FEDERATION OVER TEXT

Dixi Yao, Tahseen Rabbani, Tian Li
University of Chicago {dixi,trabbani,litian}@uchicago.edu

ABSTRACT

LLM-powered agents often reason from scratch given a new problem instance and lack effective mechanisms to transfer learned skills to other agents. We propose the first federated-like framework, *federation over text* (FoT), where multiple agents solving different tasks collectively learn a library of metacongnitive insights by federating local reasoning processes iteratively. Instead of federation over gradients (i.e., distributed training), FoT operates at the **semantic level**. Each agent does local thinking and reflection on their specific tasks and shares reasoning traces with a server, which aggregates them into a cross-task (and cross-domain) insight library that agents can leverage to improve performance on new reasoning tasks. Experiments show that FoT improves reasoning ability and efficiency on mathematical problem-solving and machine learning insight discovery. For math problems, we achieve up to **63%** improvement in accuracy and **28%** reduction in generated tokens across benchmarks.[1]

1 INTRODUCTION

LLM-powered agents are increasingly deployed for complex, multi-step tasks (Guo et al., 2025) . However, they can suffer from two major limitations. First, reasoning can be inefficient: agents often reason from scratch, repeatedly solving similar problems, which can be computationally expensive and slow. Second, reasoning typically occurs in an isolated manner, where agents simply transmit user prompts or final outputs (e.g., Ke et al., 2025). One agent's hard-won skills or reasoning procedures for solving a problem can be lost. It remains unclear how to easily transfer learned skills (thinking procedures) to other agents or domains.

How agents can interact at a higher metacognitive level to create reusable insights (Didolkar et al., 2025; Tran et al., 2025) that generalize to new agents or domains remains underexplored. At the same time, traditional federated or distributed learning schemes (McMahan et al., 2017) may not be suitable for learning generalizable insights from agents' reasoning traces. Averaging the gradients or weights of entire LLMs is computationally prohibitive and, more importantly, is not designed to explicitly share *abstract, high-level reasoning strategies*. We propose *federation over text* (FoT), the first federated-like framework for multi-agent collaboration that shares learned skills (i.e., reasoning processes) rather than model weights or prompts. FoT enables a set of agents to collaboratively build a shared, evolving library of reusable *insights* from metacognitive reasoning traces without sending local problem instances. The insight library is a high-level aggregation of the common principles and ideas of the agent tasks. For example, agents working on text modeling and agents exploring admixture inference in population genetics can share the same underlying principle and techniques.

Unlike traditional federated learning which averages gradients or model updates in a high-dimensional, uninterpretable space, FoT operates at the **semantic** level. Agents transmit concise metacognitive summaries of their problem-solving processes to a central (physical or virtual) server. The server, acting as an aggregator, identifies and distills recurring reasoning fragments from these summaries into new, explicit insights. This allows agents to create a library of generalizable insights that improves the reasoning capability and efficiency of all agents in the network. We note that FoT is a *general* framework that allows to use any local agentic models coupled with any local thinking and self-reflection approaches, as long as the reasoning traces are shared at each iteration.

Interestingly, we highlight that our FoT approach is analogous to the classic federated/distributed learning setting. In particular, in Table 1, we show that these two regimes can be interpreted within

[1]Work in progress. A longer version of this manuscript is on arXiv.

the same conceptual framework, where the core algorithmic components can be mapped to each other. By making these connections and drawing inspiration from federated optimization, it enables a range of potential design choices within FoT, regarding how to evaluate such a multi-agent iterative reasoning framework, how to personalize the services, and how to perform better joint reasoning. We discuss these implications and some future directions in more detail in Section 3.1.

In this work, we present preliminary results applying FoT to challenging mathematical problem solving and machine learning insight discovery (Section 4). For instance, we show that FoT can improve accuracy by **63%** on the AIME24 benchmark compared with directly querying the LLM model while decreasing the number of generated tokens by **28%** on average. In the second application, we demonstrate that our insight library generated by learning from old machine learning papers can cover the core technical contribution of over **70%** of subsequent machine learning papers.

2 Related Work

Multi-Agent Collaboration. Recent agent systems explore collaboration mechanisms for solving complex tasks. A common paradigm decomposes a complex task into smaller subtasks, each solved by an individual agent, followed by aggregation of partial results (Tran et al., 2025). Related efforts aggregate knowledge at scale: Schmidgall et al. (2025) propose agent laboratory to aggregate human research artifacts via an LLM server, while ResearchTown (Yu et al., 2025) simulates a community of LLM agents for writing and reviewing papers. In contrast, Federation over Text (FoT) operates at a higher level of abstraction, focusing on discovering shared underlying principle or insight of solving related problems rather than coordinating end-to-end task execution.

Collaborative and Federated Learning. Federated learning (FL) (McMahan et al., 2017) aggregates model parameters from multiple clients while keeping data local. Existing FL multi-agent systems primarily focus on training a reinforcement learning model or agent model via classic parameter- or gradient-level aggregation (Li et al., 2025), which is insufficient for capturing abstract, high-level reasoning strategies. Social learning (Mohtashami et al., 2023) allows teacher agents to generate synthetic examples or abstract prompts that are aggregated to train a student model. However, the student solves the same task as the server, since social learning transfers knowledge from teachers to a student. FoT and FL share interesting connections (Section 3.1); and FoT operates in the semantic space while being compatible with heterogeneous agent models and domains.

Reusable Agent Experience. To embed past experience into LLM-based agents, existing approaches such as `Skills` and `Agents.md`[2] rely on human-authored or prompt-engineered artifacts to explicitly package domain knowledge and agent instructions. Hence, researchers further study metacognition (Didolkar et al., 2025) and self-improvement (Zhuang et al., 2026) strategies, aiming to automatically capture and reuse previously successful behaviors by storing them as structured handbooks that guide future decision-making. FoT can plug in any existing self-improvement method as the agent local thinking approach, and collaboratively construct and refine insights given agent reflection traces in a federated manner.

3 Federation over Text

In Figure 1, we present the overall workflow of FoT, comparing it with federation over gradients (e.g., FedAvg (McMahan et al., 2017)). Federated learning aims to train a global model while keeping all data on local devices, whereas FoT aims to build an *insight library* that helps both existing and incoming agents better resolve various tasks, where the insight library is a high-level aggregation of common principles and ideas across heterogeneous agent tasks. As shown in Figure 1, federation over gradients first broadcasts an initial global model to each client, and each client trains a local model. Clients then upload model weights to the server, which aggregates them to update the global model. This process repeats iteratively. Similarly, FoT starts with an empty library, and each agent conducts local reasoning via any model. Reasoning traces are then uploaded to the server, which aggregates them to generate and merge new insights into the library. The library is next distributed in the following round for further improvement. The pseudo algorithm is in Appendix B.

[2] `https://github.com/anthropics/skills`; `https://agents.md/`

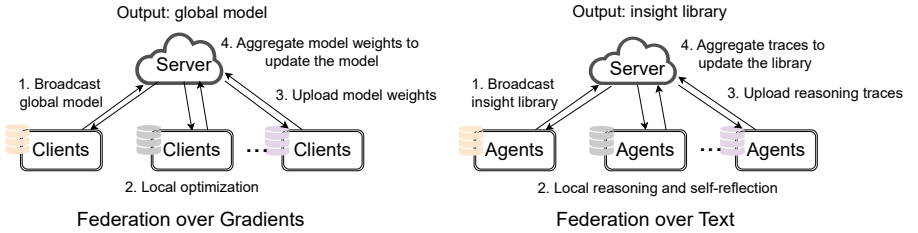

Figure 1: The workflow of federation over (accumulated) gradients and federation over text.

Table 1: Comparison of key aspects between federated learning over gradients (e.g., FedAvg) and our proposed federation over text (FoT).

Components	**Federation over Gradients**	**Federation over Text**
Learning Objective	Learn a global model that generalizes to test data or new clients	Learn an insight library that scales up and improves the reasoning of existing and new agents
Local Work	Local training via any optimization method	Local problem solving via any thinking and self-reflection approach
Local $\rightarrow$ Server	Clients upload locally trained model weights or gradients	Agents upload reasoning traces from their problem-solving processes
Server Aggregation	Averaging over uploaded model weights or gradients	Text generation by an agent using curated prompts
Server $\rightarrow$ Local	Current learnt global model	Current learnt insight library
Privacy	Clients share model updates or parameters rather than raw data	Agents share abstracted insights rather than raw problem instances
Personalization	Clients maintain personalized models adapting to local distribution	Agents adopt personalized reasoning strategies based on the task at hand

In the workflow, several (or potentially many) decentralized agents each solve a specific problem or task. These tasks can span different disciplines such as mathematical reasoning, coding, and scientific question-answering. Given a problem, an agent first generates a solution, then reflects on whether the reasoning process contains reusable techniques and what key skills can be extracted. Finally, the agent produces a *reasoning trace* for each problem based on the solution and reflection, specifying when and how to apply the extracted skills through step-by-step instructions. Only traces are shared with the server; all answers, contents of reasoning process generated in the solution phase, and reflections remain local. If an insight library is available, agents use it as reference.

After collecting traces, the server first clusters reasoning traces sharing similar techniques, then uncovers connections among traces that solve the same problem using different techniques or can be combined for joint use. The server then updates the current insight library by distilling high-level insights rather than simply concatenating reasoning traces. Finally, the server broadcasts the updated library to all agents for the next iteration. The aggregated library can be re-used as an external tool to improve reasoning of either existing agents, or future agents that are not in the current network. We provide all the designed prompts that we use in Appendix C.

3.1 Implications of Connecting FL with FoT

Considering that both Federated Learning (FL) and Federation over Text (FoT) operate in a federated manner, we can leverage insights on how to optimize FL to similarly optimize FoT, realizing more efficient and effective self-improvement. In federated learning, clients may adopt different optimization methods to improve training performance and efficiency. Analogously, in FoT, each agent may leverage distinct local reasoning strategies and prompt designs to generate insights.

For example, in federated learning, personalized federated learning addresses challenges such as uneven data distribution across clients, heterogeneous memory and computational capabilities, and diverse task requirements. Similarly, in FoT, agents can be personalized on each client, which is often easier to implement than in FL. For example, personalized FL uses techniques such as clustering to group clients with similar data distributions. Analogously, in FoT, we can cluster agents handling similar tasks to better identify relationships between their reasoning traces. In FoT, each agent may employ different language models or even different reasoning systems. Another aspect is evaluation. In federated learning, we can evaluate in two scenarios: performing inference on previous clients with new data, or letting new clients use the global model for inference. Similarly, in FoT, we can evaluate scenarios where we either let previous agents use the insight library to

resolve new tasks, or send the insight library to new agents and let them complete tasks; see the next section for details. Future work should continue to fully explore the rich design space of FoT inspired by further connections with FL, such as personalization strategies, controlling distribution drift across rounds, and optimizing communication efficiency between agents and the server.

4 APPLICATIONS

In this section, we study two applications where the insight library (that captures a set of shared, non-trivial principles of a type of problems) improves downstream performance and reasoning efficiency.

4.1 MATHEMATICAL PROBLEMS

In this application, our goal is to solve math problems at various difficulty levels, and we assume that solving them can inform each other. There are eight agents, each working on a benchmark dataset from AIME24, AIME25, AMC, CCEE, CNMO, WLPMC, V202412 Hard, and V202505 Hard from LiveMathBench. In each round, each agent uses the current insight library as reference and reasons over their local benchmark again. We experiment with DeepSeek-R1-Distill-Qwen-7B (DeepSeek) and Gemini 3 Pro as the base LLM models. In FoT, each agent and the server can use different models, but in this experiment, we use on the same LLMs in each row of the table below.

Table 2: Mathematical problem-solving accuracies across benchmarks and settings. With the FoT-generated library, agents can solve problems better. Stronger LLMs yield greater benefits, and with a library generated by Gemini in FoT, local DeepSeek agent achieves a much higher accuracy.

		AIME24	AIME25	AMC	CCEE	CNMO	WLPMC	V202412 Hard	V202505 Hard
DeepSeek	Single Agent	0.500	0.400	0.674	0.841	0.722	0.273	0.524	0.360
	FoT (Round 2)	0.500	0.400	**0.717**	0.841	0.722	**0.364**	0.524	0.310
	FoT (Round 3)	0.500	0.400	**0.717**	0.841	0.722	**0.364**	0.524	0.360
Gemini 3 Pro	Single Agent	0.967	0.933	0.935	0.864	0.889	0.727	0.762	0.690
	FoT (Round 2)	**1.000**	**0.967**	0.935	**0.909**	**0.944**	**0.909**	**0.952**	**0.720**
DeepSeek + Gemini 3 Pro Insight Library		**0.833**	**0.567**	**0.696**	0.841	0.722	**0.455**	0.524	0.360

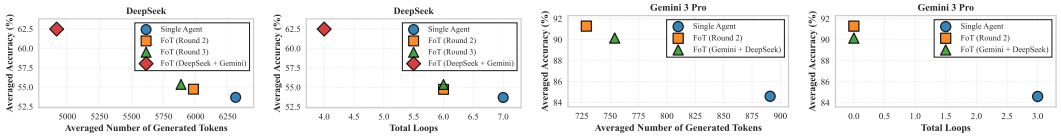

Figure 2: Comparison of reasoning efficiency and accuracy across different settings. With the presence of a insight library, the system generates fewer tokens during the reasoning processes while achieving better final performance. Additionally, with FoT, agents produce fewer loops, reducing the occurrence of repeated content (Pipis et al., 2025).

From Table 2, we observe that for challenging tasks such as AIME25 and V202412 Hard, FoT constructs useful insights that improve accuracies. Moreover, insights produced by stronger LLMs transfer effectively: when DeepSeek agents use a library generated by Gemini 3 Pro, performance improves substantially (comparing the first panel with the last row), demonstrating that FoT effectively distills and replays successful reasoning experience across agents. From Figure 2, we see that FoT constructs an insight library that leads to fewer loops and fewer generated tokens in the answers.

4.2 MACHINE LEARNING INSIGHT DISCOVERY

In this application, we aim to collectively discover insights from ML papers, which is different from prior works that build engineered workflows for paper writing. Each agent is assigned to one paper and tasked with identifying its main non-incremental novel contribution. We then evaluate how many future papers could be guided by insights in the library as their core contribution. Specifically, we input a paper and the insight library to an LLM, which judges on four aspects: concrete methodology usage, how methods are presented, whether the paper's core contribution would fundamentally differ without the insight, and whether the insight is novel, with details in Prompt 5.

We consider two settings. In the first, 281 agents each read one accepted ICLR 2023 paper, and we evaluate on ICLR 2024 accepted papers. All agents, the server, and the evaluation LLM use

Gemini 3 Pro Preview. However, this LLM may already contain knowledge of ICLR 2024 papers. Therefore, in the second setting, we use Gemini 2.0 Flash: 453 agents read oral and spotlight papers from ICLR 2024, and we evaluate on ICLR 2025 papers. Gemini 2.0 Flash is designed to have no access to 2025 papers, as its cutoff date is August 2024, before the ICLR 2025 submission deadline.

We evaluate the percentage of subsequent ML papers whose core technical contribution can be covered by principles from FoT generated insight library (22 insights total). We compare against two baselines. First, we let a standalone Gemini model generate the top 200 insights based on the same insight definition and the keywords of ICLR 2023 papers (setting 1) or ICLR 2024 oral and spotlight papers (setting 2), then evaluate whether the insights guide papers from the following year. Second, we use Vertex AI retrieval-augmented generation (Lewis et al., 2020) (RAG) to build a datastore of all selected papers from the previous year, ask the LLM to generate top-200 insights with retrieved content as reference, and apply the same criteria. These baselines assess whether FoT improves insight discovery beyond the LLM's standalone capabilities and beyond simply storing old papers for retrieval. We also verify the validity of generated insights and demonstrate a subset in Appendix D. In summary, FoT generates insights corresponding to cutting-edge research topics from ICLR 2023 and 2024 papers, though under different terminology.

Table 3: Percentage of papers guided by insights generated from different methods. In both models, FoT improves over the baseline without the library ('Singel Agent') and the baseline that simply stores previous tasks in RAG for retrieval ('RAG').

	FoT (Gemini + Insight Library)				**Single Agent (Gemini)**				**RAG (Gemini + Vertex AI)**			
	Overall	Oral	Spotlight	Poster	Overall	Oral	Spotlight	Poster	Overall	Oral	Spotlight	Poster
Gemini 3 Pro	**67.4**	67.4	68.7	**67.2**	41.3	**70.9**	**73.8**	33.3	66.5	58.1	69.2	66.4
Gemini 2.0 Flash	**82.4**	**82.2**	**82.1**	**82.4**	0.7	0.5	0.8	0.7	77.4	63.4	67.4	79.6

REFERENCES

A. Didolkar et al. Metacognitive reuse: Turning recurring llm reasoning into concise behaviors. *arXiv:2509.13237*, 2025.

D. Guo et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv:2501.12948*, 2025.

Z. Ke et al. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv:2504.09037*, 2025.

P. Lewis et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*, volume 33, pp. 9459–9474, 2020.

C. Li et al. Fedcot: Communication-efficient federated reasoning enhancement for large language models. *rXiv:2508.10020*, 2025.

B. McMahan et al. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

A. Mohtashami et al. Social learning: Towards collaborative learning with large language models. *arXiv:2312.11441*, 2023.

C. Pipis et al. Wait, wait, wait... why do reasoning models loop? *arXiv:2512.12895*, 2025.

S. Schmidgall et al. Agent laboratory: Using llm agents as research assistants. *EMNLP*, pp. 5977–6043, 2025.

K.-T. Tran et al. Multi-agent collaboration mechanisms: A survey of llms. *arXiv:2501.06322*, 2025.

H. Yu et al. Researchtown: Simulator of human research community. In *ICML*, 2025.

Y. Zhuang et al. Test-time recursive thinking: Self-improvement without external feedback. *arXiv:2602.03094*, 2026.

A USAGE OF AI IN WRITING

We employ large language models (LLMs) primarily to improve the grammar and clarity of our writing and assist coding. All research ideas, directions, and decisions, however, are independently conceived and carried out by the authors.

B COMPLETE ALGORITHMS

Algorithm 1: Federation over Gradients

Input: Initial global model, client datasets
for *each iteration* **do**
  for *each **Client*** **do**
    Local optimization starting from the current global model
    Upload new model weights
  end
  Server:
  Collect all local model weights
  Aggregation over model weights
  Broadcast the global model to clients
end
Output: Global model

Algorithm 2: Federation over Text

Input: Empty insight library, agent tasks, LLM
for *each iteration* **do**
  for *each **Agent*** **do**
    Local reasoning given the LLM and current insight library
    Upload reasoning traces to the server
  end
  Server:
  Gather all reasoning traces
  Update the insight library
  Broadcast the new library to agents
end
Output: Insight library

C COMPLETE PROMPTS

Prompt 1 is used for the reflection step before an agent generates reasoning traces during Step 2: Local reasoning and reflection in Figure 1.

Prompt 1: Definition of reflection on solution

```
Analyze the solution below to extract procedural knowledge that reflect
    the reasoning traces.

Problem:
{problem}

Step-by-Step Solution:
{solution}

Your task: Extract the fundamental techniques used in reslution that can
    be packaged as reasoning traces. Focus on:

1. What step-by-step procedures were used? How can these be repeated?
2. What conditions determined which approach to use? When should each
    technique apply?
3. What methods, strategies, or workflows can be applied to similar
    problems?
4. What made this approach effective? What should someone know to use it
    correctly?
5. What types of problems would benefit from these techniques?

Output your analysis covering:

### I. Procedural Knowledge
- Break down the solution into clear, repeatable procedures
```

```
- The extracted traces should be concrete solutions rather than general
    principles.

### II. Reusable Techniques and Methods
- List specific techniques, strategies, or workflows used
- The techniques should be solid on practical questions rather than very
    general and high-level principles.
- For each technique, identify:
  * When it should be used (conditions/triggers)
  * How it was applied (concrete steps)
  * Why it was effective (insights)
  * What problems it could solve (applicability)

### III. Critical Insights and Guidelines
- What key insights made this solution work?
- What common pitfalls should be avoided?
- What variations or edge cases should be considered?

Focus on extracting actionable, procedural knowledge that can be
    packaged as reusable insights for similar problems.
```

Prompt 2 is used for the generation of reasoning traces during Step 2: Local reasoning ans self-reflection in Figure 1. The description includes our definitions of reasoning traces.

Prompt 2: Reasoning Trace Prompt

```
Extract reasoning traces from the solution below. Analyze the solution
    and reflection to identify concrete, actionable traces that similar
    problems can be solved via the traces

Problem: {problem}

Solution: {solution}

Reflection: {reflection}

**Your Task:**
Identify and extract all reusable reasoning traces, techniques, and
    methods used in the solution. Each trace should be a concrete
    procedure that can guide someone to solve similar problems.

**What Makes a Good Reasoning Trace:**
- A specific technique or method that was used in the solution.
- Something that can be applied to similar problems, not just this one.
- Includes guidance on when and how to use it with clear steps that can
    be followed if necessary.
- Not repeatance of already well-known or commonly adopted techniques.
- Not too general and high-level but contains actional procedural
    knowledge.

**Description Must Include:**
1. **Core idea**: The fundamental concept of what this trace is about.
    What is the main technique or method? What does it do?

2. **When to use**: Explain when this skill should be applied. What
    types of problems? What conditions must be met? What situations
    trigger this skill?

**Output Format (Simple JSON):**
Output a simple JSON object with skill names as keys and descriptions as
    string values:
```

```
{{"trace_name": "description"}}

Format Rules:
- Use valid JSON format
- Each trace name must start with "trace_"
- Keep JSON simple - no nested objects, just key-value pairs
- Escape quotes in descriptions with backslash: \\"

**Example:**
{{
  "trace polynomialFactoring": "The major idea is how we can turn a
    polynomial into a product of simpler expressions. This skill is
    particularly useful for quadratic and higher-degree polynomial
    equations where factoring can simplify the problem. Factoring
    reduces complex polynomials to simpler equations. When solving
    equations with polynomial expressions that can be factored,
    especially when the polynomial has recognizable patterns like
    difference of squares (a^2-b^2), perfect square trinomials
    (a^2+2ab+b^b), or common factors.
  "trace depthFirstSearchImplementation": "This algorithm is essential
    for problems involving path finding, cycle detection, topological
    sorting, connected components, or exploring all possible solutions
    in a search space. DFS explores depth before breadth, using
    stack-based recursion or explicit stack. It is memory-efficient for
    deep structures and naturally handles backtracking. The visited set
    prevents infinite loops and redundant work. DFS is the foundation
    for many graph algorithms including topological sort, strongly
    connected components, and maze solving. When you need to explore or
    traverse a graph, tree, or nested structure systematically, going as
    deep as possible before backtracking. Use DFS when you need to visit
    all nodes in a connected component, find paths between nodes, detect
    cycles, or explore recursive structures like file systems, nested
    data, or game states. "
}}

**Output your response as a valid JSON object only:**
```

Prompt 3 is used for clustering and identifying connections between reasoning traces before updating the insight library during Step 4: Aggregate traces to update the library in Figure 1.

Prompt 3: Prompt for constructing relationship between reasoning traces

```
You are analyzing a collection of reasoning traces generated for
    problem-solving.  Understand their relationships and structure and
    build a profiling of their relationships.

**Collected {len(insight_store)} traces in total:**
Note: This collection includes ALL traces from all problems without
    deduplication. Similar names with different indices (e.g., _001,
    _002) represent different occurrences that may have variations in
    their descriptions.

{insights_text}

**Your Task:**
Analyze these traces and build a profiling of their relationships:

1. **Identify Clusters**:
   Group related traces that share:
   - Resolve the same or similar problem
   - Similar approaches or techniques
```

```
   - Nearly identical traces (e.g., same trace with minor variations in
    description or parameters)
   - Traces in the same cluster should be higly similar.

2. **Build Trace Relationships**:
Record all important relationships - traces don't exist in isolation and
    build a relationship graph that records:
   - **Prerequisite relationships**: Traces that must be learned/used
    before others
   - **Composition relationships**: Traces that can be chained/composed
    together
   - **Alternative relationships**: Different approaches to the same
    problem
   - **Complementary relationships**: Traces that work better together
    than individually used
   - **Derivation relationships**: Traces derived from or based on others
   - **Similar relationships**: Traces that are similar but not identical
Map relationships between traces within clusters and across clusters.

# Output Format:
{{
  "clusters": [
    {{
      "cluster_id": 0,
      "cluster_name": "Domain/Theme Name",
      "traces": ["name1", "name2", "name3"],
      "theme": "What is the high-level techniqual idea of the traces in
    this cluster?",
    }}
  ],
  "relationships": [
    {{
      "trace_a": "trace_name1",
      "trace_b": "trace_name2",
      "relationship_type":
    "prerequisite/complementary/alternative/similar/derived_from/composes_with",
      "description": "How these traces relate to each other and Why"
    }}
  ]
}}

**Output your analysis as JSON only:**
```

Prompt 4 is used for updating the insight library during Step 4: Aggregate traces to update the library in Figure 1.

Prompt 4: Definition of Insight Library

```
**Your Task:**
You are extracting fundamental insights from a collection of
    problem-solving traces.

**Output Requirements (STRICT):**
1. Return EXACTLY one valid JSON object and nothing else.
2. Do NOT output markdown code fences.
3. Do NOT output explanations, notes, reasoning, prefixes, suffixes, or
    `<think>` content.
4. Do NOT output list/array at top-level.
5. Every key must start with "insight_".
6. Every value must be a single string.
7. No nested objects, no nested arrays.
```

```
**Required JSON shape:**
{{
    "insight_name1": "description string",
    "insight_name2": "description string"
}}

**Formatting Rules:**
- Use valid JSON syntax only.
- Keep top-level as key-value pairs only.
- Escape quotes in descriptions with backslash: \\"
- If uncertain, still output a valid JSON object (possibly with fewer
    insights), never free text.

Your goal is to extract a comprehensive set of fundamental, cross-domain
    insights that can be derived and applied beyond their original
    domain meet following requirements:
- Combine previous insights (if any): {existing_encyclopedia if
    existing_encyclopedia else "None"} with new insights.
- Extract your insights based on all client reasoning traces:
    {all_insights_text}. These traces are derived from solving specific
    problems (bottom-up approach)
- Use clusters of reasoning traces: {clusters_text if clusters_text else
    "None identified"} to help organize.
- Use relationships between traces:
    {json.dumps(profiling.get('relationships', []), indent=2) if
    isinstance(profiling, dict) else "None identified"} to help
    organization
- Your task is to extract multi-disciplinary, fundamental knowledge
    (top-down approach) which can be generalized to multi-domain
    problem-solving.
- The extracted insights should be able to DERIVE and GUIDE the use of
    the collected insights
- The extraced insights cannot be too general. They are not supposed to
    be knowledge which can be applied to any problem. They should be
    fundamental knowlege to particular several domains but specific.
- You should extra {proper_number} insights. Not too few. Not too
    many.Do not over simplified or too detailed.
- DO NOT over-merge insights.

Insights should have following properties:
1. **Extract Reusable Primitives**:
   - For EACH cluster, extract multiple fundamental insights capturing
    core essence and variations (DO NOT over-merge)
   - Identify cross-domain patterns that apply to multiple fields
   - Create reusable, composable primitives specific enough to be
    actionable

2. **Knolwedge to include**:
   - **Fundamental Level**: Core principles underlying multiple domains
   - **General Level**: Broad techniques for related problem types
   - **Cross-Domain**: Insights transferable beyond origin field

3. **Preserve While Generalizing**:
   - Create fundamental versions that can guide/derive original insights
   - Maintain important variations rather than collapsing into single
    insight

4. **Description Format:**
   Each description is a single string containing:
   - What the insight is and how it solves problems
   - When to use: problem types, conditions, triggers (be comprehensive
    and specific)

**Example 1 - Transformer Architecture:**
```

```
Input reasoning traces:
- reasoning_trace_VisionTransformerImageClassification: "I need to
    classify medical X-ray images into disease categories. CNNs aren't
    working well - they can't capture relationships between distant
    regions like how fluid in the lower right lung might relate to heart
    enlargement. Let me try Vision Transformer (ViT). I'll divide each
    X-ray into 16x16 patches - a 224x224 image gives 196 patches. Each
    patch becomes like a token in NLP. I flatten each to a 256-dim
    vector. Since transformers don't know spatial positions, I add
    position embeddings so the model knows patch [0,0] is top-left. I
    prepend a [CLS] token to gather global info. Feeding through 12
    transformer encoder layers - the self-attention lets every patch
    attend to every other patch directly, so patch [2,5] can look at
    patch [10,12] even though they're far apart spatially. This is
    exactly what I need! After 12 layers, I extract the [CLS] token and
    feed to MLP classifier. Training on 50k chest X-rays: 94.2%
    accuracy, beating ResNet-50's 89.3%. The attention maps show it's
    correctly attending to both lungs simultaneously for pneumonia
    detection, linking heart size to lung fluid - this cross-region
    reasoning is what CNNs miss. The key: treating image patches as
    tokens with self-attention enables global spatial reasoning."

- reasoning trace TransformerNextWordPrediction: "I'm building
    autocomplete for a text editor. The challenge: predict next word
    given arbitrary-length context. RNNs struggle with long sequences -
    the hidden state forgets earlier context. Let me use a transformer
    decoder. I tokenize 'The cat sat on the' using WordPiece -> tokens
    [254, 8901, 4523, 651, 278]. Convert each to 512-dim embedding and
    add positional encodings. Critical part: causal masking so the model
    can't cheat by seeing future words. When predicting token at
    position 4, it should only see 0-3. I implement lower-triangular
    attention mask. Processing through 6 decoder layers with masked
    self-attention. At position 4 ('the'), attention computes similarity
    between its query and keys of previous words. It attends strongly to
    'sat' (0.8) and 'on' (0.7), weakly to 'cat' (0.2). Using 12 heads
    helps - different heads capture different patterns: head-2 learns
    syntax (preposition+article), head-5 learns semantics
    (actions+objects), head-8 learns long-range dependencies. After
    final layer, project last hidden state through 50k-dim softmax. For
    'the': top predictions are 'mat' (0.73), 'floor' (0.12), 'rug'
    (0.08). Deployed in production - users accept top-3 suggestion 85\%
    of the time, reducing typing by 40\%. Transformer's self-attention
    captures context way better than RNN's sequential processing."

Output aggregated insight:
{{
  "insight_transformerArchitecture": "This fundamental neural network
    architecture applies across natural language processing, computer
    vision, time series analysis, graph neural networks, and multi-modal
    learning. This insight is essential for modern AI applications
    including language models, image processing, code generation, and
    scientific computing. When you need to capture relationships between
    all elements simultaneously (self-attention), you're working with
    sequences of variable length, you need parallel processing of
    sequences, or when the problem involves understanding context and
    relationships. Details: 1) Design input representation - convert
    your data into embeddings (token embeddings for text, patch
    embeddings for images, node embeddings for graphs), add positional
    encodings to preserve sequence information, and prepare input for
    transformer blocks 2) Create models with transformer blocks 3) Apply
    task-specific architecture - use encoder-only for understanding
    (BERT, ViT), decoder-only for generation (GPT), or encoder-decoder
    for translation"
}}
```

```
**Example 2 - Surface-Enhanced Raman Spectroscopy:**

Input reasoning traces:
- reasoning_trace_SERSMedicalDetectionR6G: "I need to detect cancer
    biomarkers in blood at incredibly low concentrations - 10^-12 M,
    like finding molecules in a swimming pool. ELISA only goes to 10^-9
    M, not sensitive enough for early diagnosis. Let me try SERS -
    Surface-Enhanced Raman Spectroscopy. Metal nanoparticles create huge
    EM field enhancements. I synthesize 60nm gold nanoparticles via
    citrate reduction. At 785nm laser, these have plasmon resonance
    amplifying local field by ~10^6. But I need selectivity too - can't
    detect everything. So I functionalize the gold with anti-PSA
    antibodies for prostate cancer. When I add patient serum, only PSA
    proteins bind. Here's the clever part: I add R6G (Rhodamine 6G)
    reporter molecules. R6G has enormous Raman cross-section and when it
    sits in nanogaps between gold particles, field enhancement shoots to
    10^8 or 10^10. Incubate 30 min for PSA binding, add R6G which sticks
    near bound PSA. Hit with 785nm laser at 5mW - I see characteristic
    R6G peaks at 1650, 1510, 1310 cm^-1. Peak intensity directly
    proportional to PSA amount. Integrate 60 sec for good SNR. Comparing
    to calibration: detecting PSA at 0.1 ng/mL - that's 10,000x more
    sensitive than ELISA! On clinical samples, detected prostate cancer
    3-6 months earlier than conventional tests. The breakthrough:
    combining selective antibody recognition with SERS amplification
    gives single-molecule sensitivity while maintaining specificity."

- reasoning trace SERSPollutantDetection: "Monitoring river water for
    pesticides. EPA limit for malathion is 0.1 ppb but standard
    chromatography needs 1 ppb minimum. I need 10x better for early
    warning. SERS might work. Instead of spherical particles, I'll
    fabricate silver nanorod arrays - sharp tips create hotter hotspots
    than spheres. Using oblique angle deposition: 80nm nanorods with 4:1
    aspect ratio on silicon. Gaps between rods only 5-10nm - perfect for
    trapping molecules. I calculate enhancement should hit 10^10 at
    532nm. Collect river water, filter through 0.2\mu m to remove debris
    and bacteria. Drop 50 \mu L onto nanorod substrate. During 5-min
    adsorption, pesticide molecules diffuse into nanogaps. Small gap
    means molecules guaranteed in enhancement zone (<10nm from metal).
    Rinse gently - removes interfering organics/salts but leaves
    adsorbed pesticides. Excite with 532nm at 2mW, matching silver
    plasmon peak. Even at 0.01 ppb malathion, clear peaks at 1440 cm^-1
    (P=S stretch), 1080 cm^-1 (P-O-C), 640 cm^-1 (C-S). Measuring 1440
    peak height vs calibration standards for quantification. Tested 50
    river sites, cross-validated against LC-MS: R^2=0.97 correlation.
    Best part: do this in field with portable Raman - no lab needed.
    Real-time monitoring at 10x below regulatory limits. The nanorod
    geometry is critical - those sharp tips and tight gaps push
    enhancement to 10^10."

Output aggregated insight:
{{
  "insight_surfaceEnhancedRamanSpectroscopy": "This powerful technique
    applies across analytical chemistry, materials science, biosensing,
    pharmaceutical analysis, environmental monitoring, and forensics.
    The technique achieves single-molecule sensitivity (10^6-10^11
    enhancement) while providing molecular structural information
    through vibrational fingerprints. When to use: When you need
    ultra-sensitive detection below conventional analytical limits, when
    you want label-free molecular identification, when analyzing trace
    contaminants or biomarkers, or when field-portable real-time
    analysis is required. Common steps: 1) Prepare SERS-active substrate
    - synthesize plasmonic nanostructures (gold/silver nanoparticles,
    nanorods, nanostars) optimizing particle size (20-100 nm), shape,
    and inter-particle spacing (1-10 nm gaps) to maximize
    electromagnetic field enhancement at laser wavelength 2)
```

```
    Functionalize substrate if needed - modify metallic surface with
    antibodies, aptamers, or molecular recognition elements for
    selective analyte binding and improved specificity 3) Prepare and
    apply sample - process sample (filter, dilute, concentrate as
    needed), deposit onto SERS substrate via drop-casting or
    flow-through, allow adsorption time for molecules to enter hot spots
    (<10 nm from metal surface) 4) Select laser parameters - choose
    wavelength matching plasmon resonance (532, 633, or 785 nm),
    optimize power (0.1-10 mW) to avoid sample damage while maximizing
    signal 5) Acquire SERS spectrum - collect Raman scattered light with
    appropriate integration time, record vibrational spectrum showing
    characteristic molecular peaks 6) Analyze spectral fingerprint -
    identify molecules by comparing peak positions to reference spectra,
    quantify concentration from peak intensities using calibration
    curves, assess molecular orientation from peak ratios 7) Validate
    and control quality - average multiple spots for reproducibility,
    use internal standards, verify with orthogonal methods, consider
    substrate heterogeneity and enhancement factor variations"
}}
```

Prompt 5 is used for determining whether a paper can be covered by insights from the library in Section 4.2.

Prompt 5: Definition of criteria for matching insights and the library

```
You are evaluating whether a research paper's proposed solutions are
    guided by or directly derived from insights in an encyclopedia.

Insights:
{insights_prompt}

Research Paper:
{paper_text}

Evaluation Criteria - An insight guides the paper ONLY IF ALL of the
    following are true:

1. CONCRETE METHODOLOGY USAGE: The insight's methodology or approach is
    concretely used in the paper's methods/approach section, not just
    theoretically relevant or mentioned in motivation.

2. METHODS SECTION PRESENCE: The insight must be related to how the
    paper actually implements its solution (methods, algorithms,
    techniques), not just in problem statement or related work.

3. COUNTERFACTUAL TEST: The paper's core contribution would
    fundamentally differ or fail without this insight. Ask: "If the
    authors didn't know this insight, could they still arrive at the
    same core solution?"

4. SPECIFICITY: The insight must specifically address a key challenge or
    component of the paper's solution, not just be generally applicable
    background knowledge.

Response Format:
- Respond ONLY in valid JSON with keys: guided (boolean),
    matched_insights (array of insight names)
- Set guided=true ONLY when at least one insight passes ALL criteria
    above
- Use only exact insight names from the Insights list above
```

D REPRESENTATIVE INSIGHTS

In this preliminary work, we show the three most representative insights from the library in Setting 1 (derived from ICLR 2023 top 25% papers) and Setting 2 (derived from ICLR 2024 Oral and Spotlight papers) here.

With Gemini 3 Pro Preview and ICLR 2023 top 25% papers, we obtain three insights: structured output prediction (Prompt 1), causal representation learning (Prompt 2), and physics-informed deep learning (Prompt 3), which provide interesting ideas and instructions on how to approach related research problems. These topics remain cutting-edge research questions today.

Insight 1: insight_structuredOutputPrediction (setting 1)

```
When to use: When the model output is not a simple class or scalar, but
    a complex structured object (e.g., a sequence, a tree, a matching, a
    code program) where dependencies between output elements must be
    modeled. Step-by-step: 1) Define the output space constraints and
    dependencies (e.g., grammar rules, bi-partite matching constraints).
    2) Use autoregressive models (like Transformers) to generate
    elements sequentially conditioned on history, or non-autoregressive
    models with iterative refinement. 3) Apply structural losses (e.g.,
    connectionist temporal classification, graph edit distance, optimal
    transport cost) rather than simple pointwise error. 4) Use inference
    techniques like beam search or constrained decoding to find the most
    likely valid structure. Key insights: This covers skills like
    hierarchical bipartite matching, permutation equivalent modeling,
    and program synthesis.
```

Insight 2: insight_causalRepresentationLearning (setting 1)

```
When to use: When building models that must robustly identify
    cause-and-effect relationships rather than spurious correlations,
    enabling interventions and counterfactual reasoning. Step-by-step:
    1) Model the system using a Structural Causal Model (SCM) or
    Directed Acyclic Graph (DAG). 2) Distinguish between invariant
    causal mechanisms (stable across environments) and spurious
    associations. 3) Use interventions (do-calculus) or environmental
    diversity to discover the causal graph. 4) Learn representations
    that satisfy conditional independence constraints implied by the
    causal structure (disentanglement). 5) Optimize for Invariant Risk
    Minimization (IRM) to find predictors that are optimal across all
    interventional distributions. Key insights: This unifies skills like
    causal mediation analysis, front-door prompting, and causal shift
    diagnosis.
```

Insight 3: insight_physicsInformedDeepLearning (setting 1)

```
When to use: When modeling physical systems governed by partial
    differential equations (PDEs), conservation laws, or geometric
    constraints (e.g., fluid dynamics, structural mechanics, climate
    modeling) where purely data-driven models violate physical validity
    or require excessive data. Use this to integrate physical knowledge
    directly into the learning process. Step-by-step: 1) Formulate the
    governing physical equations (residuals) of the system (e.g.,
    Navier-Stokes, Eikonal equation). 2) Design the neural network to
    output the physical state variables (velocity, pressure) given
    spatiotemporal coordinates. 3) Compute derivatives of the outputs
    with respect to inputs using automatic differentiation (Autograd) to
```

```
    evaluate the PDE residuals. 4) Construct a composite loss function:
    Data Loss (fitting observations) + Physics Loss (minimizing PDE
    residuals) + Boundary/Initial Condition Loss. 5) Optionally, use
    hard constraints (ansatz) or coordinate transformations to enforce
    boundary conditions by construction. Key insights: This guides
    specific skills like PDE-constrained latent dynamics, soft contact
    modeling, and neural operator learning.
```

In Setting 2, insights generated by Gemini 2.0 Flash are generally shorter than those by Gemini 3 Pro. However, even without knowledge of future papers, we still obtain interesting insights such as neuro-symbolic reasoning (Prompt 4) and parameter-efficient model adaptation (Prompt 5), which corresponds to what is currently known as parameter-efficient fine-tuning. It also produces the zero-to-one capability testing insight (Prompt 6), which can be adapted to systems research.

Insight 4: insight_neuroSymbolicReasoning (setting 2)

```
When to use: When a problem requires both the perceptual flexibility of
    neural networks (handling noisy, unstructured data) and the
    precision/interpretability of symbolic logic (handling rules, math,
    constraints). Step-by-step: 1) Decompose the system into a Neural
    Module (perception/pattern matching) and a Symbolic Module
    (reasoning/execution). 2) Design an interface layer: converting
    neural outputs into discrete symbols (concepts) or differentiable
    approximations (soft logic). 3) Use the Symbolic Module to execute
    logical rules, programs, or arithmetic on the extracted symbols. 4)
    Train end-to-end (via gradients through soft logic) or separately
    (via REINFORCE or supervision) to optimize both perception and
    reasoning accuracy. Key insights: This abstracts skills like
    neuro-symbolic prompt construction, logic constraint integration,
    and program synthesis.
```

Insight 5: insight_parameterEfficientModelAdaptation (setting 2)

```
When to use: When adapting large pre-trained 'foundation' models (LLMs,
    ViTs) to specific downstream tasks without the computational cost of
    full fine-tuning. Step-by-step: 1) Freeze the bulk of the
    pre-trained model parameters. 2) Introduce a small set of trainable
    parameters: Adapters (bottleneck layers between existing layers),
    Prompt/Prefix Tuning (learnable input tokens), or Low-Rank
    Adaptation (LoRA - decomposing weight updates into low-rank
    matrices). 3) Train only these new parameters on the task-specific
    dataset. 4) Optionally, use techniques like weight averaging (model
    soup) or dynamic routing to combine multiple adaptations. Key
    insights: This covers skills like minimalist adapter co-training,
    prompt engineering, and efficient fine-tuning protocols.
```

Insight 6: insight_zeroToOneCapabilityTesting (setting 2)

```
When to use: When assessing the functional value of a new technology or
    method to distinguish between efficiency gains and capability
    unlocks. This applies to product management, feature prioritization,
    and evaluating 'breakthrough' research. Step-by-step: 1) Ignore
    quantitative metrics (speed, cost, error rate) initially 2) Define
    the 'Capability Boundary' of the previous
    state-of-the-art---specifically what tasks it physically could not
    perform 3) Ask the binary question: 'Does the new solution enable a
```

```
    task that was previously impossible/intractable?' 4) If the answer
    is 'No' (it just does existing tasks better), classify as '1-to-N'
    (Incremental) 5) If the answer is 'Yes' (it opens a new action
    space), classify as '0-to-1' (Non-Incremental) 6) Prioritize 0-to-1
    capabilities for strategic innovation as they create new markets.
```