rules. It's easy to adjust (add a new rule if a certain important pattern is under-rated). It continues to allow quick scanning of data by category. However, it might become complex to maintain as rules multiply, and there's risk of **overlap** (a fragment triggering multiple rules). There's also the issue of calibrating multiple scores meaningfully. Still, it ensures no single metric oversimplifies a fragment and can highlight multi-dimensional utterances (some statements truly do two things at once, e.g. reflect *and* plan). This extended rubric would better capture nuanced advisor behaviors while staying interpretable.

2. **Data-Driven Clustering Rubric:** In this system, we dispense with predetermined categories and let clusters define the rubric. We might derive, say, **5–10 major clusters** from the fragment embeddings and then label them as categories. An example outcome could be categories like: *"Clarification Questions"*, *"Planning Steps"*, *"Stating Principles/Knowledge"*, *"Meta-Reasoning (agent talks about its process)"*, *"Miscellaneous/minimal"*. Each new fragment can be assigned to the nearest cluster (perhaps giving a probability or distance measure as a score of fit). The rubric is essentially the set of cluster centroids plus a descriptive label for each.

   Trade-offs: The strength here is that categories emerge from actual usage patterns, potentially matching the reality of conversations better than arbitrary ranges. It might reveal a category the designers hadn't considered. For example, a cluster might correspond to *"Agent expressing uncertainty or limitations"* if such fragments exist – which could prompt a new training objective about handling uncertainty. The scoring (distance to centroid or cluster membership probability) can indicate how typical a fragment is for that category, which could help find outliers or new sub-categories. On the downside, the categories might not align perfectly with conceptual training needs; they may need interpretation. Also, clusters can change with more data or different parameter choices, making the rubric less stable or intuitive to humans. Essentially, it's *harder to explain* to a stakeholder why a fragment got a certain label ("the model clustered it this way" is less satisfying than "it had a 'plan' verb"). So this system sacrifices some interpretability for adaptiveness and discovery. It would likely be used in combination with human oversight – e.g., use clustering to propose rubric categories which are then reviewed and adjusted by experts before solidifying the rubric.

3. **Cognitive Taxonomy Rubric (Theory-Driven):** Here we apply a formal educational or cognitive framework as the rubric. For illustration, we can use **Bloom's Taxonomy levels** (Knowledge, Comprehension, Application, Analysis, Synthesis, Evaluation) as categories. Each fragment is categorized into one (or more) of these levels:

   - *Knowledge/Remember:* recalling facts – e.g. a fragment that just states a piece of info or system detail (likely a *descriptive observation* in the original model).
   - *Comprehension:* demonstrating understanding – e.g. rephrasing the user's request or summarizing (the advisor often does this to confirm understanding).
   - *Application:* using knowledge in a new situation – e.g. planning a step-by-step approach using known principles.
   - *Analysis:* breaking down information into parts, seeing relationships – e.g. comparing alternatives, identifying a gap or constraint.
   - *Synthesis:* combining elements to form a new whole – e.g. the agent formulating a new approach or hypothesis from various inputs (perhaps seen in later-stage dialogues).
   - *Evaluation:* judging and critiquing – e.g. the agent evaluating the viability of a plan or reflecting on what is good/bad about an approach.

We would devise guidelines or use an LLM-based classifier to assign each fragment to one of these (some fragments may hit multiple levels; we could choose the highest applicable level or allow multi-label). The result is a rubric where each category corresponds to a broad cognitive ability that should be trained. For example, if very few fragments hit *Synthesis*, perhaps the advisor isn't doing much synthesis and we might prioritize training that skill. If many are *Analysis* and *Evaluation*, that confirms the agent often engages at high cognitive levels, which is a good sign for an advisor agent.

Trade-offs: The taxonomy-based rubric ensures we cover a spectrum of cognitive complexity. It can be appealing for designing curricula (each level suggests different training content or methods). It is also easier to communicate ("We want the agent to achieve more *evaluation-level* thinking in conversations."). However, mapping dialogue to Bloom's (or any generic framework) can be subjective and sometimes forced. Conversational utterances often perform multiple functions simultaneously, which a linear taxonomy doesn't capture well. Also, Bloom's was designed for learning objectives of human learners in educational settings; using it for AI dialogue might be metaphorical. Some categories like *Knowledge* vs *Comprehension* may be hard to distinguish from just one line of dialogue. Therefore, while this rubric gives a high-level overview of cognitive objectives, it might gloss over domain-specific or role-specific nuances (e.g. *Application* could mean very different things for an Advisor vs a Prompt Architect). We might need to tailor the framework (or choose a different one like SOLO taxonomy or a conversational acts taxonomy) to better fit our domain.

4. **Multi-Agent Role-Based Rubric:** Given we have two distinct agent roles of interest, another rubric system can revolve around *who the fragment is relevant to*. We could label fragments (or the skills they imply) as *Advisor-centric, Meta-agent-centric,* or *Both*. For instance, when the fragment is about understanding user goals, that's Advisor-centric; if it's about designing a prompt sequence, that's Meta-agent centric; if it's a general principle (like "test alternatives in parallel"), it might apply to both. This rubric isn't so much about scoring fragments as it is about sorting the resultant objectives. It becomes a matrix: cognitive process X agent role. Each fragment would be placed in the matrix, like "reflective + advisor" or "procedural + meta-agent". We could then formulate objectives per cell.

   Trade-offs: This rubric ensures we explicitly consider each agent's perspective and don't assume an objective is one-size-fits-all. It's very relevant to this task since our goal is to produce objectives for two roles. It leverages the content we have (some fragments are clearly advisor conversations, others describe the architect's duties) to separate concerns. On the downside, doing this categorization for every fragment might be overkill – many fragments are only spoken by the advisor anyway (so inherently advisor-centric). And for user-spoken fragments that reference the system, interpretation is needed to decide which agent the content informs. This system is less about discovering new categories and more about organizing by role, so it should be used in conjunction with another content-based rubric (like one of the above). We will effectively be doing a form of this in the *Subject-Centered Objective Inference* section by discussing each fragment type for each agent, which illustrates this rubric in practice.

Each proposed system brings a different lens:

- The **Extended Rule-Based** rubric is essentially an improved version of the status quo, emphasizing interpretability and ease of implementation.
- The **Clustering rubric** embraces a bottom-up, data-first view that can catch unknown unknowns but may need interpretation for use in training design.
- The **Taxonomy rubric** imposes a top-down structure that aligns with educational theory, ensuring completeness and clarity of objectives but possibly lacking specificity.
- The **Role-based rubric** ensures alignment of objectives with specific agent needs, which is crucial when multiple agent types are being trained concurrently.

In an ideal strategy, one might combine these: use clustering to find natural groupings, then perhaps map those clusters to a taxonomy or to the four cognitive categories (to understand their nature), and finally separate them by agent role relevance. The combination yields a robust rubric system: for example, we identify a cluster of "clarifying questions", recognize it as an Analysis/Evaluation level skill (taxonomy) largely used by the Advisor, and thus formulate a training objective "Advisor agent should be adept at formulating clarifying questions to evaluate user requirements." Meanwhile, another cluster "prompt design steps" might map to Application/Synthesis and be meta-agent relevant, yielding an objective for the prompt-architect agent.

The trade-off across all these systems is between **simplicity vs. richness**. A simpler rubric (like a tweaked version of the current one) is easy to apply but might overlook subtle training signals. A richer rubric (multi-dimensional or data-driven) captures nuance but can be harder to operationalize or communicate. For an effective training program, it's often worth embracing some complexity in analysis to ensure no major learning need is missed, then distilling the results into a clear set of objectives for implementation.

Subject-Centered Objective Inference

In this section, we translate the patterns observed in each fragment type (cognitive category) into concrete **learning objectives**, and we do so for each of the two agent roles. By examining *Reflective*, *Procedural*, *Descriptive*, and *Minimal* fragments, we infer what each agent (Advisor vs. Prompt-Architect meta-agent) should learn or improve, and highlight how these objectives differ given the agent's role and responsibilities. Each fragment type encapsulates certain cognitive behaviors:

Reflective Fragments

Reflective fragments are those where the speaker (often the advisor agent) is analyzing, evaluating, or showing awareness of their knowledge and approach. They often include self-referential insights or critical thinking about the task. For example, a reflective fragment from the data was: *"I need to understand the constraints before deciding."* – here the agent explicitly recognizes a knowledge gap and the need to fill it, demonstrating metacognitive awareness.

- **Advisor Agent Objectives (Reflective):** The Advisor should be trained to **engage in introspective reasoning and critical evaluation of the problem context.** From fragments like the above, a likely learning objective is: *Enable the advisor to identify what it doesn't know and formulate strategies to gain clarity.* This includes skills like:
  - **Clarifying Uncertainties:** The advisor frequently must reflect on what information is missing or unclear (e.g., *"I must clarify what 'maximally useful' means in this context"* as seen in the conversation). A training objective might be: *"The advisor will learn to pause and articulate assumptions or ambiguities in the user's request or the task brief, and prompt the user to resolve them."* This ensures the advisor doesn't proceed with false assumptions – a critical reflective practice.
  - **Validating Understanding:** Another observed behavior: *"I will first validate my understanding by asking targeted clarification questions..."*. The learning objective here: *"Train the advisor to routinely summarize its understanding of the system/task and ask the user for confirmation or clarification."* This reflective-check skill improves alignment with user intentions.
  - **Awareness of Role and Limits:** We saw fragments like *"I do not design workflows... I am a reflection partner"* which is the advisor reflecting on its own role boundaries. An objective could be: *"Advisor should internalize its role constraints and communicate them, reflecting on what it can or cannot do."* This is reflective self-awareness important for multi-agent coordination (knowing when to hand off to the Architect, for example).
  - **Critical Evaluation of Solutions:** If any reflective fragment shows the agent evaluating a potential approach (e.g. weighing pros/cons), the advisor should be trained in basic evaluative reasoning on user-proposed ideas (e.g. *"Is this approach feasible given the constraints?"*). The objective: *"The advisor can critique initial ideas or plans at a high level, identifying potential pitfalls or missing elements."*

  In summary, reflective fragments suggest the Advisor's learning objectives should focus on *meta-reasoning* capabilities: understanding when and how to reflect, double-check, and articulate thought processes to ensure clarity and correctness in problem framing.

- **Prompt-Architect Meta-Agent Objectives (Reflective):** The meta-agent (which designs and validates prompt architectures) also benefits from reflective skills, but in a different vein. Its reflection is about the *system and prompt strategy* rather than understanding user intent. Training objectives for the prompt-architect based on reflective patterns include:
  - **Self-Evaluation of Prompt Efficacy:** The meta-agent should be able to reflect on whether a prompt/workflow it designed is working as intended. For instance, analogously to the advisor saying "I need to understand constraints…", the meta-agent might think "I need to understand the model's limitations before finalizing this prompt." A learning objective: *"Train the prompt-architect agent to critically evaluate its prompt designs, checking if they meet the desired criteria (clarity, safety, goal alignment) and identifying where adjustments are needed."* This reflective check could manifest as the agent running a small test or analyzing the prompt's output quality (a meta-level evaluation).

- **Awareness of System Principles:** Many reflective fragments in the advisor logs pertain to system principles (since the user and advisor discuss them). The meta-agent should have ingrained *knowledge of and reflection on those principles*. For example, an objective from learning.json is *"Post-generation uncertainty calibration is essential"*. The meta-agent's reflective objective could be: *"The agent will internalize key architectural principles (e.g., uncertainty calibration, defense-in-depth) and reflect those in its reasoning; it will regularly cross-check whether a solution adheres to these principles."* Essentially, it needs to be a *self-aware architect* that doesn't just generate prompts blindly but cross-references best practices in real-time.
- **Failure Analysis:** If something goes wrong (prompt fails or output is poor), a meta-agent should reflect on why. This parallels an advisor reflecting on a misunderstanding. Training objective: *"Enable the meta-agent to diagnose failures or suboptimal results by reflecting on which part of the prompt sequence might have caused the issue (e.g., was context insufficient? was the instruction misunderstood by the model?), and learn from it."* In practice, this might involve the meta-agent simulating or analyzing prior prompts (perhaps using a technique akin to "critique loops" or having an internal critic model).
- **Continuous Improvement Mindset:** A more general reflective objective is fostering a habit of *continuous improvement*. The meta-agent should be trained to ask itself after each task: "How could this prompt or workflow be improved further?" just as a human engineer would. This could involve keeping track of lessons (akin to the "failure memory" concept in learning.json). An explicit training goal: *"The prompt-architect will log and recall previous mistakes or adjustments (a 'prompt memory'), reflecting on them to refine future prompt iterations."*

In essence, reflective fragments translate to the meta-agent needing strong *meta-cognitive and analytical skills about its own domain (prompts and system behavior)*. It should emulate a thoughtful engineer: always checking assumptions, reviewing outcomes, and aligning with overarching principles.

Procedural Fragments

Procedural fragments are action-oriented, focusing on planning, step-by-step methods, or execution of tasks. For example: *"Let's plan the next evaluation cycle."* or *"I will help the user articulate what the Meta Architect needs..."* (a plan to guide the user). These indicate goal-directed behavior sequences.

- **Advisor Agent Objectives (Procedural):** For the Advisor, procedural fragments indicate the agent's role in guiding process and structure during the conversation. Training objectives include:

- **Task Structuring:** The advisor often has to help break down a complex goal into manageable parts or outline a brief. An observed fragment: *"project stages… Provide structured, high-signal thinking… Transform messy user intention into clear inputs"* (though that one was oddly marked minimal due to being a list, it contains procedural intent). Objective: "*Train the advisor to co-create a structured plan or outline with the user for any given high-level task*." This means if a user goal is vague, the advisor can propose a step-by-step approach or a checklist of what needs to be done first, second, etc. It's essentially project scoping and planning in conversational form.
- **Guiding the User Through Workflow:** The advisor might lead the user through interactive steps. For instance, *"Let's go through this example"* or *"First, we'll clarify requirements, then we'll brainstorm solutions, finally we'll refine the best idea*." Even if not verbatim in our fragments, this style is implied. So an objective: *"Advisor should learn to guide the user through a logical sequence of steps (clarification -> ideation -> evaluation -> conclusion) depending on the task context*." This ensures consistency and thoroughness in how the advisor assists.
- **Ensuring Procedural Completeness:** One fragment shows "*This step ensures accuracy.*" – presumably the agent commenting on why a step is needed. That implies an objective: "*The advisor will learn to not only suggest steps but also explain the purpose of each step, ensuring the user understands why the process is structured that way.*" This builds user trust and clarity in the process.
- **Adapting Procedures to User Needs:** The advisor should also be procedural in a flexible way – adapting the plan if the user's situation changes. For example, if mid-conversation new info comes, the advisor might say *"Given this new information, let's adjust our plan*." A training goal could be: *"Ability to dynamically update the task structure or next steps in response to new context or user feedback*."
- **Time/Resource Awareness:** Though not explicitly in fragments, an advisor might need procedural awareness of constraints (time, tools available). For instance, *"We have X minutes, so I suggest doing Y first*." If any fragment indicated managing scope, that yields an objective around procedural prioritization.

- **Prompt-Architect Meta-Agent Objectives (Procedural):** For the meta-agent, procedural fragments inform objectives about designing and executing multi-step prompt workflows or validation processes:
  - **Workflow Design:** The meta-agent's core job is *designing prompt sequences and experiments*. A direct training objective is: *"Enable the meta-agent to plan multi-step prompting strategies to accomplish complex tasks*." For example, if solving a problem requires a chain of prompts (e.g., one to gather requirements, one to generate candidates, one to evaluate candidates), the agent should be trained to outline and implement that sequence. This is analogous to the advisor's task structuring but at a more technical level (structuring the interaction with the model(s) and tools).

- **Procedural Validation Loops:** Many principles in learning.json highlight validation, e.g. *"Defense-in-depth validation architecture"*, *"Multi-stage alignment loops"*, *"verification loops"*. The meta-agent should be trained to follow procedural **validation and verification steps** for any output. So an objective: *"The prompt-architect agent will incorporate multiple validation checkpoints in its workflows (such as intermediate verifications of output correctness, consistency checks, etc.), rather than producing a single-step answer."* Essentially, it learns to design a *procedure* around generation (like generate -> critique -> refine).
- **Tool Orchestration:** If the meta-agent has access to tools or multiple models (as hints of multi-model orchestration suggest), an objective is: *"Learn to procedurally orchestrate different models or modules (e.g., a generator and a critic, a retriever and a composer) in a structured manner to leverage their strengths."* For example, the agent might plan: call Model A to retrieve info, then Model B to draft an answer, then Model C to review the answer. This is a complex procedure the agent must execute reliably.
- **Reusability and Modularity:** A more abstract procedural skill is designing prompts in a modular way so they can be reused or updated (connected to *"Prompts as production assets"* principle). Training objective: *"Teach the meta-agent to create and maintain prompts as reusable modules, with version control and iterative improvement – effectively, treat prompt design itself as a procedural craft."* For example, it might maintain a library of tested prompt components and assemble them as needed (very much a procedure of reuse).
- **Efficiency and Optimization:** The meta-agent should also learn procedures for optimizing performance – e.g., *caching intermediate results*, doing parallel prompt explorations (as one principle suggests *"Parallel exploration of alternatives"*). So an objective could be: *"Train the agent in procedural optimization techniques, like caching, parallel branching, and token budget management, to ensure efficient prompt execution."* These are procedural in the sense of following a certain method to save resources.

In summary, procedural fragments emphasize **planning and execution** skills. The Advisor's objectives center on guiding the human through a clear process, while the meta-agent's focus on designing and following multi-step **prompt workflows** and validation processes. Both share the need to be systematic and methodical, but the content of their procedures differs (human-centric process vs. AI system-centric process).

Descriptive Fragments

Descriptive fragments involve stating facts, observations, or explaining something without explicitly driving action or reflection. They often provide context or share information. For example: *"Explains why failure memory exceeds success memory,"* (a fragment likely summarizing a concept), or *"UI affordances must match task types"* (stating a principle). These are more informational in nature.

- **Advisor Agent Objectives (Descriptive):** For the Advisor, descriptive moments in conversation usually mean the agent is **conveying information or summarizing knowledge** for the user's benefit. Training objectives derived from this could include:

- **Effective Summarization:** The advisor often has to summarize the current system, the blueprint, or prior results for the user. For instance, Kimi's output might include summarizing the system context ("My Understanding of the System…" which spans multiple fragments). An objective: *"Train the advisor to succinctly summarize relevant background information or prior discussions in a way the user can easily grasp."* This is a key skill especially if users come in with partial information or if the conversation is long and the advisor needs to recap.
- **Clarity in Explanations:** When explaining a concept (like what "failure memory" is or why a principle matters), the advisor should do so clearly and accurately. An objective: *"The advisor should be able to explain technical concepts or design principles (from the system's blueprint) to the user in clear, layman-understandable language when needed."* Essentially, the advisor acts as a translator between complex system details and the user's understanding. If the logs show the user was sometimes confused, the advisor's descriptive ability could be improved.
- **Highlighting Key Details:** If a fragment is descriptive, e.g. *"This module tests whether the system can guide prompt design"* (paraphrasing a fragment about a test), the advisor is providing an observation. A training objective can be: *"Teach the advisor to pick out and verbalize key details or observations about the current task state or output."* For example, if the user gets a draft output, the advisor might highlight a critical part: "Notice that the second section addresses X, which was a key requirement."
- **Contextual Knowledge Integration:** Some descriptive fragments might be the advisor bringing in external knowledge (if allowed) or reminding of something previously stated ("The blueprint I envision is a master design document…" is a user statement being descriptive). If the advisor can incorporate known info (like definitions of terms or relevant examples) at the right time, that's useful. So an objective: *"Enable the advisor to inject relevant contextual knowledge or definitions to assist the user's understanding (without derailing the conversation)."*

Overall, descriptive fragments for the Advisor highlight the role of the advisor as a **knowledge conduit** – it should have enough understanding and articulation to inform the user without necessarily solving the problem itself. Training should ensure the advisor can *communicate information effectively* and knows when to provide factual input.

- **Prompt-Architect Meta-Agent Objectives (Descriptive):** For the meta-agent, descriptive content often relates to *documenting or explaining the system's behavior or design decisions.* Potential objectives:
  - **State Reporting:** The meta-agent might need to report on the current state of a workflow or the outcome of a step (like logging). For instance, after running part of a prompt sequence, it might produce a description: "Step 1 completed, retrieved 5 relevant articles." A training objective: *"Teach the meta-agent to generate clear descriptions of intermediate results and system state changes as it executes a prompt architecture."* This keeps any human overseer or linked agents informed (and is analogous to how the advisor explains things to a user, but the meta-agent might explain to logs or to a monitoring system).

- **Documentation and Justification:** Many learning.json objectives read like documentation of design principles. The meta-agent should arguably be able to justify its own design choices in similar descriptive terms. Objective: *"The prompt-architect agent will be able to articulate the rationale behind its prompt designs or architectural decisions in descriptive terms (e.g. 'Because prompts are treated as production assets, I have versioned this prompt with comments...')."* This could help in audits or collaboration with human developers.
- **Knowledge Base Population:** If the system includes a knowledge base of verified outputs or principles (as suggested by *"Semantic knowledge bases store verified outputs"*), the meta-agent might contribute to it by writing descriptions of new learnings or results. Training objective: *"Enable the agent to summarize and record new insights or verified outcomes into a knowledge repository in a concise descriptive format."* Essentially, it needs to do the "paperwork" of learning – describing what was learned for future reference.
- **Interpreting User Objectives in System Terms:** When the advisor passes a well-structured input to the meta-agent, the meta-agent must interpret it (which might involve re-stating it in internal terms). A descriptive skill here is: *"The meta-agent can take a user-level goal description and restate it as an internal objective or set of specifications for the system."* This translation is partly procedural, partly descriptive (it's explaining the user's need in system language). The training objective: *"Prompt-architect is able to expand or clarify user requests into technical specifications (e.g. listing requirements or criteria the solution must meet)."* We glean this because often the advisor's output (System Initialization Brief or learning objectives) might be consumed by the meta-agent, who then has to understand them fully.

In short, descriptive capabilities for the meta-agent revolve around *communication and recording within the system's context*. The meta-agent may not have a human user to explain things to in real-time (if it's an autonomous back-end agent), but it still benefits from being able to describe its process and decisions for transparency, debugging, and knowledge retention.

Minimal Fragments

Minimal fragments were classified as having little explicit learning content (weights ~0.1–0.29). They might be fragments that are very short, generic, or context-setting without an obvious cognitive action. For example: *"Your job is to ..."* (instruction prefix), or *"related roles (Designer, Auditor, etc"* (partial phrase), or even just connectors. These often got labeled "Minimal learning content detected.". While individually not insightful, in aggregate they provide context or structure to the conversation.

- **Advisor Agent Objectives (Minimal content):** For the Advisor, minimal fragments might correspond to housekeeping utterances or fragments of instructions. The learning value is low per fragment, but patterns might still inform objectives:

- **Reduce Filler and Off-Task Content:** If the advisor sometimes produces chatty filler or irrelevant statements, an objective would be to minimize that. E.g., if an advisor response started with "Sure, I can help with that." (which might be considered minimal content if not followed by substance), training should aim for more substantive contributions. Objective: "*Train the advisor to maximize informational content in each turn, avoiding empty polite responses or repetitions that don't advance the task*." (Of course politeness is good, but from a learning perspective, we want high signal.)
- **Context Bridging:** Some minimal fragments might be things like segment titles or transitions (like "STEP 3 CROSS MODEL COVERAGE MAP" which looks like a header). While these are not learning-rich alone, the objective could be making sure the advisor can context switch or introduce new sections clearly. Perhaps: "*The advisor should be able to clearly indicate conversation structure or next section (e.g. 'Now, let's discuss X...') without overloading content*." Essentially, minimal content sometimes is just structural glue in conversation, which is fine, but it should be purposeful.
- **Accurate Role Acknowledgment:** If minimal fragments include things like the advisor stating role constraints ("You are a reflection partner, an input structuring expert" pieces, some of which were low-weight fragments), an objective might be to ensure the advisor always maintains its role. E.g., it should explicitly *not do tasks outside its scope*. Actually one fragment: *"You do not execute the tasks of the Architect... You are a reflection partner"* was marked reflective with 0.98 (likely because of the phrasing "You are a reflection partner"). But surrounding lines splitting that instruction might be minimal. The objective here is more like a rule: "*The advisor must consistently stay in its advisory role and refrain from performing actions reserved for other agents*." This is gleaned from the fact that the initialization prompt hammered those points (though we categorize it minimal, the instruction itself is critical).
- **Following Conversation Protocols:** Minimal content may include acknowledgments or short answers. The advisor should still handle those properly – e.g., saying "Yes, got it." when appropriate, or providing a short confirmation. Training might include **dialogue etiquette** objectives like giving confirmations or asking for pause when needed. These aren't "learning objectives" in the cognitive sense, but are part of training the conversational ability of the advisor (like social/pragmatic competence).

- **Prompt-Architect Meta-Agent Objectives (Minimal content):** For the meta-agent, minimal fragments from the logs might correspond to system messages, placeholders, or partial references. In the conversation, a lot of minimal fragments were actually the user's references to sources (like "[Source: L1, L2]") or half-sentences broken by segmentation. In terms of meta-agent training:

- **Handle System Instructions & Metadata:** The meta-agent should learn to handle pieces of input that are not full instructions but meta-instructions. For instance, if the advisor or system feed includes labels, the meta-agent must not get confused. A possible training objective: *"Teach the prompt-architect to recognize and ignore or appropriately process non-content tokens, placeholders, or system tags in its input."* This comes from seeing those bracketed [Source] references which a naive agent might try to interpret literally.
- **Integration of Minor Signals:** If minimal content fragments include small hints (like a user saying "e.g., ..." or an incomplete sentence), the meta-agent should not drop them if they matter. So an objective could be: *"Ensure the meta-agent can integrate minor context clues (like partial examples, or titles of steps) into its understanding of the task."* For example, if the advisor output had a structure with headings, the meta-agent should interpret the headings appropriately even if they carry no verb.
- **Efficient Parsing:** The meta-agent likely will parse inputs that might include some noise or extraneous content. Training objective: *"The meta-agent will robustly parse input prompts or briefs, filtering out non-essential verbiage while capturing essential details."* This essentially means not being thrown off by irrelevant or minimal bits.
- **Avoid Unnecessary Verbosity:** On the output side, the meta-agent should be trained not to generate excessive minimal content. For instance, if it's producing a prompt sequence, it shouldn't produce a lot of boilerplate text. It should be succinct and focused – something like an objective of *"Minimize superfluous content in designed prompts or system messages to keep them efficient."*

While *minimal fragments* by definition carry low direct learning content, analyzing them ensures that our training objectives don't ignore the "glue" that holds interactions together. For the Advisor, it's about maintaining role and structure without fluff; for the meta-agent, it's about dealing with or producing formal/structural content correctly. Both agents should learn to *reduce noise and focus on signal* in their respective contexts.

Synthesis of Fragment-Type Objectives

Combining the above for each fragment category, we can see a clear divergence in focus between the two agent types:

- The **Advisor's objectives** emphasize *interactive reasoning skills*: reflecting to understand the user and task, guiding the process, explaining concepts clearly, and structuring the conversation effectively. The advisor is the facilitator of human thought – so its training revolves around communication, clarification, and high-level planning in dialog.
- The **Prompt-Architect's objectives** emphasize *technical reasoning and design skills*: reflecting on prompt and system issues, planning multi-step prompt solutions, ensuring thorough validation and integration of principles, describing and documenting processes. The meta-agent is the orchestrator of AI workflows – so its training focuses on designing effective prompts, maintaining system integrity (through principles like those in learning.json), and self-improvement loops.

Even where both agents share a fragment type (e.g., both have to "reflect"), the **content and purpose** of that reflection differ. We've highlighted those differences in each category. This ensures that each agent's training curriculum can be specialized: the Advisor is trained in "people-facing" cognitive competencies, and the Prompt-Architect in "system-facing" cognitive competencies – a necessary distinction to fulfill their complementary roles.

Synthesis with learning.json

Now we compare the inferred objectives from our analysis with the reference learning objectives provided in `learning.json`. The goal is to identify where they **match** (i.e., our analysis independently arrived at similar objectives, confirming their importance), where there are **gaps** (objectives present in the logs/our inference but not explicitly in the reference, or vice versa), and potential **extensions** to the framework (new objectives to add, or ways to broaden existing ones).

Matches (Alignment with Existing Objectives):
It was heartening to find that many high-level principles discussed in the advisor conversations align directly with the `learning.json` objectives. For example:

- **Prompts as Lifecycle-Managed Assets:** Both the conversation fragments and `learning.json` stress that prompts are not one-off queries but enduring artifacts that require maintenance. We saw multiple fragments where the user (or advisor) notes *"Prompts must be treated as production assets requiring lifecycle management…"*, which corresponds exactly to a listed objective. Our training objective inference for the meta-agent included learning version control and iterative improvement of prompts, which echoes this point.
- **Defense-in-Depth Validation:** The idea that user content should be isolated from system instructions for security (and generally having layered validation) is explicitly in `learning.json` and was repeatedly referenced in the conversation (fragments about *"validation architecture separates user content from system instructions"* appear in the logs). We included training for the meta-agent to implement multi-layer validation steps, aligning with this.
- **Retrieval-Augmented Generation (RAG):** The reference lists RAG grounding outputs with external knowledge. In the conversation data, terms like "RAG grounding" showed up when enumerating principles. While our focus was more on the process, a likely training objective for the meta-agent (not explicitly stated earlier but implied) is to incorporate retrieval steps when needed. The presence of this in both the reference and logs suggests the meta-agent training *must* include knowledge of when/how to use retrieval.

- **Continuous Monitoring and Feedback Loops:** `learning.json` contains objectives like *"Continuous performance monitoring tracks semantic drift"* and *"Multi-stage alignment loops integrate human preferences and constitutional constraints"*. The advisor's conversation indeed involved multi-stage refinement of learning objectives (with user critiques and revisions – a human-in-loop alignment process). Our inferred objective for the meta-agent to have iterative improvement and failure analysis aligns with the idea of continuous monitoring and adjustment. Similarly, we talked about the advisor validating understanding continually, which is a micro-level feedback loop. So the concept of ongoing checks and refinements is well represented in both.
- **Parallel and Multi-Model Approaches:** The reference mentions *"Parallel exploration of alternatives"* and *"Multi-model orchestration"*. In the conversation logs, we saw references to parallel exploration, and mention of multiple roles and models (Kimi was aware of a family of roles, implying multiple agents/models). Our proposed training objectives for the meta-agent included orchestrating multiple models and possibly parallel prompting to compare results. This is a direct match with the blueprint's vision for a sophisticated meta-agent.
- **Result Reuse and Efficiency:** One objective in JSON is *"Result reuse optimizes cognitive workload by avoiding redundant verification..."*. In conversation, the user and advisor discussed caching and reusing outputs (e.g., the principle *"Multi level caching optimizes cost"* was mentioned). We incorporated the idea of caching and not redoing work in the meta-agent's procedural skills. This indicates alignment on efficiency-focused objectives.
- **Verification Burden & Manual Signals:** The reference emphasizes *"Verification burden is the primary constraint"* and *"Manual verification actions are stronger training signals"*. The logs contained instances of discussing verification steps and how to involve manual checks (the user was actively verifying principles). Our analysis did factor in the advisor guiding verification and the meta-agent building it into workflows. This alignment suggests our inferred objectives duly emphasize verification, consistent with the known framework.
- **Cognitive Overhead and Safety:** Finally, `learning.json` has items like *"Cognitive overhead minimization improves verification consistency"* and *"Constitutional rules require periodic re-derivation"*. The advisor's job in conversation was partially to reduce cognitive load on the user by structuring information (which is cognitive overhead management). We also saw the advisor mention the blueprint evolving and needing to update understanding (which touches on re-derivation of rules when the blueprint changes). So our advisor objectives around structuring and clarifying help with overhead, and meta-agent reflective objective about periodically re-checking core rules aligns with re-derivation. These subtle points show up in both, confirming their importance.

In summary, there is a strong correspondence between the high-level system design objectives in `learning.json` and the needs we identified for the prompt-architect agent. Many of the *technical and architectural* objectives we inferred (like layered validation, prompt versioning, orchestration, etc.) are explicitly in the reference list, which is reassuring – it means the training objectives we're formulating for the meta-agent are grounded in the intended system blueprint. Similarly, some of the advisor's behaviors (continuous clarification, summarizing to reduce confusion) indirectly support those same principles (ensuring alignment, minimizing misunderstanding which is akin to minimizing drift).

Gaps and Differences:
However, there are notable areas where our analysis (especially on the Advisor side) extends beyond or differs from the current learning.json objectives. These gaps highlight opportunities for expansion:

- **Advisor Conversational Skills vs. Technical Objectives:** The learning.json entries skew towards *system-level and technical considerations* (prompt engineering, validation, architecture). They do not explicitly cover the *conversational strategies* that an Advisor agent needs. For instance, nowhere in the JSON does it say "Clarify user requirements" or "Ask targeted questions to verify understanding" – yet these were clearly vital in the advisor's dialogues and hence in our inferred objectives. This is a gap: the framework might need additional objectives related to **interactive clarity and user guidance**. These could be framed as: *"Ensure thorough requirement clarification before solution design"* or *"Employ reflective listening to confirm understanding"* as general principles. Currently, the blueprint might assume an advisor does this, but it's not enumerated as a learning objective for the system. We'd recommend adding something along those lines to capture the training focus for advisors.
- **Human-Agent Interaction Objectives:** Similarly, the reference objectives don't mention user experience or how the agent should interact with humans (they are mostly about internal workings). Our analysis touched on things like the advisor explaining clearly, minimizing filler, etc. These could be formulated as objectives like *"Communication clarity and completeness"* or *"User guidance and education"*. In a holistic training blueprint, you'd want a section on conversational etiquette or effectiveness. This is currently a gap.

- **Meta-Agent Self-Management:** Another gap is that `learning.json` lists principles but doesn't explicitly say the meta-agent should learn them – it assumes these are guiding truths. We extrapolated to training objectives like the agent reflecting on those principles, documenting decisions, etc. The gap here is subtle: the blueprint lists what should be done (prompts should be versioned, etc.), but not necessarily *as learning objectives phrased for an agent*. We might reframe some of them as actionable objectives. For example, instead of just stating the principle, an objective could be *"The meta-agent consistently applies lifecycle management to prompt assets, including version control and performance tracking"*. Many objectives in JSON start with nouns ("Prompts function as…", "Retrieval-Augmented Generation grounds…"), whereas training objectives for an agent could be phrased as capabilities or behaviors ("Agent treats prompts as assets with lifecycle management"; "Agent can utilize RAG to ground answers"). Ensuring each principle is translated into an agent behavior might be an extension needed.
- **Missing Principles Not in Conversation:** Conversely, there might be objectives in learning.json that did *not* prominently feature in the observed conversation. For example, *"Four-layer architecture: constitutional shielding, heuristic routing, critique loops, semantic knowledge bases"* is one listed objective. In the advisor logs, pieces of that appear (they mention four-layer architecture vaguely in an "only mine" list), but the advisor wasn't directly dealing with implementing a four-layer architecture – that's more a design blueprint. Our training objectives for the meta-agent didn't explicitly mention "implement four-layer architecture" (that's more of a system design than a single agent skill). This could be a gap: if the meta-agent is responsible for such an architecture, maybe training should include *understanding the layers and interacting with them properly*. Perhaps the meta-agent needs to be aware of which layer it is operating in or coordinate with those layers (if they correspond to different subsystems or policies). This suggests an objective like *"The meta-agent will interface correctly with each layer of the system (ensuring constitutional rules are checked, routing is respected, critique feedback is incorporated, knowledge base is updated)"*. It's an area not explicitly covered in our earlier inference but worth noting if the meta-agent is in charge of that architecture.
- **Agent Wellness and Sustainability:** One intriguing objective in JSON is *"Sustainability requires interaction thresholds and rest periods"*. This implies concerns about the agent's operational health or avoiding overuse (maybe to prevent degraded performance or high cost). This did not come up in the conversation with the advisor (understandably, since that's more of a system ops consideration). Our analysis similarly didn't cover training the agent to throttle itself or take breaks. If the meta-agent runs continuously, it might need logic to avoid spamming an API or exhausting resources. That objective is a bit meta – possibly an operational guideline rather than something an AI "learns". However, to align, we could incorporate that as a training point: e.g. the meta-agent should be aware of not exceeding certain usage limits and perhaps escalate or pause if thresholds are hit. It's a niche point, but it's in the blueprint, so we highlight it as an area our inference hadn't touched (gap from our side).

- **Opportunity: Unifying Advisor and Meta-agent Objectives:** The synergy between what the advisor helps produce (the consolidated learning objectives L1–L4) and what the meta-agent consumes begs an objective around **inter-agent collaboration**. For example, *"Tagging reveals actual behavioral priorities"* might refer to analyzing conversation logs (tagging them) to see what was actually prioritized – something a training pipeline might do. The conversation logs themselves are a form of tagging (with weights). There might be an opportunity to explicitly include an objective about learning from the advisor's outputs. Perhaps: *"The meta-agent leverages advisor-generated insights (like identified ambiguities or consolidated objectives) to adjust its planning."* While not in JSON, this is an extension that ensures the chain from advisor to architect is closed. It's implied by having an advisor at all, but making it a training objective means the meta-agent is tested on using advisor input effectively.

Extensions and Additions:
Based on the above gaps, we can suggest extending the learning objectives framework:

- **Conversational Mastery Objectives:** Add objectives targeted at the Advisor role, such as *"Employ iterative querying to refine user goals"*, *"Provide structured summaries of complex information"*, *"Maintain role boundaries while maximizing helpful input"*, etc. These could complement the existing technical objectives by addressing the human-facing side.
- **Agent Collaboration Objectives:** Perhaps objectives about how different agents or roles interact. E.g., *"Downstream agents (Designer, Architect) effectively utilize Advisor's structured outputs"* and conversely *"Advisor anticipates the needs of downstream agents in its outputs"*. This two-way objective would formalize the relationship seen in logs where the advisor is explicitly trying to make outputs "maximally useful" for the Meta-Architect.
- **Refinement of Existing Phrasing:** Ensure each `learning.json` objective can be expressed as a capability for training. We partially did that in our inferred objectives. For instance, where JSON says *"Parallel exploration... enables comparison"*, we could add *"The meta-agent can spawn and manage parallel solution branches and compare outcomes to choose the best approach."* This makes it actionable.
- **Missing content from logs to objectives:** The advisor logs had some unique terms (e.g., "emoji routers", "tri verifier pattern" in the fragments). These might be highly specific and not in JSON. If they are considered important, they might become new objectives. For example, "tri verifier pattern" might refer to using three verifiers for something – that could turn into an objective about redundancy in verification. Without outside context it's hard to say, but the presence of jargon in logs that isn't in JSON implies either those were experimental ideas or not consolidated. If they are valuable, they'd be an extension to incorporate.
- **User-Centered Outcomes:** One perspective not in JSON is measuring success in terms of user satisfaction or task success. Training objectives could include *"maximize user's clarity and satisfaction by end of advisor interaction"* or *"meta-agent solutions meet defined user success criteria"*. These are more outcome metrics than design principles, but tying objectives to user outcomes can strengthen the training alignment with actual needs. This extension goes slightly beyond internal operation to external effect.

In conclusion, the learning.json provided a robust set of objectives, mainly oriented to the **meta-agent's technical domain**, and our analysis both confirms many of those and complements them with **advisor-oriented and interaction-oriented objectives**. The matches reassure us that the agent training will incorporate critical system design principles, while the gaps show where we should broaden the training curriculum to fully encompass the advisor's role and ensure the meta-agent doesn't just follow principles in isolation but also interacts well in the multi-agent ecosystem. By filling these gaps – adding conversational objectives and refining agent collaboration aspects – the framework of learning objectives can be made truly comprehensive, covering everything from low-level dialogue tactics to high-level architectural strategy.

Missing Data & Future Improvements

Analyzing the current dataset and methodology revealed several limitations in the available data and signals. Addressing these would significantly improve the extraction of meaningful training signals and the accuracy of objective inference. Here we identify the **missing metadata and signal layers** that, if added, would enhance future analyses:

- **Full Dialogue Context and Segmentation:** The fragments in `advisor_learning_weights.csv` are isolated sentences or clauses extracted from their dialogue context. While useful for pinpointing cognitive acts, this loses the surrounding conversation that gives each fragment meaning (who said what before, and what happened after). For deeper analysis (e.g., sequence patterns or cause-effect of questions and answers), we need the full turn-by-turn dialogue. An improvement is to retain references or metadata linking each fragment to the preceding and following utterances in the conversation. This could be as simple as including a conversation turn ID and an index of the fragment within that turn. With that, one could reconstruct threads or at least reason about whether a fragment was an opening question, a mid-conversation reflection, or a concluding summary. **Having context windows** around each fragment as part of the data would allow training models to understand when to use that behavior. For example, an objective "ask clarifying questions early" can only be learned if we know which questions occurred early vs late – currently, that timing isn't explicit in the fragment list.
- **Speaker Role Consistency and Identification:** We noticed entries labeled as `unknown` or inconsistent casing (`User` vs `user` , `Kimi` vs `assistant`) in the dataset. This indicates some role attribution issues. It's important that each fragment is clearly tagged with who is speaking and possibly their role type (human user, advisor agent, or other agent). In multi-agent setups, there might be multiple AIs (the advisor, the architect, etc.) – if their dialogues get logged, we need a way to distinguish fragments by agent. Consistently capturing the **agent identity** for each utterance is crucial. For future logs, a standardized schema (e.g., `role: advisor|architect|user|...`) would help filter relevant fragments per agent and avoid confusion. This also helps training signals: we wouldn't mistakenly train the advisor on something the architect said or vice versa.

- **Outcome Labels or Success Metrics:** The current data doesn't indicate which conversations (or which strategies within them) led to successful outcomes (e.g., the user was satisfied or the final objectives were high quality). If we had some metric of success or quality for each session or fragment (even a coarse one, like "did the user accept the advisor's proposal? yes/no"), that could serve as a powerful training signal. We could correlate fragment types with successful interactions. For example, maybe sessions where the advisor asked more reflective questions ended up more successful – that would reinforce an objective to do that. Or if certain procedural outlines correlate with user satisfaction, we'd emphasize those. In future data gathering, incorporating **feedback signals** (user ratings, whether the final solution was adopted, etc.) would let us not just infer what agents do, but what they should do more or less of to be effective. This moves towards reinforcement learning type signals, which are valuable for training optimization.
- **Intent Annotations or Dialogue Act Tags:** As discussed, one could classify each utterance by its intent (question, instruction, confirmation, etc.). Currently, the rationale kind of serves this purpose at a high level (reflective, procedural...), but those are broad. A more fine-grained annotation of each fragment's function in conversation would allow more precise objective formulation. For instance, tagging a fragment as "clarification question" vs "open-ended question" vs "reflection statement" can help differentiate training scenarios. If manual annotation is costly, even using a model to generate these tags for analysis could help. These tags become additional features for clustering or for training supervised models to detect when the agent is doing X or Y. They can also reveal if certain important acts are underrepresented (maybe the advisor seldom uses a "summarizing statement" tag – meaning we should train more on summarizing). So, having a **dialogue act layer** would be beneficial.
- **Emotional or Sentiment Context:** While not a primary focus in this technical context, user sentiment or confusion level could be a helpful signal. If, say, the user expresses confusion, that's a cue for the advisor to engage a certain mode (clarify further, simplify language). Right now, the logs likely don't label user emotion or satisfaction at each turn. If future data can include sentiment analysis of user responses or a simple "user asked for re-clarification here", we could identify which advisor behaviors triggered confusion or resolution. This goes somewhat into UX territory, but it's relevant for training advisors that interact with humans to be sensitive to those cues.
- **Connection to Source Materials (Traceability):** The methodology emphasizes traceability – each fragment has an ID tied to source. However, we might also want to link fragments to the *knowledge source or principle* they reference, if any. E.g., a fragment that states "Prompts must be treated as production assets…" came from a source document or user input (with [Source: L1, L2]). If we tag that fragment with which principle it belongs to (like a principle ID from the blueprint), then we can see how often each principle was mentioned or learned. In the data, we see those [Source: L1] tags inside the text but not as structured data. Structuring it (like an additional column "linked_objective_id") would allow a very direct analysis: which objectives are heavily discussed vs which not at all. For training, if an objective was never mentioned, maybe the agent didn't learn it – we might need to simulate scenarios for it.

- **Temporal Information and Turn Durations:** Including timestamps or at least turn indices might expose interesting dynamics (e.g., does the advisor take longer or produce more fragments in early phase vs later? Did the meta-agent's process slow down at some complex step?). Temporal data might help with objectives like the sustainability one (monitor how long interactions go). While not critical, it can reveal if an agent tends to "ramble" or respond too quickly without reflection, etc. Possibly, a future system could monitor time taken per reasoning step as a signal of complexity or agent certainty.
- **Error or Correction Markers:** Were there places where the advisor made a mistake and corrected itself, or had to revise something after user feedback? If yes, marking those would be very valuable: they highlight learning opportunities (the agent should avoid that mistake next time or handle that differently). The current fragment list might include the correction as just another fragment (maybe reflective "Okay, let's adjust since X was wrong"), but we wouldn't know it was a correction unless reading full context. Explicitly labeling corrections or missteps in logs could help target objectives around error handling and recovery.
- **User Profile or Task Type Metadata:** If applicable, knowing what kind of user or what category of task each session was could allow tailoring objectives. For example, if some tasks are design tasks vs troubleshooting tasks, the advisor might need slightly different approaches. Our analysis was generic, but in future, classifying conversations by scenario and seeing which objectives matter per scenario could refine training (e.g., maybe in troubleshooting, verifying assumptions is key, whereas in brainstorming, encouraging creativity might be a new objective). So, adding a "task type" label to each conversation or even a brief description of the goal could contextualize the fragments.

To summarize, enhancing the dataset with **richer annotations and metadata** will improve the fidelity of any automated inference:

- We need context to understand sequence and causality.
- Clear speaker roles to separate agent behaviors.
- Success/failure signals to focus training on what works.
- Fine-grained tags for utterance function and content to allow targeted objective mining.
- Trace links to source principles to measure coverage.
- Possibly additional layers like sentiment or error flags to incorporate human reaction.

By addressing these missing pieces in future data collection (for instance, designing the logging system for the agents to output structured event logs, or performing post-hoc annotation of dialogues), we will get a more **multi-dimensional view** of the conversation. That in turn leads to more accurate and comprehensive training objectives. Essentially, the more we know about each fragment's role and effect, the better we can translate it into what the agent should learn from it.

Reusable System Architecture

Finally, we propose a **reusable system architecture** for continuously inferring training objectives from future advisor and prompt-architect logs. The idea is to create an end-to-end pipeline that can take raw conversational data from these agents' interactions and output an updated set of learning objectives or insights, in a repeatable and scalable manner. Such a system ensures that as the agents are used (and conversations accumulate), the training objectives can evolve and stay aligned with actual usage patterns and emerging challenges.

The architecture can be outlined in stages, each responsible for a part of the inference process:

1. **Data Ingestion & Preprocessing:**
   Raw logs from agent conversations (could be chat transcripts, JSONL files, etc.) are collected. This includes dialogues of the Advisor with users, and potentially logs of the prompt-architect agent's internal reasoning or interactions (if available). The ingestion system normalizes these logs (unifying format, ensuring all relevant fields like timestamps, speaker roles, and session IDs are present). Preprocessing then segments the conversations into meaningful units:
   - Identify candidate **learning fragments** using an updated extraction method (for example, the improved semantic pattern mining or a lightweight classifier to flag sentences of interest, not just based on fixed verbs). This stage essentially reproduces and generalizes the step that created `advisor_learning_first_semantics.csv`. It should be flexible to agent type (maybe a slightly different set of patterns for the prompt-architect logs vs advisor logs, since their content differs).
   - Attach any available metadata to each fragment: conversation context snippet, speaker role, turn position, etc., as discussed in Missing Data. The output of this stage is a structured dataset (let's call it *Enriched Conversation Fragments*) that includes all fragments with metadata and is ready for analysis.
2. **Cognitive Classification & Weighting:**
   Each extracted fragment is then analyzed to classify its cognitive or functional category. This corresponds to applying one (or multiple) of the **rubric systems** discussed. For example:
   - Use a **trained classifier or rule engine** to assign categories like Reflective/Procedural/Descriptive/Minimal (improved version) or other labels (dialogue act, Bloom level, etc.). This could output multiple labels or scores per fragment, creating a high-dimensional profile of the fragment.
   - Optionally, compute a **relevance weight or score** for how strongly the fragment represents a learning moment. The existing weighting model (0.1–0.99) can be one input, but we could refine it by also considering context (e.g., if a fragment is central to resolving the task, boost its weight). This could be learned via regression if we had training data (like which fragments were crucial).

- Perform **role-based tagging:** mark whether this fragment pertains to Advisor behavior, Meta-agent behavior, or both. This might be straightforward since advisor logs and architect logs might be separate sources. But if in a single conversation the advisor references what the architect should do (like user says "architect will use this"), that fragment might be relevant to both roles.
- The output is the fragment dataset augmented with labels such as {category labels, scores, role tag}. Essentially, we mirror what `advisor_learning_weights.csv` had but with potentially richer columns (and for both agent types as applicable).

3. **Pattern Mining & Clustering (Insight Generation):**
With classified fragments, the system now seeks higher-level patterns:
   - **Clustering:** Using vector representations of fragments (that can encode semantic meaning beyond the explicit labels), cluster the fragments to see emerging themes. This might separate out distinct sub-categories of reflections or plans beyond what the initial classification had. For instance, reflective fragments might cluster into "understanding-checks" vs "strategy-evaluations".
   - **Frequent Pattern Extraction:** Use algorithms to find common n-grams or semantic motifs in the fragments. E.g., maybe "I will" starts a lot of high-weight fragments (indicating the agent planning) or "ensure [X]" is a motif (indicating a verification step). Identifying these can highlight recurring techniques that should be formally captured as objectives or best practices.
   - **Outlier Detection:** Find fragments that didn't fit any category well or got very low weights but intuitively seem important. These might indicate either gaps in the classification (e.g., a new type of cognitive move not accounted for) or noise. Outliers should be flagged for human review to decide if a new objective category is needed or if they are irrelevant.
   - **Trend Analysis:** If this system runs continuously over time, it can track if certain types of fragments are becoming more or less frequent. For example, as the agent improves, perhaps "clarification questions" become shorter or fewer (if the user instructions improve). Trends could inform objective emphasis (e.g., if an important behavior is declining, maybe training needs reinforcement in that area).

   This stage essentially converts raw classified data into **insights** – groupings and statistics that suggest what to focus training on. It aligns with the planned "thematic clustering" and analysis steps in the methodology.

4. **Objective Formulation & Mapping:**
Now, the system needs to translate the patterns into actual training objectives. This involves:

- **Mapping clusters/categories to objectives:** For each identified cluster or category of fragments, generate a candidate training objective. This could be templated (e.g., for a cluster identified as "clarify requirement questions", generate objective "Agent should practice asking clarifying questions when requirements are ambiguous"). A knowledge base of objective phrasing (possibly seeded from learning.json's style) could be used to help word these consistently. Some could map directly to existing objectives: e.g., cluster around "lifecycle management" maps to the known objective about prompt lifecycle. The system should recognize that and link them (perhaps via keyword match or semantic similarity to the text of known objectives).
- **Gap detection (vs existing framework):** By linking observed patterns to known objectives (like those in learning.json or an internal list of current training objectives), the system can mark which objectives are evidenced in the data and which are not. If some known objective never appears in any fragment, that might be flagged as a *training gap* (maybe the agent isn't demonstrating that principle, so it needs more focus or scenario coverage). Conversely, if a strong cluster emerges that doesn't match anything in the existing list, that suggests a new objective proposal. For instance, if advisors consistently do something clever that wasn't anticipated, that becomes a new best practice objective.
- **Ranking or weighting objectives:** Not all objectives are equally urgent. The system can use the fragment weight data or frequency to prioritize. Objectives backed by many high-weight fragments in the logs are likely core skills (thus high priority for training). Rarely seen ones might be lower priority or perhaps future capabilities. We might produce a "heat map" of objectives by how much supporting evidence exists.
- **Human-in-the-loop validation:** At this formulation stage, it's wise to involve a human expert or at least present the draft objectives for review. The architecture could include an interface where an analyst sees the suggested objectives, along with example fragments supporting each and stats, and can modify phrasing or approve/reject additions. This ensures quality control before updating the official curriculum.

5. **Update Learning Objectives Knowledge Base:**
   The final list of confirmed or adjusted objectives (with their metadata: which agent role, linked source evidence, date observed, etc.) is then stored in a **Learning Objectives Repository**. This could be an updated `learning.json` (versioned) or a database that tracks objectives, their definitions, and their status (new, confirmed, to-be-developed, etc.). The repository might also keep associations: which objectives map to which training data or scenarios, tying back into the training loop.

   - If the system architecture is integrated with the training pipeline, these updated objectives could directly inform curriculum generation or fine-tuning priorities. For example, if "ask clarification questions" is identified as an objective to reinforce, the training system might increase the weight of dialogues in fine-tuning that contain that pattern, or add synthetic training prompts for it.

- The architecture should support **ongoing accumulation**: as more conversations come in, this pipeline runs (perhaps in batches or periodically) and keeps refining the objectives. It's an iterative learning process for the development team as much as for the agent.

6. **Reporting & Monitoring:**
   The architecture would include a monitoring component that produces reports or dashboards from the above process. For instance:
   - A dashboard showing the distribution of fragment types over time (to monitor changes in agent behavior).
   - A list of objectives with indicators of evidence (how many times seen, trend up/down).
   - Alerts for novel behaviors (new cluster without objective mapping).
   - Perhaps evaluation metrics if we integrate outcome success: e.g., "Conversations where Objective X was fulfilled had a 90% success rate."
     These reports help stakeholders understand where the agent's learning focus should shift. It also provides transparency (one of the design philosophies mentioned was transparency) so that one can trace why a certain objective was added – you can point to the conversation data that justified it.

To give a concrete picture, imagine this architecture in use: The Advisor and Meta-agent have been deployed in a pilot. After 100 sessions, we run the pipeline. It ingests all chats, extracts 5000 fragments. Classification finds, say, 300 reflective (mostly advisor), 100 procedural (advisor), 200 procedural (meta-agent logs), 50 new type "creative suggestion" that wasn't in our original taxonomy. Clustering shows "creative suggestion" is a cluster (advisor proposing novel ideas, perhaps). The system sees that's not in the current objectives list. It proposes a new objective: "Advisor should be able to generate creative solution ideas when appropriate." The human reviewer agrees and adds it. Meanwhile, it matches many meta-agent fragments to the known objective "structured output validation" from learning.json (since the meta-agent was logging schema validations). It marks that objective as actively demonstrated. Another objective "context isolation" wasn't seen at all – maybe the meta-agent hasn't encountered that scenario, so it flags it as an area to test or train further. All this goes into an updated objectives file. The training team then knows to include some sessions focusing on context isolation, and to commend that creative suggestions are now part of the advisor's repertoire to cultivate.

Scalability and Reusability: This architecture is designed to be reusable in that it can handle new logs with minimal manual adjustment. If new agent roles are introduced in the future, the classification step might need new patterns, but the pipeline remains similar. The modular separation (extract -> classify -> cluster -> map to objectives) means improvements in one module (say a better classifier model) can be integrated without overhauling the whole system. It's also scalable: clustering and classification can be done with big data techniques or cloud services if logs grow large. By automating much of the insight extraction, it reduces the burden on human experts to comb through transcripts; they can instead focus on reviewing aggregated insights and making final objective decisions.

In essence, this system architecture closes the loop between **agent performance in the wild** and **training program updates**. It treats conversation logs as a continual source of lessons. Over time, this should lead to a virtuous cycle: agents improve, their conversations reveal higher-level new challenges, those become new training objectives, which then further improve the agents. This keeps the training objectives **dynamic and evidence-based**, ensuring the agents remain aligned with real-world demands and the evolving blueprint of the AI system.