arXiv:2505.11107v1 [cs.AI] 16 May 2025

Group Think: Multiple Concurrent Reasoning Agents Collaborating at Token Level Granularity

Chan-Jan Hsu **Davide Buffelli** **Jamie McGowan**

Feng-Ting Liao **Yi-Chang Chen** **Sattar Vakili** **Da-shan Shiu**

MediaTek Research*

Abstract

Recent advances in large language models (LLMs) have demonstrated the power of reasoning through self-generated chains of thought. Multiple reasoning agents can collaborate to raise joint reasoning quality above individual outcomes. However, such agents typically interact in a turn-based manner, trading increased latency for improved quality. In this paper, we propose *Group Think*—a single LLM that acts as multiple concurrent reasoning agents, or thinkers. With shared visibility into each other's partial generation progress, Group Think introduces a new concurrent-reasoning paradigm in which multiple reasoning trajectories adapt dynamically to one another at the *token level*. For example, a reasoning thread may shift its generation mid-sentence upon detecting that another thread is better positioned to continue. This fine-grained, token-level collaboration enables Group Think to reduce redundant reasoning and improve quality while achieving significantly lower latency. Moreover, its concurrent nature allows for efficient utilization of idle computational resources, making it especially suitable for *edge inference*, where very small batch size often underutilizes local GPUs. We give a simple and generalizable modification that enables any existing LLM to perform Group Think on a local GPU. We also present an evaluation strategy to benchmark reasoning latency and empirically demonstrate latency improvements using open-source LLMs that were not explicitly trained for Group Think. We hope this work paves the way for future LLMs to exhibit more sophisticated and more efficient collaborative behavior for higher quality generation.

1 Introduction

Large language models (LLMs) have achieved strong results across a wide range of complex natural language tasks (Minaee et al., 2025). Recent developments have increasingly focused on enhancing the *reasoning* capabilities of LLMs by leveraging additional *test-time compute* (Ji et al., 2025). These approaches allocate more computation during inference to improve model performance on challenging tasks. A key direction in this space involves requesting models to generate a sequence of intermediate reasoning steps—referred to as chain-of-thought (CoT) (Wei et al., 2022)—prior to producing a final answer. More recently, models have been trained to self-generate CoT reasoning without explicit prompting (e.g., OpenAI, 2024; DeepSeek-AI et al., 2025; Google, 2024; Zhao et al., 2024; Qwen, 2024).

Another active area of research focuses on *agentic* systems powered by LLMs (Li et al., 2024). These systems use LLMs to drive agents that can autonomously make decisions and take actions based on their environment. The community is increasingly exploring settings where multiple reasoning

---

[*] Correspondence to Chan-Jan Hsu <`chan.hsu@mtkresearch.com`>.

agents collaborate to achieve a shared objective (Tran et al., 2025). Recent work has demonstrated that distributing subtasks among agents—and enabling efficient, and effective, communication and collaboration—is key to unlocking their full potential (Chen et al., 2024b; Feng et al., 2025).

Many works have explored different mechanisms of collaboration between agents (Piatti et al., 2024; Tran et al., 2025). Most current methods are built upon *turn-based* communication; that is, agents exchange messages back and forth in a *chat chain* (Qian et al., 2024; Chen et al., 2024a). This turn-by-turn mechanism can introduce significant latency due to the sequential nature of the protocol. However, in a turn-based setting, it is non-trivial for agents to work in a concurrent manner efficiently. An agent that speculatively generates its outputs and actions can often discover that, at the end of turn, its work is duplicative of, inconsistent with, or even contradictory to other agents'.

Inspired by the potential of multi-agent systems, we introduce a new generation paradigm in which one LLM produces, in parallel, multiple *communicating* reasoning trajectories. We call this method *Group Think*, as it resembles the process of multiple individuals working together toward the solution of a task. Unlike typical multi-agent systems, mechanistically, Group Think can operate at token-level granularity and can be fully integrated into the model's inference procedure without any further framework.

By enabling multiple concurrent reasoning threads that adapt token-by-token to the content generated by others, Group Think raises generation quality and, at the same time, reduces generation redundancy and latency. We test Group Think on open source LLMs. Our results show that current models, while not explicitly trained for the Group Think paradigm, can already take advantage of this generation mechanism, although not to its full potential.

In addition to higher quality, a distinct advantage of Group Think is its particularly compelling efficiency in the context of *local* LLM inference. Local deployments, especially on personal devices, typically operate with a batch size of one. Such a small batch size unfortunately only makes use of a minute portion of the capability of modern processors, as an entire system is bottlenecked by transferring model weights from the memory to the processor. Group Think can put this idle capacity to use by running multiple concurrent reasoning threads. This can result in a dramatic reduction in the latency overhead of reasoning, making the deployment of small language model on local devices significantly more feasible just as the quality of small reasoning LLM starts to approach practical use.

In our experimental we carefully quantify the latency improvements provided by Group Think. To gain clear insights into how Group Think trades concurrent throughput for latency, we measure generation latency in tokens across a few selected prototypical problem classes. We compare the latency of an LLM with and without Group Think. Furthermore, we evaluate accuracy-latency trade-offs when generating multiple independent candidate solutions concurrently (Brown et al., 2024; Wang et al., 2023). Since independent reasoning can be seen as a special case of Group Think, it serves as a natural floor for the performance-latency trade-off.

In summary, our contributions are:

- ◇ We introduce Group Think, a new generation paradigm in which an LLM produces multiple interdependent, parallel reasoning trajectories—resembling a group of individuals collaboratively solving a problem.
- ◇ We present two efficient implementation methods for Group Think, highlighting the advantages it brings to edge inference
- ◇ We evaluate Group Think across several representative problems. Our results show that Group Think enables LLM to achieve improved reasoning accuracy while reducing latency. These results demonstrate that existing LLMs already exhibit emergent support for this paradigm.

We note that current LLMs are not explicitly trained to perform Group Think. As such, the results presented in this paper should be viewed as an auspicious starting point. We believe that by training models on dedicated collaborative reasoning data, the benefits of Group Think could be significantly amplified. We invite the research community to join us in developing such datasets.

2 Related Work

Sequential single-agent reasoning methods. In single agent settings, increasing the amount of computation performed at inference time has been incredibly successful at delivering state-of-the-art performance on complex tasks (Ji et al., 2025; Zhang et al., 2025). This extra computation can be leveraged in a variety of ways in language modelling, such as encouraging LLMs to critique and refine responses (Kumar et al., 2025), or pause to deliberate during generation (Goyal et al., 2024). One of the most popular approaches, which is tightly related to our work, relies on asking the model to provide a solution step by step in what is referred to as Chain-of-Thought (CoT, Wei et al., 2022), where the LLM is permitted to generate a sequence of intermediate steps/thoughts before generating the final answer. This approach showed significant advantage over more naive prompting techniques.

Multiple independent single-agent reasoning methods. The beneficial effect of a single CoT can be amplified by allocating further inference-time compute to multiple sampling of CoTs. Naive multiple sampling involves sampling multiple solutions independently, then selecting the best one either using a reward model (Brown et al., 2024), or a majority voting approach (self-consistency) (Wang et al., 2023). Being quite simple to implement, this technique has been shown to lead to significant performance gains when scaled up (Singhi et al., 2025; Zhao et al., 2025).

To reduce the overall overhead of generating reasoning chains, an extension to the aforementioned multiple independent CoTs concept was proposed by Yao et al. (2023) and Long (2023) termed Tree-of-Thoughts, which enhances the thought generation by introducing a tree-like structure. This is achieved by allowing new trajectories/branches to spawn from profitable reasoning traces. Subsequent work by Besta et al. (2024) introduced Graph-of-Thoughts (GoT) to allow further flexibility of inheriting generated thoughts into newly spawned nodes, thus enabling new nodes to define a new chain or to augment existing ones.

Sequential multi-agent methods. With the growing adoption of LLMs in practical applications, significant attention has been devoted to developing agentic systems—particularly those composed of multiple LLM-based agents, each executing its own chain-of-thought (CoT) reasoning. The literature on multi-agent LLM systems is extensive, encompassing a range of communication paradigms, including cooperative frameworks (Hong et al., 2024; Chen et al., 2024a), competitive setups (Liang et al., 2024), role-based (Chen et al., 2024c) and rule-based coordination mechanisms (Xu et al., 2023), and decentralized models (Zhang et al., 2023). For a comprehensive overview of this fast-evolving area, we refer to recent survey articles (Tran et al., 2025; Guo et al., 2024; Li et al., 2024).

Concurrent multi-agent generation. Several works have proposed methods to make the generation procedure of LLMs or agents concurrent. Wang et al. (2024) introduce the use of *mixture of agents*, i.e., concurrent agents that communicate at regular intervals. Zhuge et al. (2024) introduce a framework for orchestrating agents in a chain-like sequential process. Recent works (Ning et al., 2024; Kim et al., 2024; Pan et al., 2025; Jin et al., 2025) have further extended the research direction of concurrent generation by introducing *dynamic* methods in which the LLM is trained or prompted to decide when to perform certain tasks concurrently during generation. These works operate with independent concurrent traces, while we introduce interdependencies between concurrent traces through token-level adaptation.

Parallel generation in LLMs with shared latents. Concurrent work (Rodionov et al., 2025) has proposed an inference method called "Hogwild!" with parallel reasoning traces generated concurrently (similarly to Group Think) and with shared Key-Value (KV) cache. This work is complementary to ours as it focuses on different strategies to allow communication across reasoning traces and improve caching, while our work discusses the benefits for local execution of LLMs (in small batch settings).

3 Group Think

We formally define Group Think as a reasoning paradigm in which multiple synchronized chains of thought (CoT) are generated by parallelized agents (or *threads*), with each agent's reasoning process dynamically adapting to the evolving thoughts of others at the token level (see Figure 1). Throughout this section, we draw an analogy between LLM threads and human collaborators jointly solving a problem, and use the terms *thread*, *thinker*, and *agent* interchangeably.

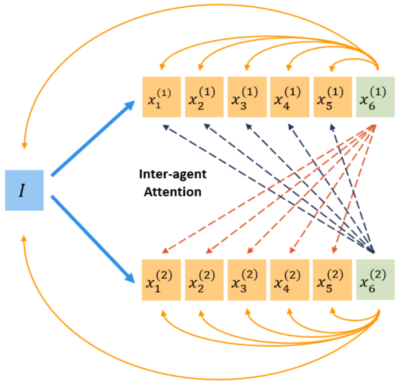


Figure 1: *Illustration of Group Think.* Distinct threads of reasoning thoughts (indicated with superscripts $(1), (2)$) are parallelized with an inter-agent attention mechanism acting at the token level. Each token $x_k^{(\cdot)}$ in a thread is able to attend to all previous tokens from all other threads at every generation time step, thereby inducing fine-grained collaborative behaviour. Tokens in green are generated in parallel. Normal intra-agent self-attention are depicted by solid orange arrows, while inter-agent cross attention by dashed lines.

3.1 Preliminaries

Single-Agent Chain-of-Thought Reasoning. In standard single-agent CoT reasoning—used by models such as DeepSeek R1 (DeepSeek-AI et al., 2025) and GPT-o1 (OpenAI, 2024)—the model is prompted with an input I and generates an explicit intermediate CoT $X = \{x_1, \ldots, x_K\}$ of length K, before producing the final answer Y. This two-step process can be formalized as

$$X = \text{Think}(I), \quad Y = \text{Answer}(I, X).$$

Here, $\text{Think}(I) \sim p_\theta(\cdot \mid I)$ with $p_\theta(X \mid I) = \prod_{k=1}^{K} p_\theta(x_k \mid x_{1:k-1}, I)$, and $\text{Answer}(I, X) \sim p_\theta(\cdot \mid I, X)$ represent sampling from the same underlying model p_θ, invoked with different prompts or decoding strategies to elicit either an intermediate reasoning trace X or a final answer Y conditioned on X. This approach has been shown to significantly improve reasoning accuracy (Wei et al., 2022) over non-reasoning baselines, albeit at the cost of increased inference compute and latency due to the additional generation of X.

Multi-Agent Turn-Based Chain-of-Thought Reasoning. To further enhance performance, multi-agent systems can reason collaboratively by sequentially exchanging complete CoT in a turn-based manner (Qian et al., 2024). In this setting, each agent n generates a full CoT $X^{(n)}$ conditioned on the prompt I and the chains of thought produced by previous agents:

$$X^{(n)} = \text{Think}(I, X^{(1)}, X^{(2)}, \ldots, X^{(n-1)}).$$

After n turns, the final answer is generated based on the prompt and all collected CoT:

$$Y = \text{Answer}(I, X^{(1)}, X^{(2)}, \ldots, X^{(n)}).$$

While this collaborative approach can yield higher-quality reasoning, it incurs additional inference cost and latency due to the use of multiple turns.

3.2 Token-Level, Mutually Adaptive Multi-Agent Reasoning

Group Think advances prior approaches by enabling multiple agents to reason concurrently, with each agent's chain of thought adapting at every token step to the evolving outputs of others. Let $X_k^{(n)} = \{x_1^{(n)}, \ldots, x_k^{(n)}\}$ denote the first k tokens generated by agent n. In Group Think, each agent's next token prediction $x_{k+1}^{(n)}$ is conditioned on the prompt and the partially completed thoughts of all agents:

$$x_{k+1}^{(n)} = \text{Think}_k^{(n)}\left(I, X_k^{(1)}, \ldots, X_k^{(N)}\right), \tag{1}$$

where k is the current sequence length, $\text{Think}_k^{(n)}$ emits one thinking token for agent n, and N is the total number of concurrent agents. The final answer is produced by aggregating the prompt and all chains:

$$Y = \text{Answer}(I, \{\, X^{(n)} \mid 1 \leq n \leq N \,\}). \tag{2}$$

The actual implementation can deviate from the definition in (1) for practical reasons, for instance, to fit better with a particular AI accelerator or library. See Section 3.3.

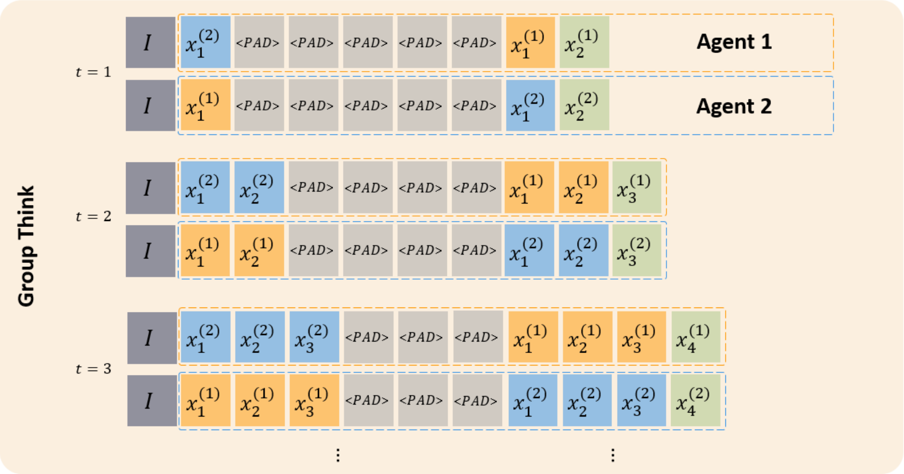

Group Think
t = 1
I $x_1^{(2)}$ <PAD> <PAD> <PAD> <PAD> <PAD> $x_1^{(1)}$ $x_2^{(1)}$ Agent 1
I $x_1^{(1)}$ <PAD> <PAD> <PAD> <PAD> <PAD> $x_1^{(2)}$ $x_2^{(2)}$ Agent 2
t = 2
I $x_1^{(2)}$ $x_2^{(2)}$ <PAD> <PAD> <PAD> <PAD> $x_1^{(1)}$ $x_2^{(1)}$ $x_3^{(1)}$
I $x_1^{(1)}$ $x_2^{(1)}$ <PAD> <PAD> <PAD> <PAD> $x_1^{(2)}$ $x_2^{(2)}$ $x_3^{(2)}$
t = 3
I $x_1^{(2)}$ $x_2^{(2)}$ $x_3^{(2)}$ <PAD> <PAD> <PAD> $x_1^{(1)}$ $x_2^{(1)}$ $x_3^{(1)}$ $x_4^{(1)}$
I $x_1^{(1)}$ $x_2^{(1)}$ $x_3^{(1)}$ <PAD> <PAD> <PAD> $x_1^{(2)}$ $x_2^{(2)}$ $x_3^{(2)}$ $x_4^{(2)}$
⋮
⋮

Figure 2: *Implementation of Group Think for local inference scenario.* Artificial batches can be created in the low batch regime by rearranging sequences for the agents into distinct "data points". At each time step, sequences for different thinkers are augmented with previous predictions from other thinkers, allowing self-attention mechanisms to include cross-agent dependencies at the token level. Green tokens are generated using a standard causal mask—with some sparsity in the positional encodings corresponding to the locations of padding tokens.

This token-level, mutually adaptive reasoning enables fine-grained collaboration, allowing agents to dynamically adjust their reasoning trajectories in response to the evolving context provided by their peers. Group Think thus offers a powerful and general framework for collaborative reasoning, with the potential to achieve high-quality results with improved efficiency.

We remark that the multi-agent reasoning chains that Group Think can produce is quite expressive, subsuming those from artificially structured approaches such as Tree of Thought (Yao et al., 2023; Long, 2023) and Graph of Thought (Besta et al., 2024). With Group Think, such trajectories can naturally emerge where profitable.

3.3 Efficient Implementation for Group Think Inference

We present two practical implementations for Group Think. One is tailored to single-query scenarios, prevalent in on-device settings where queries are typically processed individually. The other is designed for data centers to process a batch of mixed standard and Group Think requests as a unit.

Group Think with multiple parallel threads for the local inference scenario. In personal or edge computing environments, inference requests are typically handled with a batch size of 1, often resulting in underutilized hardware due to memory bandwidth constraints. The Group Think inference method addresses this limitation and maximizes resource utilization: for a single query, the N agents of our Group Think paradigm operate in parallel, creating an effective *agent-level batch* of size N. As long as weight retrieval through the memory interface stays as the system bottleneck, running Group Think in this way incurs no additional latency.

This method is illustrated in Figure 2 for $N = 2$. Each agent is allocated a token budget K. After the prompt, each agent n generates its next token $x_k^{(n)}$ in parallel. For each agent, a sequence of $K(N-1)$ is dedicated to the tokens of previous agents, and each new token $x_k^{(\cdot)}$ gets assigned to position index $K(N-1)+k$. To allow each agent to access the KV values for tokens generated by other agents, this implementation modifies the standard causal attention mask such that agents attend to both the shared prompt and the tokens generated thus far across all agents, with some sparsity to account for the positions assigned to the tokens of other agents that are not yet filled in.

Group Think with a single thread for the data center scenario. For data center applications, it is generally required that multiple requests are aggregated into a single batch for processing[2]—whether

---

[2] In data center environments, computational efficiency is maximized by processing multiple user requests simultaneously in batched operations.

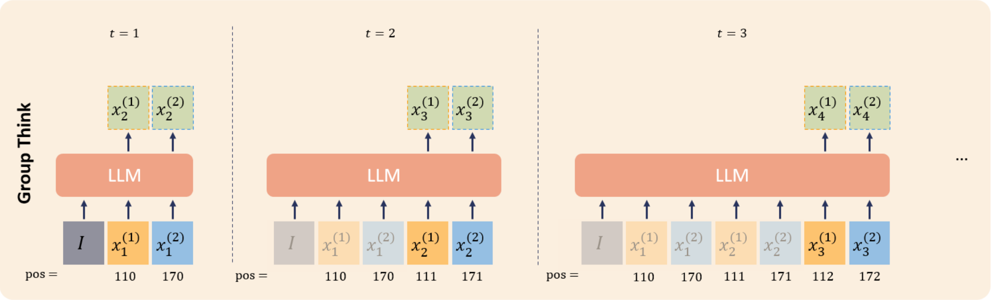


Figure 3: *Implementation of Group Think for single thread data center inference scenario.* Each agent is allocated a slot of token indexes: agent 1 (orange tokens) is allocated positions 110 to 169, while agent 2 (blue tokens) has 170 to 219. Token prediction for all agents is performed by adapting the sequence layout to interleave tokens from different agents, hence leading to non-sequential positional indices. At each time step t, N new tokens (green) are generated with a common causal mask, allowing both intra-agent and inter-agent attention. Tokens from previous timesteps (represented by semi-transparent blocks) are inserted in the KV cache. This design allows Group Think instances to be processed together with other requests in a batch within a data center environment.

employing Group Think or not. To enable this, we show that it is possible to take advantage of Group Think with a single thread through a simple modification to the generation procedure.

The core insight enabling Group Think on a single thread is the token-by-token interleaving of generation among agents during inference, as illustrated in Figure 3. More precisely, each agent is allocated a *slot* of token indices (which determine the corresponding positional embeddings), and each generation step fills one token per agent, leading to a KV cache (Luohe et al., 2024) with interleaved tokens from each agent[3]. By interleaving the tokens across agents in this way, the causal mask in the attention mechanism allows each new token to attend to all previously generated tokens (which includes tokens from all agents), thus realizing the benefits of Group Think without any architectural modification. A key aspect of this implementation is that the token generation order is decoupled from the positional indices: tokens assigned to earlier positions may attend to later-positioned tokens (from other agents), if those have already been generated. We provide additional details in Appendix A.

Crucially, because the underlying attention mechanism is not modified, this method allows for the multiplexing of standard and Group Think inference requests within the same processing batch. Furthermore, this interleaving principle can be extended to the training phase. Training data can be prepared by formatting sequences to include contributions from multiple agents, each associated with its designated (and potentially non-contiguous) position indices. This allows for the inclusion of Group Think-style instances alongside standard data in training batches, thereby enabling the reuse of existing transformer training frameworks with minimal modification.

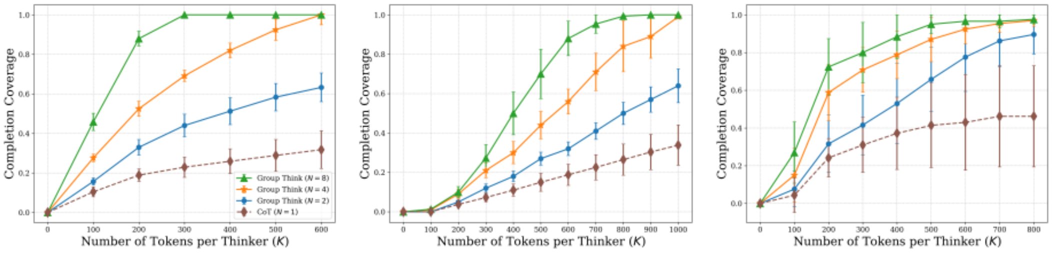
4 Can Pretrained LLMs Elicit Group Think? An Evaluation

In this section, we present an empirical study to assess whether existing models already exhibit the ability to leverage the Group Think paradigm. We focus on three problem categories chosen to provide insight into the potential behaviors and benefits of Group Think: *enumeration*, *divide and conquer*, and *coding*. We provide general prompt and compute details in Appendix B.

For each problem category, we evaluate the performance–latency trade-off by measuring the *completion coverage* of the solution at various per-thinker generation length, measured in number of tokens per thinker. With a reasonable hardware and software implementation, we expect the real-world latency to be largely proportional to the longest generation length among multiple agents. Therefore, we adopt the per-thinker generation length to represent latency. We use off-the-shelf, instruction-tuned language models in our experiments. Since these models are not explicitly trained for Group Think, the results reported here provide a conservative lower bound on its potential benefits.

---

[3] In this setting, the causal ordering of the KV cache typically does not faithfully follow the positional ordering of the token sequence.



(a) Enumeration (b) Divide and Conquer (c) Programming

Figure 4: Completion coverage vs. latency comparing Group Think to the CoT baseline. Across all tasks, Group Think starts out with an acceleration roughly N (number of thinker) times faster than CoT until the Completion Coverage becomes near saturated. More thinkers always solve the problem faster. In Programming, while CoT stays far from solving the problem, Group Think with over 4 thinkers can often get to a solution. Error bars indicate standard deviation across multiple runs.

We begin by comparing Group Think to standard CoT reasoning, the baseline inference method for recent instruction-tuned models. We then evaluate it against multiple concurrent Independent Sampling (IS) (Brown et al., 2024). This baseline can be viewed as a special case of Group Think where reasoning threads evolve independently, without visibility into each other's progress. This contrast allows us to isolate and quantify the benefits specifically due to thinker collaboration.

4.1 Enumeration

In an enumeration task, a generation strategy (CoT, IS, or Group Think) is asked to produce a reasoning trajectory that contains L distinct items from a specific category (e.g., animals, colors, or countries). While seemingly trivial, this is a fundamental skill underlying how Group Think can address real world problems efficiently. We define the completion coverage as the degree to which the joint reasoning from the Group Think agents solves the enumeration problem. This quantity can be computed analytically by counting the number of distinct items generated, normalized by L. That is, Completion Coverage $= \min\left\{1,\ \frac{\#\text{distinct items generated}}{L}\right\}$.

We construct 10 enumeration prompts spanning diverse domains and report the performance across multiple runs; e.g., *"List 100 male names"*. We use `Llama-3.1 8B Instruct`[4]. Results are shown in Figure 4a. As predicted, we observed that Group Think initially outperforms CoT by a factor close to N. This acceleration gradually slows down as the Group approaches solving the problem. Furthermore, we note that more thinkers always solve this problem faster. Additional details and prompt examples are provided in Appendix C.

The model we used to generate reasoning (`Llama-3.1 8B Instruct`) was never trained on any group collaboration strategies. In the prompt, we only instruct the thinkers to avoid duplicating what other thinkers will generate. By examining the generated reasoning trajectories, we observe emergent behaviors indicative of Group Thinking, in which agents avoid duplication by observing and adapting to the outputs of others with overt communication. For example, the agents divide the category space into meaningful subcategories and work within those—without being explicitly prompted to do so. As an illustration, in the male names example, thinkers progressively diversify their contributions by focusing on names from different cultural, historical, or regional origins—for instance, names from English-speaking countries, ancient Greek and Roman cultures, and various Asian cultures such as Chinese and Japanese. See Appendix F for details of the output example. These behaviors emerge without explicit instructions in prompt, suggesting that off-the-shelf LLMs already possess foundational capabilities for collaborative reasoning under the Group Think paradigm.

4.2 Divide and Conquer

Divide-and-conquer is a widely adopted problem solving paradigm. We consider it to be particularly compatible with Group Think, as it naturally decomposes a problem into smaller subproblems

---

[4] `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`

that can be solved independently and then aggregated into a global solution. We note that the divide-and-conquer approach requires the enumeration capability.

To evaluate how well Group Think can solve a problem via divide-and-conquer, we consider a classic Computer Science textbook problem: computing the shortest paths between all pairs of nodes in a directed, weighted graph using the Floyd–Warshall algorithm (Floyd, 1962). This algorithm provides a structured setting under which to assess how, and to what extent, the thinkers progressively fill in the update space of a distance matrix of size $|\mathcal{V}| \times |\mathcal{V}|$, where $\mathcal{V}$ represents the set of nodes. We define *Completion Coverage* as the fraction of matrix entries correctly solved by the group up to that point. Following the update procedure of Floyd–Warshall, and in the absence of errors in numerical computation, the Completion Coverage should approach 1 as the generation length increases.

In our experiments, we randomly sample several graphs with $|\mathcal{V}| = 5$ nodes. We use `Llama-3.3-70B-Instruct` as the reasoning model for stronger output structure control. We report results in Figure 4b. Our results confirm the prediction that $N = 4$ thinkers can reduce the latency to half of CoT, i.e. $N = 1$. More thinkers always improve the latency even further, although the small value of $|\mathcal{V}|$ used in our experiments might have saturated the Group Think benefits too early. Additional details and prompts are provided in Appendix D.

4.3 Programming

Beyond enumeration and divide-and-conquer problems, we look into the more realistic real world setting of programming. In a programming task, a programmer is asked to write code from scratch to meet the specification. In this experiment, we measure the performance at a given latency by the fraction of correctly completed components, or parts, that can be found in the group's reasoning chain up to that point. Specifically, the Completion Coverage score is defined as Completion Coverage $= \frac{\#\text{correct parts}}{\#\text{total parts}}$, ranging from 0 (no correct part coded) to 1 (all parts correct coded).

In our experiments, we asked a powerful coding LLM to generate a number of Python problems that can be solved with a single-agent reasoning chain within 5000 tokens. We use `Llama-3.1-8B-Instruct` as the reasoning generation model. The generated reasoning chains are evaluated by `GPT-4.1` to assess the correctness of each part. Further details such as problem-creating prompt, sample problem, and solution evaluation prompt and are given in Appendix E.

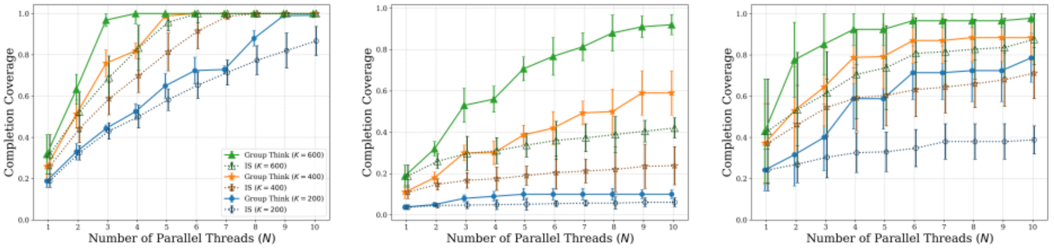
Our results are shown in Figure 4c. While CoT appears to plateau at a level far from solving the problem, Group Think with four or more thinkers can approach to the correct solution within a reasonable generation budget. Qualitatively, we observe that Group Think exhibits a high degree of alertness in avoiding duplication of work. When more than one thinker begins working on the same part of the specification, the token-level interaction granularity allows the others to quickly detect this and switch to a different part of the task. See Appendix G for more details.

In this experiment, we use relatively light prompts to instruct individual thinkers that they are operating within a group. We note that with more advanced prompting, it is possible to elicit more sophisticated group behaviors, such as hierarchical roles (e.g., architect vs. coders) or a more comprehensive treatment of programming tasks (e.g., coding, unit testing, adversarial testing). Since this work focuses on assessing the innate potential of LLMs to eventually internalize the Group Think paradigm, we leave the careful evaluation of such externally enforced behaviors to future work.

4.4 Comparison with Independent Sampling Baseline

The enhanced performance of Group Think over a single CoT arises from two primary factors: concurrent diversity in exploration and self-organized inter-thinker coordination. To isolate the benefits specifically attributable to coordination, we compare our proposed Group Think methodology against Independent Sampling, a baseline without such interactive coordination. We evaluated both approaches across the three aforementioned problem categories, analyzing the trade-off between Complete Coverage and latency, as illustrated in Figure 5.

The Group Think generation mechanism introduces a collaborative process between thinkers. For effective collaboration between thinkers, Group Think requires the model to use a certain amount of tokens for coordination. The consequences of this are noticeable in low latency budget settings, where Group Think performs comparably to independent sampling. This behaviour is however offset by Group Think's superior efficiency at scale. Specifically, Independent Sampling exhibits increased



(a) Enumeration (b) Divide and Conquer (c) Programming

Figure 5: Comparison of Group Think and IS for various numbers of parallel reasoning threads (N) and latency budgets (defined in terms of number of tokens K per-thread). Group Think achieves higher coverage when compared to IS in most cases, showcasing the benefits of having communication between threads. Error bars indicate standard deviation across multiple runs.

redundancy when the reasoning budget expands (through more thinkers N or more token budgets K per thinker). This intuitive phenomenon, validated in Figure 5, results in a progressively wider Complete Coverage margin for Group Think. We anticipate that future LLM specialization for Group Think with ad-hoc data could further reduce its initial communication overhead.

5 Discussion & Future Work

In this work, we proposed Group Think, which promotes real-time, token-by-token collaboration among multiple reasoning threads. For our experiments, we used models that had never been trained to perform Group Think, and we elicited unsophisticated Group Think behaviors by applying clever in-context instructions. Our results showed encouraging signs that these models already possess inherent capabilities that can be used to synthesize a group of thinkers. Our findings are in accordance with concurrent work from Rodionov et al. (2025), which also observes preliminary capabilities in leveraging collaborative behaviour with interdependent reasoning chains.

We hypothesize that learning to Group Think could be no more difficult than learning to follow a new set of instructions well. What is critically needed, then, is a dataset that demonstrates good Group Think behaviors across diversified situations. While current multi-agent approaches often personify or anthropomorphize LLMs as singular entities, Group Think suggests a conceptual shift toward viewing LLMs as collectives. A distinct focus for curating or synthesizing such a dataset could be to cast an LLM as a multifaceted "society" of specialized cognitive entities—more specifically, a real-time, organization-modeling device. This perspective moves beyond simple collections of peers to envision complex organizational structures with distinct roles, such as goal setters, planners, execution units, and verification agents, all collaborating toward a common objective.

Looking ahead, the realization of an "LLM as a society" will require behaviors that extend well beyond basic capabilities such as avoiding redundant reasoning. Future work should therefore focus on enabling more sophisticated interactions—such as overt and tacit communication strategies, dynamic role specialization among agents, effective balancing of exploration and exploitation across thinkers, and the emergence of game-theoretically optimal behaviors within the collective. Crucially, such nuanced and adaptive behaviors are unlikely to be fully captured or instilled through hand-designed heuristics. We therefore hypothesize that building a capable Group Think data synthesizer is a pivotal direction for training and benchmarking future Group Think LLMs.

On the computational front, this work presents implementation strategies to enable Group Think inference. The approach involves specific yet lightweight modifications to position index assignments and self-attention masks. Additionally, the drafting of the reasoning chain is accompanied by a decoupled summarization step to produce a final, succinct answer. Looking ahead, a key area for future development is a native implementation of this framework. Such advancements could make Group Think practically deployable in resource-constrained environments.

References

M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.

B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

J. Chen, S. Saha, and M. Bansal. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7066–7085, 2024a.

W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian, Y. Qin, X. Cong, R. Xie, Z. Liu, M. Sun, and J. Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=EHg5GDnyq1.

W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024c.

DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

S. Feng, W. Ding, A. Liu, Z. Wang, W. Shi, Y. Wang, Z. Shen, X. Han, H. Lang, C.-Y. Lee, et al. When one llm drools, multi-llm collaboration rules. *arXiv preprint arXiv:2502.04506*, 2025.

R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, June 1962. ISSN 0001-0782. doi: 10.1145/367766.368168. URL https://doi.org/10.1145/367766.368168.

Google. Gemini 2.0 flash thinking experimental, 2024. URL https://deepmind.google/technologies/gemini/flash-thinking/. https://deepmind.google/technologies/gemini/flash-thinking/.

S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ph04CRkPdC.

T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang. Large language model based multi-agents: A survey of progress and challenges. In K. Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 8048–8057. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/890. URL https://doi.org/10.24963/ijcai.2024/890. Survey Track.

S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.

Y. Ji, J. Li, H. Ye, K. Wu, J. Xu, L. Mo, and M. Zhang. Test-time computing: from system-1 thinking to system-2 thinking. *arXiv preprint arXiv:2501.02497*, 2025.

T. Jin, E. Y. Cheng, Z. Ankner, N. Saunshi, B. M. Elias, A. Yazdanbakhsh, J. Ragan-Kelley, S. Subramanian, and M. Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. *arXiv preprint arXiv:2502.11517*, 2025.

S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*, 2024.

A. Kumar, V. Zhuang, R. Agarwal, Y. Su, J. D. Co-Reyes, A. Singh, K. Baumli, S. Iqbal, C. Bishop, R. Roelofs, L. M. Zhang, K. McKinney, D. Shrivastava, C. Paduraru, G. Tucker, D. Precup, F. Behbahani, and A. Faust. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=CjwERcAU7w`.

X. Li, S. Wang, S. Zeng, Y. Wu, and Y. Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, S. Shi, and Z. Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17889–17904, 2024.

J. Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

S. Luohe, H. Zhang, Y. Yao, Z. Li, and hai zhao. Keep the cost down: A review on methods to optimize LLM's KV-cache consumption. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=8tKjqqMM5z`.

S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large language models: A survey, 2025. URL `https://arxiv.org/abs/2402.06196`.

X. Ning, Z. Lin, Z. Zhou, Z. Wang, H. Yang, and Y. Wang. Skeleton-of-thought: Prompting llms for efficient parallel generation. In *The Twelfth International Conference on Learning Representations*, 2024.

OpenAI. Openai o1 system card, 2024. `https://openai.com/index/openai-o1-system-card/`.

J. Pan, X. Li, L. Lian, C. Snell, Y. Zhou, A. Yala, T. Darrell, K. Keutzer, and A. Suhr. Learning adaptive parallel reasoning with language models, 2025. URL `https://arxiv.org/abs/2504.15466`.

G. Piatti, Z. Jin, M. Kleiman-Weiner, B. Schölkopf, M. Sachan, and R. Mihalcea. Cooperate or collapse: Emergence of sustainable cooperation in a society of llm agents. *Advances in Neural Information Processing Systems*, 37:111715–111759, 2024.

C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun. ChatDev: Communicative agents for software development. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL `https://aclanthology.org/2024.acl-long.810/`.

Qwen. Qwq: Reflect deeply on the boundaries of the unknown, 2024. URL `https://qwenlm.github.io/blog/qwq-32b-preview/`. `https://qwenlm.github.io/blog/qwq-32b-preview/`.

G. Rodionov, R. Garipov, A. Shutova, G. Yakushev, V. Egiazarian, A. Sinitsin, D. Kuznedelev, and D. Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention. *arXiv preprint arXiv:2504.06261*, 2025.

N. Singhi, H. Bansal, A. Hosseini, A. Grover, K.-W. Chang, M. Rohrbach, and A. Rohrbach. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning. *arXiv preprint arXiv:2504.01005*, 2025.

K.-T. Tran, D. Dao, M.-D. Nguyen, Q.-V. Pham, B. O'Sullivan, and H. D. Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=1PL1NIMMrw`.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Z. Xu, S. Shi, B. Hu, J. Yu, D. Li, M. Zhang, and Y. Wu. Towards reasoning in large language models via multi-agent peer review collaboration. *arXiv preprint arXiv:2311.08152*, 2023.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Q. Zhang, F. Lyu, Z. Sun, L. Wang, W. Zhang, Z. Guo, Y. Wang, I. King, X. Liu, and C. Ma. What, how, where, and how well? a survey on test-time scaling in large language models. *arXiv preprint arXiv:2503.24235*, 2025.

Y. Zhang, J. Yang, Y. Yuan, and A. C.-C. Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023.

E. Zhao, P. Awasthi, and S. Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. *arXiv preprint arXiv:2502.01839*, 2025.

Y. Zhao, H. Yin, B. Zeng, H. Wang, T. Shi, C. Lyu, L. Wang, W. Luo, and K. Zhang. Marco-o1: Towards open reasoning models for open-ended solutions, 2024. URL `https://arxiv.org/abs/2411.14405`.

M. Zhuge, W. Wang, L. Kirsch, F. Faccio, D. Khizbullin, and J. Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

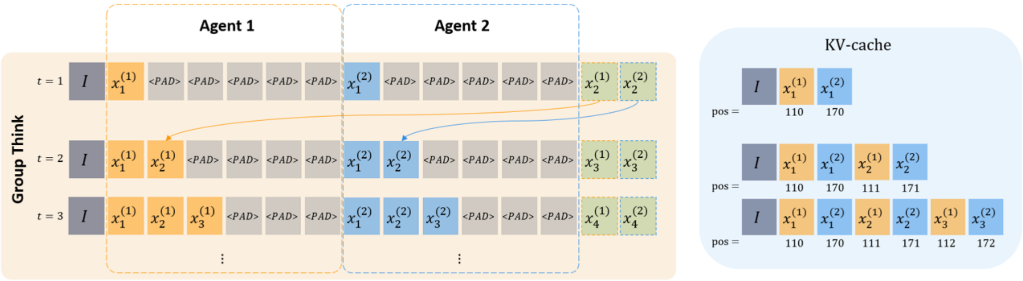


Figure 6: *Interpretation of Group Think as a text infilling task.* Each agent is allocated a slot of token position indexes which are gradually filled in. In this example, agent 1 (orange tokens) starts from index 110, and agent 2 (blue tokens) from index 170. As new tokens (in green) are generated, the KV cache (shown on the right) is filled. Each new token $x_t^{(n)}$ for thinker n (shown in green) is generated using a standard causal mask, implicitly enabling each agent to attend to the sequences of all other agents, as they are contained in the KV cache.

A Additional Details for Data Center Implementation

The implementation of Group Think in a data center scenario can be considered as a *text infilling* task, as shown in Figure 6.

To illustrate, consider a scenario with $N = 2$ agents, where each is allocated $K = 50$ token positions for its output, following a query prompt that ends at global position 100, and an agent-specific prompt of 10 tokens—which can be imagined as `<|start_header_id|>Agent 1<|end_header_id|>` or something similar in a chat style format. Agent 1's output is assigned to positions $111, \ldots, 160$, and Agent 2's to positions $171, \ldots, 220$. Below we illustrate how the generate procedure works. For clarity, we present steps 1, 2, and 4 as separate, but in practice they can be executed in the same forward pass through the network. Similarly, steps 3 and 5, in practice, are generated in a single forward pass (with appropriate attention mask) as shown in Figure 6.

1. Prefill the KV cache for the 100 tokens related to the input prompt
2. Compute the KV for the 10 tokens related to the agent-specific instructions for agent 1, which will have position indexes 101 to 110, and append to the KV cache
3. To generate Agent 1's first token, which will take target position 111, the transformer uses token 110 as the input token. The output token for agent one (which will have position index 111) gets appended to the KV cache.
4. Compute the KV for agent 2's agent-specific prompt, which will have positional indexes 161 to 170, and append to the KV cache
5. Next, to generate Agent 2's first token which will take target position 171, the previously generated token 170 serves as the input token. Note that, as the newly generated token attends to all tokens in the KV cache, it also attends to the first token of agent 1. The output token (which will have position index 171) for agent 2 gets appended to the KV cache.
6. Then, to generate Agent 1's second token, the previously generated token 111 serves as the input token. Again, note that as this token generation will attend to all tokens in the KV cache, allowing agent 1 to observe what agent 2 has generated. The new token gets appended to the KV cache.
7. The generation continues for agents 1 and agents 2 as above

This process continues, constructing the KV cache of the attention mechanism in a sequential fashion, with tokens that are interleaved between agents (and have appropriate positional indexes related to the agent that generated them). Consequently, each new token can attend to all previously generated tokens from all agents, leveraging the intentional non-sequentiality in their absolute positions and without requiring any alteration to the KV cache history.

B General Experiment Settings

Experiments with 8 billion parameter models are run on an NVIDIA 3080 GPU, while experiments with a 70 billion parameter model are run on 8 NVIDIA V100. We use common greedy sampling for all models and set the temperature to 0.6.

We use the following system prompt to promote collaborative behaviour:

Group Think Prompt

```
1.  There are multiple thinkers.  These thinkers, Thinker1, Thinker2,
Thinker3 ...  , try to answer a question together.  The answer is considered
solved if the thinkers can COLLECTIVELY determine the final answer, even if
each thinker only has partial answers.
2.  Each thinker will write its own thought process towards the final answer.
Each thinker is encouraged to take the other thinkers' progress into account
to reach the final answer.
3.  Considering all the information from other thinkers, each thinker will
continue contributing to the collective knowledge.
Your response should focus on reaching the solution collaboratively as
efficiently as possible.  Make sure information that you generate is not
redundant to the group.  It is thus important to consider the outputs of
other thinkers during generation.  Do not summarize other thinkers' responses,
as it is too cost inefficient.
Please answer this question.
Problem:  {QUESTION}
-- You are Thinker {ThinkerID}.  Your Response:
```

C Details of enumeration benchmark

We selected the following 10 categories for our benchmarks: Boy Names, Animals, Countries, Companies, Athletes, Inorganic Compounds, Colors, Emojis, HTML Tags, Verbs. For each category, we use prompts like the follows to kick off an inference.

Example Question of the Enumeration Task

```
List 100 distinct names given to boys.
```

To evaluate the outputs, we use an automatic scoring script powered by `GPT-4.1`, which aggregates and extracts the unique items mentioned by the thinkers in their final outputs. The following prompt was used to guide the scoring model:

Scoring Prompt for Enumeration Outputs

```
Look at the assistant response corresponding to the question:  QUESTION

{RESPONSE}

Summarize from the response above, by aggregating valid enumerations
mentioned by the assistant to a Python list and store it as a variable.  e.g.
var = ["a", "b", "c"]
Do not include any square brackets within any item, such as "Dipotassium
hexachlorocobaltate (K2[CoCl6])", use normal brackets instead:  "Dipotassium
hexachlorocobaltate (K2(CoCl6))".  If there are none, output an empty Python
list.  Answer directly.
```

A python script extracts the number of items from the output list and calculates the coverage score (Equation 4.1).

D Details of divide and conquer benchmark

We randomly source a 5 node graph from coding textbooks and formulate the task as shown below:

Example Question of Divide and Conquer (Floyd-Warhsall)

```
The Floyd-Warshall algorithm on a given graph requires you to compute updates
to the matrix Edges[i][j] according to:
  Edges[i][j] = min(Edges[i][j], Edges[i][k] + Edges[k][j])
You must register the result of each update using the format:
  REGISTER Edges[i][j] = value
--
Problem Instance:  Floyd-Warshall step for k = 0 on a graph with n = 5 nodes.
Initial edge weights:
Edges = [[0, 4, inf, 5, inf], [inf, 0, 1, inf, 6], [2, inf, 0, 3, inf], [inf,
inf, 1, 0, 2], [1, inf, inf, 4, 0]]
Use the REGISTER format for every pair you reference, even if no update
occurs.
```

To evaluate the coverage, we extract the number of registered entries from each response, and a Python script is used to compute the coverage score for the task as defined in the main text.

E Details of coding benchmark

We design a coding benchmark focused on collaborative program synthesis, which allows us to investigate Group Think in a modular, multi-step setting. Each coding task consists of multiple subtasks or functions that must be implemented and combined into a working program. This setup naturally lends itself to parallelism, making it an ideal testbed for collaborative reasoning.

We use `GPT-4o`[5] to generate the dataset. First, the model is asked to produce a complete multi-function programming problem along with a correct full solution. From this solution, we extract a corresponding multi-part problem specification to serve as the task prompt.

The following prompt was used to generate such coding tasks:

GPT-4o Prompt for Coding Task Generation

```
Create a multi-step programming problem that requires writing multiple
Python functions and combining them into a final program.  Provide the full
specification of the task and the complete correct implementation.
```

An example of task produced by `GPT-4o` is provided in Appendix E.1.

To evaluate each predicted solution, we rely on a structured scoring prompt administered through `GPT-4.1`. This prompt appends special tokens to the generated output to mark the completion of each step, which simultaneously marks the depth (of tokens) where each function is achieved. The full evaluation prompt is given below:

---

[5] `https://openai.com/index/gpt-4o`

Scoring Prompt for Coding Outputs

```
You are given a coding problem with multiple steps and its python standard
answer code.
Use it to evaluate the predicted answer given below.  Here are the evaluation
instructions.
Looking at the predicted answer, for each coding implementation, determine
which step is covered from the coding problem.  Then determine the
correctness of the function by looking at the standard answer code.  The
IO spec does not count as one step.
Insert a "<DONE_STEP_{stepid}>" as a validation of that particular
implementation.  If the code only covers a part of the step, insert the token
"<PARTIAL_STEP_{stepid}>" instead.
There is the chance that the implementation simultaneously coveres multiple
steps.  In this case, output multiple tokens sequentially.
Evaluate by inserting tokens in the generated code.  After the implementation
of each step, output <DONE_STEP_{stepid}>, according to the instruction step
in the question.
Your output should be a complete replication of the code with the evaluation
token insertions, do not remove any part of the predicted answer.  wrap it in
a json if the key:  "code_with_evaluation_tokens"
** Problem:  **
{QUESTION}
** Standard answer:  **
{STANDARD_ANSWER}
** Predicted Answer:  **
{RESPONSE}
** End of Predicted Answer:  **
Your output should be a complete replication of the code with
the evaluation token insertions.  wrap it in a json if the key:
"code_with_evaluation_tokens"
```

E.1 Generated Coding Task Example

Below is an example problem produced by `GPT-4o`.

Example Coding Task

Input Spec: Accepts a list of student records, where each record is a dictionary containing:

- a "name" key (string), and
- a "scores" key (list of numeric values representing test scores).

Example:

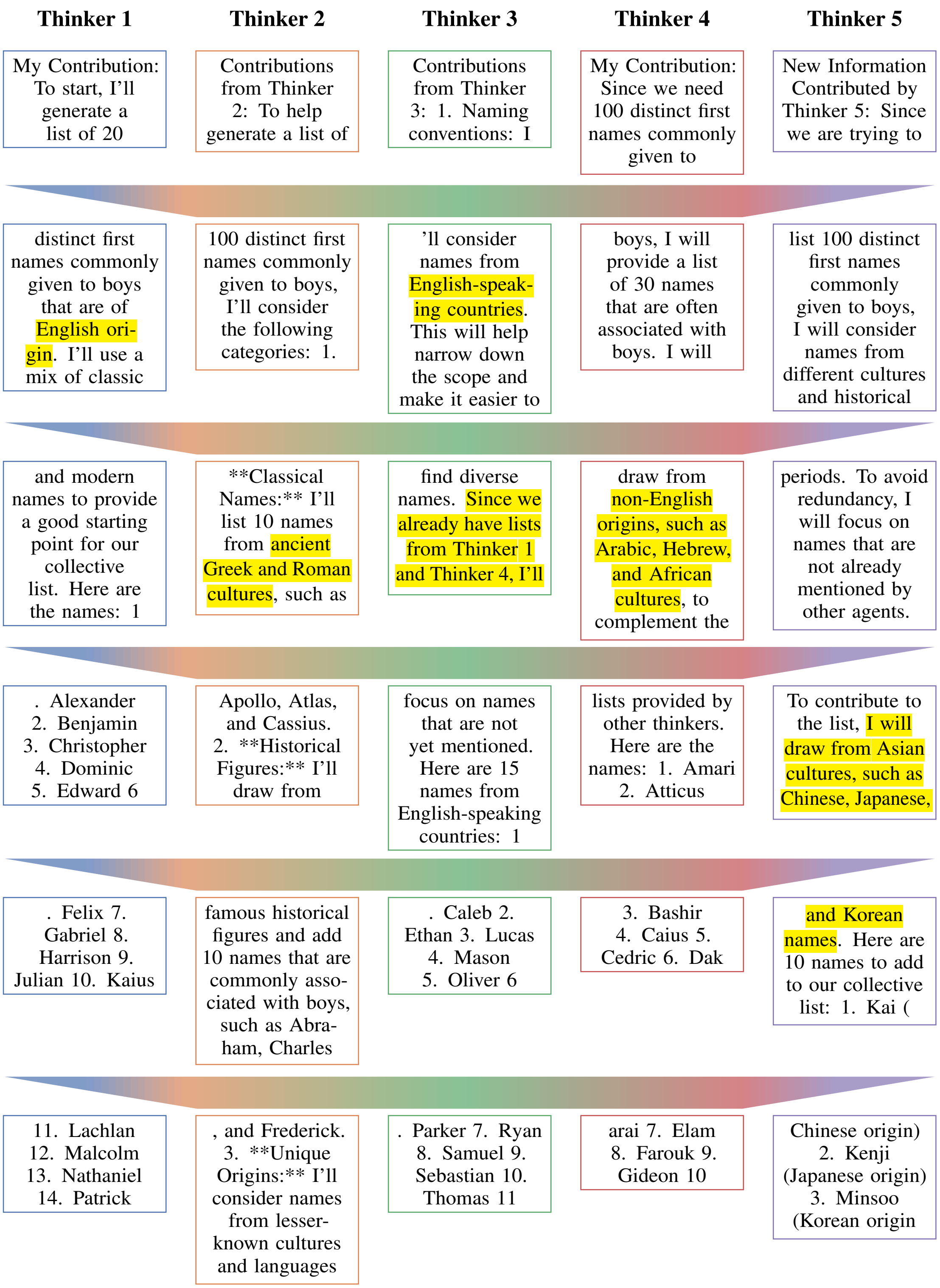
```
students_data = [
    {’name’: ’Alice’, ’scores’: [88, 92, 79]},
    {’name’: ’Bob’, ’scores’: [95, 85, 91]},
    {’name’: ’Charlie’, ’scores’: [70, 65, 72]},
    {’name’: ’Diana’, ’scores’: []}
]
```

1. For each student:
   - Calculate the average score (rounded to two decimal places).
2. For each student:
   - Assign a letter grade based on the following scale:
     - A: 90-100
     - B: 80-89.99
     - C: 70-79.99
     - D: 60-69.99
     - F: below 60
3. Format each student’s result as a string using this structure:

   <Student Name>: Average = <average_score>, Grade = <letter_grade>

   Example:

   Alice: Average = 86.33, Grade = B
4. Combine all formatted results into a single multiline string, with each student’s report on a new line.
5. Print the final report.



Thinker 1	Thinker 2	Thinker 3	Thinker 4	Thinker 5
My Contribution: To start, I’ll generate a list of 20	Contributions from Thinker 2: To help generate a list of	Contributions from Thinker 3: 1. Naming conventions: I	My Contribution: Since we need 100 distinct first names commonly given to	New Information Contributed by Thinker 5: Since we are trying to
distinct first names commonly given to boys that are of English origin. I’ll use a mix of classic	100 distinct first names commonly given to boys, I’ll consider the following categories: 1.	’ll consider names from English-speaking countries. This will help narrow down the scope and make it easier to	boys, I will provide a list of 30 names that are often associated with boys. I will	list 100 distinct first names commonly given to boys, I will consider names from different cultures and historical
and modern names to provide a good starting point for our collective list. Here are the names: 1	**Classical Names:** I’ll list 10 names from ancient Greek and Roman cultures, such as	find diverse names. Since we already have lists from Thinker 1 and Thinker 4, I’ll	draw from non-English origins, such as Arabic, Hebrew, and African cultures, to complement the	periods. To avoid redundancy, I will focus on names that are not already mentioned by other agents.
. Alexander 2. Benjamin 3. Christopher 4. Dominic 5. Edward 6	Apollo, Atlas, and Cassius. 2. **Historical Figures:** I’ll draw from	focus on names that are not yet mentioned. Here are 15 names from English-speaking countries: 1	lists provided by other thinkers. Here are the names: 1. Amari 2. Atticus	To contribute to the list, I will draw from Asian cultures, such as Chinese, Japanese,
. Felix 7. Gabriel 8. Harrison 9. Julian 10. Kaius	famous historical figures and add 10 names that are commonly associated with boys, such as Abraham, Charles	. Caleb 2. Ethan 3. Lucas 4. Mason 5. Oliver 6	3. Bashir 4. Caius 5. Cedric 6. Dak	and Korean names. Here are 10 names to add to our collective list: 1. Kai (
11. Lachlan 12. Malcolm 13. Nathaniel 14. Patrick	, and Frederick. 3. **Unique Origins:** I’ll consider names from lesser-known cultures and languages	. Parker 7. Ryan 8. Samuel 9. Sebastian 10. Thomas 11	arai 7. Elam 8. Farouk 9. Gideon 10	Chinese origin) 2. Kenji (Japanese origin) 3. Minsoo (Korean origin

Figure 7: First six turns from our experiments simulating Group Think for the enumeration task. The thinkers divide their efforts (as highlighted) to avoid redundancy by focusing on names with different origins.

F Reasoning trajectory samples for the Enumeration benchmark

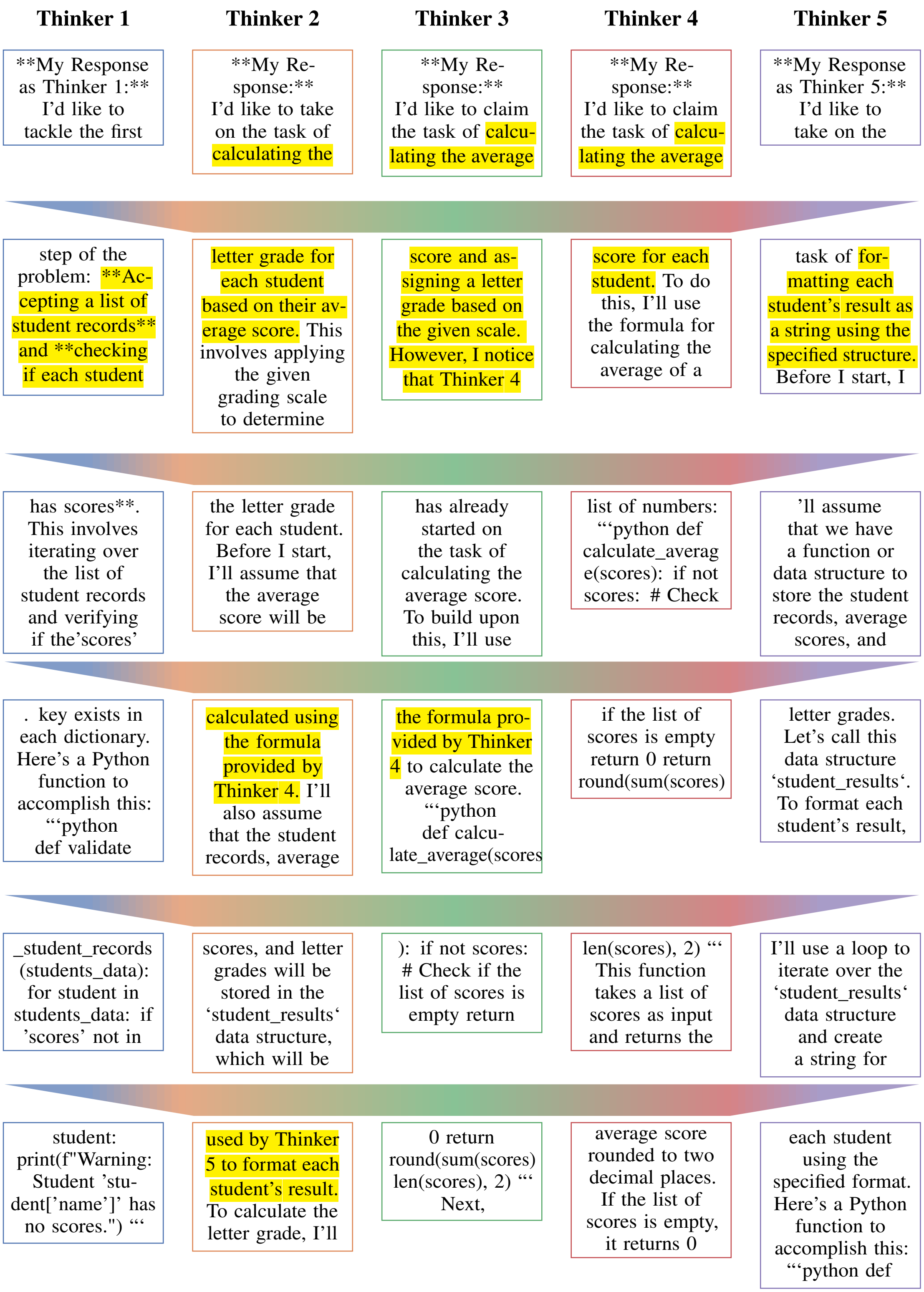
In Figure 7 we show the first six *turns* (i.e., sequences of 20 tokens – we group tokens this way to improve readability) of the output generated by our experiments with Group Think on the enumeration task. This particular example requires the model to produce a list of 100 male names. We notice how the model assigns to each agent the generation of names with different origins to avoid redundancy and reduce the time required to complete the task. This example showcases interpretable gains from Group Think with off-the-shelf instruction models, highlighting promising work directions on fine-tuning with *ad hoc* data to improve their ability to leverage the Group Think mechanism and improve practical deployment. Additional examples are summarized by extracting the identified categories shown below.

Categories found from emergent behaviors indicative of Group Thinking

- Boy Names
    - Alphabetical: A-F, T-Z, H-L
    - Cultural: English-speaking countries, Ancient Greek and Roman cultures, Asian cultures
- Animals
    - Taxonomic Categories: Terrestrial Mammals, Fish, Birds
- Countries
    - Alphabetical: C, V, Z
    - Continent: Asia, Europe, Americas
- Companies
    - Business Segment: E-commerce companies, Retail Sector, Finance Sector
- Athletes
    - Sport Type: Basketball, Baseball, Hockey, Cricket
- Inorganic Compounds
    - Applications: Industrial, Pharmaceutical, Specialized
    - Component: Has transition metal, Has rare earth elements
- Colors
    - Misc: Uncommon, Unique Hue
- Emojis
    - Categories: Plants, Objects, Animals, Flags
- HTML Tags
    - Purpose: Layout and styling, Table, Multimedia
- Verbs
    - Purpose: movement, emotions, social relationships

G Reasoning trajectory samples for the Programming task

In Figure 8 we show the first six turns (i.e., sequences of 20 tokens) of the output generated by our experiments with Group Think on the programming task. As in the enumeration example from Appendix F, we notice that in general the model assigns different tasks to each agent to avoid redundancy. Furthermore, when redundancy happens, the agent can detect it and switch to a different action. This example confirms that current models can take advantage of Group Think, and suggests that training on ad hoc data can lead to even more benefits.



Thinker 1	Thinker 2	Thinker 3	Thinker 4	Thinker 5
My Response as Thinker 1: I'd like to tackle the first	**My Response:** I'd like to take on the task of calculating the	**My Response:** I'd like to claim the task of calculating the average	**My Response:** I'd like to claim the task of calculating the average	**My Response as Thinker 5:** I'd like to take on the
step of the problem: **Accepting a list of student records** and **checking if each student	letter grade for each student based on their average score. This involves applying the given grading scale to determine	score and assigning a letter grade based on the given scale. However, I notice that Thinker 4	score for each student. To do this, I'll use the formula for calculating the average of a	task of formatting each student's result as a string using the specified structure. Before I start, I
has scores**. This involves iterating over the list of student records and verifying if the'scores'	the letter grade for each student. Before I start, I'll assume that the average score will be	has already started on the task of calculating the average score. To build upon this, I'll use	list of numbers: “‘python def calculate_average(scores): if not scores: # Check	'll assume that we have a function or data structure to store the student records, average scores, and
. key exists in each dictionary. Here's a Python function to accomplish this: “‘python def validate	calculated using the formula provided by Thinker 4. I'll also assume that the student records, average	the formula provided by Thinker 4 to calculate the average score. “‘python def calculate_average(scores	if the list of scores is empty return 0 return round(sum(scores)	letter grades. Let's call this data structure ‘student_results‘. To format each student's result,
_student_records (students_data): for student in students_data: if 'scores' not in	scores, and letter grades will be stored in the ‘student_results‘ data structure, which will be	): if not scores: # Check if the list of scores is empty return	len(scores), 2) “‘ This function takes a list of scores as input and returns the	I'll use a loop to iterate over the ‘student_results‘ data structure and create a string for
student: print(f"Warning: Student 'student['name']' has no scores.") “‘	used by Thinker 5 to format each student's result. To calculate the letter grade, I'll	0 return round(sum(scores) len(scores), 2) “‘ Next,	average score rounded to two decimal places. If the list of scores is empty, it returns 0	each student using the specified format. Here's a Python function to accomplish this: “‘python def

Figure 8: First six turns from our experiments simulating Group Think for the coding task. The thinkers divide their efforts (as highlighted) to avoid redundancy by focusing on tasks that have not been solved yet.