Fast Thinking with Structured Prompts: Enabling LLM Reasoning Without Chain-of-Thought Generation

Kirill Morozov
Bauman Moscow State Technical University
morozovka@student.bmstu.ru

Liubov Chubarova
Higher School of Economics
lyubachuba@yandex-team.ru

Irina Piontkovskaya
AI Foundation and Algorithm Lab
irapiont@yandex.ru

Abstract

The emergence of complex reasoning abilities in large language models (LLMs) has sparked great interest, and a variety of prompting techniques was proposed to coax them into emulating human thought processes. In this work, we introduce *Think Node-by-Node*, a graph-based reasoning framework inspired by mind maps, flowcharts, and other visual aids that help humans tackle complex problems. Rather than generating images directly, our approach leverages standard graph-building and rendering libraries, and requires no fine-tuning, only the model's native coding capabilities. We further explore a "Fast Thinking" regime, in which a graph-reasoning example provided in the prompt, but the model generates the answers directly, without the full thought process reconstruction. Surprisingly, this approach leads to significant improvement upon baseline in general-knowledge tasks. Remarkably, Think Node-by-Node maintains strong performance even under a strict 25-token budget for answer generation. Across two instruction-tuned LLMs (0.5B and 7B parameters), our FastTNbN strategy outperforms baseline prompting techniques, improving accuracy by up to 10%, and exceeds the capabilities of other structured prompting methods under equivalent generation constraints.

1 Introduction

Improvement of reasoning ability is an important task for modern LLMs developers. It was shown recently, that asking a model to output intermediate reasoning steps significantly improves its performance on the wide range of tasks involving logic, planning and sophisticated analysis (Wei et al., 2022; Wang et al., 2023). Although even meaningless intermediate output can be used by LLMs as a "thinking" process (Pfau et al., 2024), in general the structure of thoughts can significantly improve the model's performance on separate tasks, especially for the smaller models that are less sensitive to instruction tuning and that perform significantly worse in accurate logical inference (Lu et al., 2024). Many novel thought schemas have been proposed, including multi-agent, graph-based, sampling-based and others approaches (Rasal, 2024; Yao et al., 2024; Ding et al., 2024).

Most of modern LLMs, besides language, can also generate software code samples. It has been demonstrated that adding code data to the pretraining phase implicitly increases the model's reasoning ability (Shi et al., 2023). Coding skills can be also explicitly used in the form of code-style prompting, improving on a separate type of reasoning in natural language (Puerto et al., 2024).

Another line of research inspiring our method is the attempts to integrate a non-linear thinking process (Wen et al., 2024). Indeed, in chain-of-thoughts and most of its variations, thought steps are organized linearly, forming a chain graph. But often mindmaps or charts could represent the problem better. The issue is that it is not clear how to represent non-linear structure inside the framework of token-by-token text generation.

In our work, we propose to utilize the model's coding skills in order to build a graph, or mindmap, for the thought process enchancement. For our experiments, we first select the models able to generate code (drawing the graph) in Python. Then, we prompt a model with a graph-building example (single example for each evaluation dataset), and ask to answer the question, then draw the graph (i.e. generate the graph code) and re-consider the answer. We found that, first, even relatively small models understands the task well and are able to output meaningful graph representing thought process; and, more importantly, the performance of the models on general NLP reasoning benchmarks increases when the graph reasoning is involved.

Investigating structured-prompting techniques on smaller LLMs (under 10B parameters), we ob-

Proceedings of Recent Advances in Natural Language Processing,pages 758–766
Varna, Sep 8–10, 2025
https://doi.org/10.26615/978-954-452-098-4-087

served that many models struggle to stay within a practical generation budget, limiting real-world applicability. In exploring ways to reduce redundant output, we made the surprising discovery that most structured-prompting methods remain effective even if the model does not actually produce the full thought sequence.

Building on this insight, we introduce *Fast Thinking*, in which the model is shown an example of structured reasoning but is prompted to emit only a preliminary answer—halting before the full thought chain—with a strict 25-token limit. Across multiple benchmarks, Fast Thinking boosts answer quality by up to 10% compared to standard prompting, and methods such as Tree of Thoughts (Yao et al., 2023) and Program of Thoughts (Chen et al., 2023) perform robustly in this regime.

We further propose *Think Node-by-Node (TNbN)*—a new graph-based prompting strategy that excels under Fast Thinking. Without any weight updates, FastTNbN delivers improvements of up to 10% across diverse datasets, while keeping generation under 25 tokens. We validate our approach on two instruction-tuned models from different families—`Qwen2.5-0.5B` and `Mistral-v3-7B`—demonstrating that, with sufficient coding proficiency, even models as small as 0.5 B parameters can benefit substantially.

2 Literature Review

The manipulation with prompt has emerged as a crucial technique for enhancing the capabilities of pre-trained large language models (Radford et al., 2019; Brown et al., 2020). The Chain-of-Thought prompting technique introduced in (Wei et al., 2022), and its extensions (Zhang et al., 2024; Yang et al., 2024; Long, 2023) compels the language model to produce not merely the final answer but also a coherent sequence of intermediate reasoning steps. The Graph-of-Thoughts (GoT) method was firstly proposed by (Besta et al., 2024). The process from now is not just a simple prompt engineering but a two-stage framework with the particular module for the graph construction. However, the methodological foundation for GoT can be traced to the earlier Tree of Thoughts (ToT) paradigm in (Yao et al., 2023). ToT generalizes the popular Chain of Thought approach to prompting language models, and enables exploration over coherent units of text (thoughts). ToT allows LMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices.

Significant advancements have also been made to enhance reasoning by leveraging code and programming within the Program of Thoughts (Chen et al., 2023) method. Encouraging models to "dig inward" to arrive at an answer can be interpreted in various ways. However, approaches that compel large language models (LLMs) to produce clearer and more accurate responses without relying on additional information are undeniably worth exploring. A prime example is the Zero-shot method augmented with the prompt "Take a deep breath and work on this problem step-by-step." (Yang et al., 2024) (Zero-shot with TDB) prepended to the question. This technique has proven effective in guiding models toward more structured and precise reasoning. It is also worth noting the method that facilitates reasoning by employing a so-called "step back", enabling the model to approach the problem from a different perspective. This technique, known as Step-Back Prompting (Zheng et al., 2024), encourages the model to pause and reassess the problem, often leading to more insightful and creative solutions.

3 Method

3.1 Fast Thinking

In this work, we leverage the notion of *fast thinking*, which can be integrated with any structured prompting framework. Specifically, we present the model with few-shot examples of the desired reasoning structure, but inspect the answer obtained *before* the thought process. To encourage the model to produce this initial answer, we append a preliminary short response to each example, formatting them as *[question][preliminary answer][thought][answer]*. During inference, the generation can be halted once the preliminary answer is produced; in practice, we just restrict the generation by 25 tokens. In Section 4.6, we demonstrate that this method is surprisingly effective in combination with the existing prompting techniques , and with our Think Node-by-Node method.

3.2 Think Node-by-Node Prompting

We introduce a novel prompting technique designed to stimulate graphical reasoning in smaller LLMs without relying on a visual interface. Our

Answer: D

Primary Answer

Node of thoughts:
```python

TNbN

```
from graphviz import Digraph

# Create a directed graph
dot = Digraph(comment='Thought Process: Repeat the Investigation')

# Define nodes with descriptions
dot.node('A', 'Initial Hypothesis')
dot.node('B', 'Conclusion Supports Hypothesis')
dot.node('C', 'Understanding Why Hypothesis is Correct')
dot.node('D', 'Validating Results')

# Add edges to show the flow of steps
dot.edge('A', 'B', label='Initial Investigation Results')
dot.edge('B', 'C', label='Unnecessary to understand why', style='dashed')
dot.edge('B', 'D', label='Required to make sure results are reliable')

# Highlight the correct answer with a bold edge
dot.edge_attr.update(style='bold')
dot.edge('B', 'D')

# Render the graph
dot.render('thought_process', format='png', cleanup=True)
```

Final Answer

The final result of the Node of Thoughts after "# Highlight the correct answer with a bold edge" from dot.edge('B', 'D'): 'Validating Results': 'Required to make sure results are reliable'
```

The final result of the Graph of thoughts is 'Validating Results': 'Required to make sure results are reliable'. This means that Derek should repeat the investigation to make sure the results are valid.

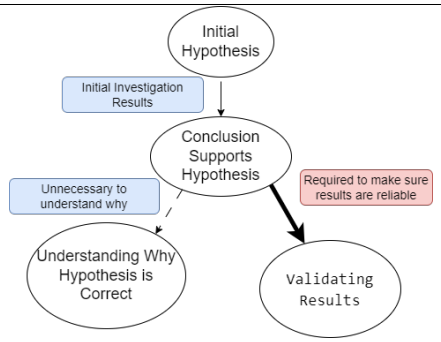

Figure 1: (top) The structure of TNbN output. (bottom) Generalized corresponding graph obtained from the code.

approach represents the model's thought graph via a *Python* script using the `graphviz` library. This choice of representation is both formally precise and readily interpretable by the model, since LLMs cannot natively ingest graph structures.

The prompt includes a single working example (see Fig. 1). First, the code defines each graph node and assigns its label. Next, it specifies the edges connecting those nodes. Finally, the standard `graphviz` layout and drawing commands produce the visual representation. For certain domains—such as the ARC science questions—we optionally include edge labels to capture relationships between scientific concepts. Additionally, we ask the model to render the edge leading to the final answer in **bold**. This format is highly general and can be adapted to any task. Depending on the application, nodes might represent hypotheses, intermediate reasoning steps, answer choices, scientific phenomena, named entities, calculations,

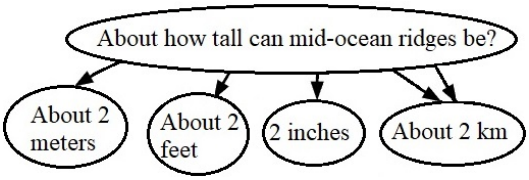

Figure 2: Sample graphs produced by Qwen2.5-0.5B-Instruct (dataset SciQ).

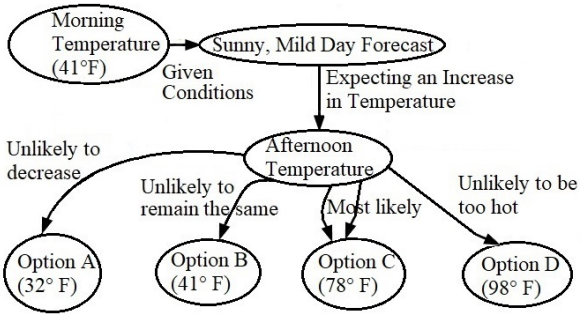

Figure 3: Sample graphs produced by Mistral-7B-Instruct-v0.3 (dataset ARC-Challenge).

and so on. The essential component is the "graph thinking" pattern itself, which we present to the model in three distinct variants:

Choice(Figure 2) This pattern is good for exam-like questions with answer options choice, which do not require complex reasoning. The first node corresponds to the whole question. Then, the model analyses each option separately, creating one node per option, and writing some commentary to it as a node label. Edges are connecting the question node with the options, and correct option is emphasized.

Relations(Figure 3) is the pattern focused on the analysis of relations between entities. Here, the node names represent entities, node descriptions represent the type of the connections to the previously introduced entities, and edges show the causal or other connections between them.

Flow(Figure 4) represents the process of thinking for general tasks with open-ended questions. Fist, the model generate nodes containing elementary claims, extracted from the task, or derived from the previous ones. Then, it draws direct edges representing the derivation process.

Overall, the code is well structured: the reasoning process remains clear even without rendering the graph (see Figure 1).

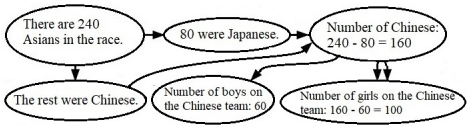

Figure 4: Sample graphs produced by Mistral-7B-Instruct-v0.3 (dataset GSM8K).

	ARC-Challenge			ARC-Easy			GSM8K			SciQ		
	Base	CoT	TNbN	Base	CoT	TNbN	Base	CoT	TNbN	Base	CoT	TNbN
Gemma-2-9B-it	1 (0.17)	1 (0.03)	1 (1)	1 (0.11)	1 (0.18)	1 (1)	1 (0.36)	1 (0.39)	1 (1)	1 (0.05)	1 (0.18)	1 (1)
Llama-3-8B	0.59 (0)	0.04 (0.02)	0.69 (0.85)	0.48 (0)	0.06 (0.05)	0.59 (0.85)	0.21 (0)	0.04 (0.03)	0.35 (0.25)	0.60 (0)	0.07 (0.03)	0.48 (0.10)
Llama-3-8B-Instruct	1 (0.05)	1 (0.04)	1 (1)	1 (0.05)	1 (0)	1 (1)	0.98 (0)	1 (0)	1 (1)	1 (0)	1 (0.14)	1 (1)
Phi-3-Mini-128K-Instruct	0.98 (0.14)	0.53 (0.81)	0.52 (1)	1 (0.13)	0.72 (0.95)	0.52 (1)	0.98 (0.84)	1 (1)	0.97 (0.89)	1 (0.22)	1 (1)	1 (1)
Vicuna-13B-v1.5	0.62 (0.10)	0.78 (0.08)	0.38 (0.43)	0.51 (0.14)	0.46 (0.12)	0.21 (0.33)	0.14 (0.12)	0.30 (0.08)	0.84 (0.71)	0.48 (0.05)	0.80 (0.05)	0.82 (0.41)
Qwen2.5-0.5B-Instruct	0.40 (0.04)	0.55 (0.83)	1 (1)	0.50 (0.10)	0.40 (0.87)	1 (1)	0.82 (0.07)	0.25 (0.54)	1 (1)	0.67 (0.07)	0.65 (0.85)	1 (1)
Mistral-7B-Instruct-v0.3	1 (0.70)	1 (1)	1 (1)	1 (0.78)	1 (1)	1 (1)	1 (0.94)	0.94 (0.93)	1 (1)	1 (0.28)	1 (1)	1 (1)

Figure 5: The level of format retention across different models for the Base (Baseline), CoT (Chain-of-Thoughts), and TNbN (Think Node by Node) methods. Cases where format retention is at least 0.7 are highlighted in yellow. Additionally, cases where both the primary response and subsequent format retention are maintained are highlighted in green.

4 Experiments

4.1 Structured Prompting Baselines

Here, we list the main types of logical thinking and several examples of benchmarks that test them.

The reasoning process for each method is distinct: **Baseline** represents the classic approach, providing a direct answer followed by an explanation. **Chain-of-Thoughts** (Wei et al., 2022) involves a structured sequence of reasoning steps. **Program of Thoughts** (Chen et al., 2023) leverages a code-based reasoning framework. **Step-Back Prompting** (Zheng et al., 2024) encourages reflective, internal textual reasoning, allowing the model to reframe the question by taking a metaphorical "step back." **Zero-shot with TDB** (Yang et al., 2024) operates without examples or additional conditions, relying solely on self-declarative instructions. **Tree of Thoughts** (Yao et al., 2023) considers several different lines of reasoning and independently evaluates choices to decide on the next course of action, and looks ahead or back when necessary to make a global choice. **Graph of Thoughts** (Besta et al., 2024) this approach enables combining arbitrary LLM thoughts into synergistic outcomes, distilling the essence of whole networks of thoughts, or enhancing thoughts using feedback loops.

4.2 Datasets

We evaluate the capabilities of the methods across diverse domains and problem types. For general multiple-choice science questions, we select SciQ (Moore et al., 2023), ARC (Bhakthavatsalam et al., 2021), MMLU (Wang et al., 2024). Mathematical skills are covered by GSM8K (Cobbe et al., 2021), AQuA (Ling et al., 2017), SVAMP (Patel et al., 2021), TabMWP (Kang et al., 2024), MultiArith (Roy et al., 2015). Multi-hop reasoning data includes StrategyQA (Geva et al., 2021), MuSiQue (Trivedi et al., 2022). Finally, we add special knowledge taks: time-related question from TimeQA (Chen et al., 2021), and financial reasoning FinQA (Chen et al., 2022b) and ConvFinQA (Chen et al., 2022a).

4.3 Models

For our preliminary experiments, we chose LLMs from different families: Gemma-2-9B-it (Team et al., 2024), Llama-3-8B, Llama-3-8B-Instruct (Grattafiori et al., 2024), Phi-3-Mini-128K-Instruct (Abdin et al., 2024), Vicuna-13B-v1.5 (Zheng et al., 2023), Qwen2.5-0.5B-Instruct (Yang et al., 2025), and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023). On the second stage of evaluation, only the latter two models were used.

4.4 Experimental Setup

We conduct a two-stage experimental evaluation. In the first (model-selection) phase, we test each model's ability to interpret and reproduce our graph-reasoning patterns and to follow the required output format. We choose three benchmarks—ARC, SciQ, and GSM8K—and compare three reasoning strategies: Baseline, Chain-of-Thought (CoT), and Think Node-by-Node (TNbN). Each model is evaluated on 100 randomly sampled instances per dataset, and outputs are assessed manually. We perform this selection phase because our method depends on the model's programming proficiency and familiarity with graph-rendering libraries; only those that reliably follow the prompt can meaningfully benefit from our approach.

In the second (main-experiment) phase, we restrict our analysis to the top-performing models identified in phase one and carry out a comprehensive evaluation across all benchmarks. For each dataset, we generate a graph-reasoning example (presented here: https://github.com/MorKir/graph-reasoning-examples.git) by selecting a random

question and asking GPT-4 to produce Graphviz code that visualizes the reasoning process. Inference uses greedy decoding; for the Fast Thinking experiments, we enforce a 25-token limit on generation.

4.5 Evaluation of LLM's Ability to Follow Graph Reasoning Format

In this phase, we are evaluating models based on their ability to handle specific reasoning formats.

In Figure 5, each model is evaluated by two metrics: the *primary score*—the probability of producing the preliminary answer immediately, as demonstrated in the example—and the *secondary score* (in parentheses)—the probability of maintaining the required format throughout the subsequent reasoning steps after that answer. Our results show that the TNbN format is remarkably accessible to these models: four out of six achieve a perfect secondary score, outperforming both the baseline and CoT prompts in format compliance. Strict adherence to the answer format is often a bottleneck for smaller LLMs, so this improvement is especially significant. Across all models, Mistral-7B-Instruct-v0.3 leads in format-compliant answer generation, followed by Qwen2.5-0.5B-Instruct and then Phi-3-Mini-128K-Instruct. Although Gemma-2-9B-it and Llama-3-8B-Instruct also attain perfect primary scores under all prompting methods and perfect TNbN secondary scores, they fail to sustain the full reasoning pattern, resulting in near-zero secondary scores for CoT and baseline. Notably, model size does not predict format compliance: among the 7–8B models, only Mistral consistently follows the prescribed format under both CoT and TNbN. Based on these summary metrics, we select Mistral-7B-Instruct and Qwen2.5-0.5B for the full evaluation. In Section 5, we provide more details about format compliance issues of the evaluated models.

4.6 Main Results

Table 3 report the Fast Thinking results under a 25-token generation limit for each prompting strategy. In both models, structured prompts consistently outperform the baseline, with improvements of up to 10 % on certain datasets. Among the methods tested, TNbN achieves the highest accuracy on most benchmarks.

Table 1 shows the average ranking score for each method. For each dataset, we rank the methods from best to worst and assign scores of {1.0, 0.8, 0.6, 0.4, 0.2, 0.0} accordingly; these scores are then averaged across datasets. FastTNbN, PoT, and ToT all exceed the baseline in the Fast Thinking regime, with FastTNbN emerging as the clear leader.

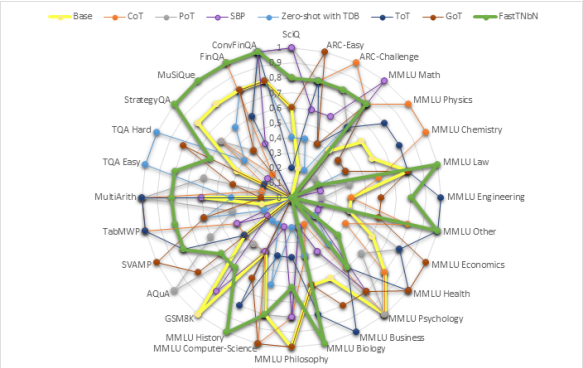

Figure 6: Generalized results of the methods for Mistral-7B-Instruct-v0.3.

In Table 4, we compare FastTNbN to its slower counterpart by reporting accuracy on the *primary answers* (before reasoning), the *final answers* (after the full thought process), and the *intermediate answers* extracted via regular expressions from each model's output. Mathematics is the only domain in which full reasoning consistently boosts the primary answer accuracy —— most notably on GSM8K, AQuA-Q, and MMLU Math —— while MMLU Economics and Psychology also benefit for Mistral. On other datasets, primary answer accuracy is equal to or better than final accuracy. This suggests that, for many tasks, generating reasoning tokens is not strictly necessary to improve the model's internal reasoning, although the choice of prompting strategy remains critical. In classical domains, the primary LLM response requires minimal cognitive effort (functioning as an automatic thinking system (Kahneman, 2011; Zhang et al., 2025)). In FastTNbN, this capability—along with graphical reasoning—is actively engaged (and is effectively applied immediately based on the given example). For other methods we also ask to repeat the reasoning process.

5 Analysis and Ablation Study

Format compliance issues of different models. Here, we describe typical issues with thought generation, observed in experiments in Sec. 4.5

- Gemma-2-9B-it. Thought generation exhibited structural breakdowns and irrational interruptions in thought flow. Final answers were frequently missing or misplaced in the output.

Mistral-7B-Instruct-v0.3							
Baseline	**CoT**	**PoT**	**SBP**	**Z-TDB**	**ToT**	**GoT**	**FastTNbN(Ours)**
15.6	18.6	16.2	10.4	10.6	17.2	18.2	**19.6**
Qwen2.5-0.5B-Instruct							
Baseline	**CoT**	**PoT**	**SBP**	**Z-TDB**	**ToT**	**GoT**	**FastTNbN(Ours)**
18.2	10.4	19.0	5.4	9.6	10.8	18.8	**20.4**

Table 1: Mistral-7B-Instruct-v0.3 and Qwen2.5-0.5B-Instruct average ranking score. The score of different approaches to thinking in LLM are presented: Baseline, CoT - Chain-of-Thoughts, PoT - Program of Thoughts, SBP - Step-Back Prompting, Z-TDB - Zero-shot with TDB, ToT - Tree of Thoughts, GoT - Graph of Thoughts, FastTNbN - Fast Thought Node by Node.

GraphQA-Easy			**GraphQA-Hard**		
	Mistral-7B	Qwen-0.5		Mistral-7B	Qwen-0.5B
Overall	**0.38**	**0.18**	**Overall**	**0.33**	**0.15**
Connected Nodes	0.38	0.01	Node Classification	0.72	0.36
Edge Existence	0.57	0.56	Maximum Flow	0.08	0.00
Node Degree	0.07	0.07	Disconnected Nodes	0.07	0.03
Node Count	0.35	0.02	Reachability	0.85	0.32
Cycle Check	0.92	0.36	Shortest Path	0.23	0.14
Edge Count	0.00	0.03	Triangle Counting	0.02	0.03

Table 2: Evaluating the models ability to solve graph-based problems.

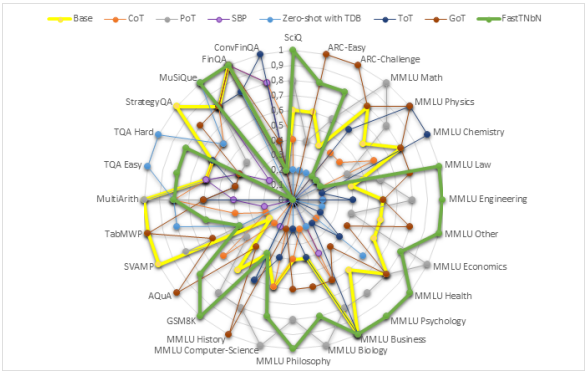

Figure 7: Generalized results of the methods for Qwen2.5-0.5B-Instruct.

- Llama-3-8B. Code generation lost its structure and meaning, the content of the thought process was looped or interrupted without good reason. The final answer could be missing despite an adequate thought process.
- Llama-3-8B-Instruct. There was a clear trend whereby LLMs did not complete the second part (code generation), let alone the final answer. The second part of the answer frequently contained repetitive content and information unrelated to the question.
- Phi-3-Mini-128K-Instruct. The LLM frequently failed to complete the second part (thought generation), often stopping short of producing the final answer.
- Vicuna-13B-v1.5. The primary answer was improperly presented. Failure to comply with the response format in principle, its absence, the output of information that is completely unrelated to the request.

Generalized results comparison. The results presented in Table 1 were compiled to clearly demonstrate the overall success of our method across the set of considered domains for both Mistral (Fig. 6) and Qwen (Fig. 7).

Graph reasoning skills of the selected models. The models ability to support the required format was verified earlier. The next step is to ensure that the selected models are fundamentally capable of solving graph-related problems. For this purpose, the **GraphQA-Easy** and **GraphQA-Hard** datasets are utilized. Each model is evaluated using 100 questions per problem category from these datasets.

The results in Table 2 show that both selected models can solve graph-structured problems, with `Mistral-7B-Instruct-v0.3` demonstrating roughly twice the accuracy of `Qwen2.5-0.5B-Instruct`. Since both models perform strongly under the FastTNbN setup, we detect the elements of graph reasoning which are

	#	Base	CoT	PoT	SBP	Z-TDB	ToT	GoT	FastTNbN
Qwen2.5-0.5B-Instruct									
Science Question									
SciQ	1000	0.42	0.36	*0.45*	0.21	0.26	0.09	0.21	***0.51***
ARC-Easy	2375	0.24	0.13	0.23	0.05	0.19	0.07	***0.32***	*0.30*
ARC-Challenge	1172	0.19	0.12	*0.22*	0.04	0.14	0.04	***0.30***	*0.28*
Overall MMLU	12032	0.15	0.10	***0.17***	0.09	0.10	0.14	0.16	***0.17***
STEM-MMLU	4873	0.15	0.10	***0.18***	0.09	0.09	0.15	0.15	0.15
NOT-STEM-MMLU	7159	0.15	0.10	*0.16*	0.09	0.11	0.13	*0.17*	***0.18***
Math									
GSM8K	1319	0.08	0.04	*0.12*	0.02	0.02	0.01	0.05	***0.14***
AQuA	254	0.10	*0.19*	*0.15*	0.02	0.01	0.03	***0.22***	0.21
SVAMP	1000	**0.19**	0.07	0.16	0.04	0.10	0.04	0.11	0.10
TabMWP	751	**0.17**	0.10	0.07	0.08	0.14	0.05	**0.17**	0.11
MultiArith	420	**0.08**	0.04	**0.08**	0.02	0.04	0.03	0.03	0.04
Multi-Hop Reasoning									
StrategyQA	1603	**0.42**	0.18	0.17	0.14	0.18	0.05	0.32	0.12
MuSiQue	2417	0.00	0.00	0.00	0.00	***0.01***	0.00	0.00	***0.01***
Special Knowledge QA									
TimeQA Easy	2996	0.01	0.01	0.01	0.01	***0.05***	0.00	0.00	*0.03*
TimeQA Hard	3078	0.01	0.01	0.00	0.00	***0.05***	0.01	0.00	*0.03*
FinQA	1146	**0.01**	**0.01**	0.00	**0.01**	**0.01**	0.00	**0.01**	**0.01**
ConvFinQA	3037	0.02	*0.05*	0.02	*0.05*	0.01	***0.06***	*0.04*	0.02
Mistral-7B-Instruct-v0.3									
Science Question									
SciQ	1000	0.84	*0.85*	***0.86***	***0.86***	0.77	0.13	0.84	*0.85*
ARC-Easy	2375	0.65	*0.83*	*0.83*	*0.81*	*0.77*	*0.83*	***0.85***	*0.83*
ARC-Challenge	1172	0.60	***0.72***	*0.71*	*0.69*	*0.63*	*0.68*	*0.68*	*0.71*
Overall MMLU	12032	0.27	***0.28***	0.26	0.24	0.24	***0.28***	***0.28***	0.27
STEM-MMLU	4873	0.25	***0.26***	0.24	0.22	0.22	***0.26***	***0.26***	0.24
NOT-STEM-MMLU	7159	0.30	0.30	0.28	0.27	0.25	***0.31***	0.30	0.30
Math Word Problems									
GSM8K	1319	**0.14**	0.09	0.09	0.11	0.05	0.04	0.04	0.10
AQuA	254	0.20	0.20	***0.24***	0.12	0.07	0.20	*0.23*	*0.22*
SVAMP	1000	0.47	*0.55*	*0.56*	*0.55*	*0.54*	*0.57*	***0.59***	*0.57*
TabMWP	751	0.39	***0.46***	*0.40*	0.25	*0.42*	***0.46***	*0.42*	*0.43*
MultiArith	420	0.13	***0.15***	***0.15***	0.13	0.12	***0.15***	0.11	*0.14*
Multi-Hop Reasoning									
StrategyQA	1603	0.82	0.81	0.81	0.69	0.75	0.68	0.68	***0.83***
MuSiQue	2417	0.10	0.05	0.06	0.02	0.09	0.04	0.06	***0.11***
Special Knowledge QA									
TimeQA Easy	2996	0.31	*0.34*	*0.36*	0.06	***0.40***	0.30	*0.35*	*0.39*
TimeQA Hard	3078	0.23	0.22	0.21	0.02	***0.31***	0.22	0.25	0.24
FinQA	1146	0.04	***0.05***	0.03	0.01	0.04	0.03	0.04	***0.05***
ConvFinQA	3037	0.01	***0.02***	***0.02***	***0.02***	0.01	***0.02***	0.01	***0.02***

Table 3: Qwen2.5-0.5B-Instruct and Mistral-7B-Instruct-v0.3 results on datasets. The results of different approaches to thinking in LLM are presented: Baseline, CoT - Chain-of-Thoughts, PoT - Program of Thoughts, SBP - Step-Back Prompting, Z-TDB - Zero-shot with TDB, ToT - Tree of Thoughts, GoT - Graph of Thoughts, FastTNbN - Fast Thought Node by Node.

	Mistral-7B-Instruct-v0.3			Qwen2.5-0.5B-Instruct		
	Primary	**In the process**	**Final**	**Primary**	**In the process**	**Final**
SciQ	**0.85**	0.58	0.82	**0.51**	0.46	0.43
ARC-Easy	**0.83**	0.24	0.64	0.30	0.16	**0.38**
ARC-Challenge	**0.71**	0.19	0.54	0.28	0.14	**0.35**
STEM-MMLU	**0.24**	0.13	**0.24**	**0.15**	0.04	0.14
NOT-STEM-MMLU	0.30	0.16	**0.31**	**0.18**	0.06	0.15
GSM8K	0.10	0.03	**0.25**	**0.14**	0.03	0.05
AQuA	0.22	0.13	**0.38**	0.21	0.26	**0.55**
SVAMP	**0.57**	0.00	0.08	**0.10**	0.00	0.00
StrategyQA	**0.83**	0.11	0.35	**0.12**	0.00	0.00
MuSiQue	0.11	0.19	**0.20**	**0.01**	**0.01**	**0.01**

Table 4: Comparison of the accuracy of Fast and Thoughtful TNbN approaches

essential for this technique. It can be said that the borderline versions of models from what can be called line small models are being tested.

Specifically, both models excel at *existence* and *classification* tasks — such as Edge Existence, Cycle Detection, Node Classification, and Reachability —- but struggle with *counting*-based problems (Node Degree, Node Count, Edge Count) and more complex algorithms like Maximum Flow. Their performance on Shortest Path queries is modest yet nontrivial. These findings suggest that fundamental graph-understanding abilities suffice for effective FastTNbN prompting.

6 Conclusion

In this work, we demonstrate the effectiveness of Fast Thinking: a method in which the model is provided an example of structured reasoning, yet reasoning-chain generation is curtailed to as few as 25 tokens. This approach is far more computationally efficient than conventional Chain-of-Thought prompting and other structured reasoning techniques. While it is often assumed that such methods succeed primarily through extended autoregressive computation, we show that structured prompting remains effective even with minimal token generation. In Fast Thinking, the bulk of computation shifts to prompt processing — performed in a single inference step — and can be further accelerated (e.g., via KV-cache optimizations) to reduce answer latency.

Next, we introduce a novel structured-prompting technique that leverages the LLM's programming capabilities to generate a task schema as a graph, encoded in a standard graph-rendering library. We find that this method excels in the Fast Thinking regime, outperforming both the baseline and alternative structured prompts by up to 10 % on various datasets under a 25-token generation budget. Remarkably, our approach remains effective even with very small models (starting at 0.5 B parameters). It requires no weight updates and is domain-agnostic, making it compatible with any training or adaptation pipeline. Although our technique depends on models with strong coding proficiency, we believe it offers valuable insights into LLM reasoning and control, and can enable efficient LLM-based systems in settings where inference speed and model size are constrained.

7 Limitations

In this work, we present the TNbN prompting strategy along with its optimized modification, the FastTNbN prompting strategy, allowing to utilize the model's coding skills for structuring general reasoning process. This is the main limitation of our work: our method is applicable only to code-language models, requires strict adherence to a specific format, and many existing LLMs are not supported. We checked our approach on several standard reasoning benchmarks, but it does not necessarily lead to high performance on general real-world situations. In our further research, we plan to extend the scenarios for testing our novel graph-code prompting approach, and consider its applicability for planning and general real life-related problems.

Acknowledgments

This work has been carried out as part of the machine learning summer school SMILES-2024, organized by the Skoltech Center for Applied AI and R&D № FSFN-2024-0059.

References

Marah Abdin et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Maciej Besta et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. volume 38, page 17682–17690. Association for the Advancement of Artificial Intelligence (AAAI).

Sumithra Bhakthavatsalam et al. 2021. Think you have solved direct-answer question answering? try arc-da, the direct-answer ai2 reasoning challenge.

Tom B. Brown et al. 2020. Language models are few-shot learners.

Wenhu Chen et al. 2021. A dataset for answering time-sensitive questions. *Preprint*, arXiv:2108.06314.

Wenhu Chen et al. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

Zhiyu Chen et al. 2022a. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. *Preprint*, arXiv:2210.03849.

Zhiyu Chen et al. 2022b. Finqa: A dataset of numerical reasoning over financial data. *Preprint*, arXiv:2109.00122.

Karl Cobbe et al. 2021. Training verifiers to solve math word problems.

Ruomeng Ding et al. 2024. Everything of thoughts: Defying the law of penrose triangle for thought generation.

Mor Geva et al. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Aaron Grattafiori et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Albert Q. Jiang et al. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Daniel Kahneman. 2011. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, NY.

Xiaoqiang Kang et al. 2024. Template-driven llm-paraphrased framework for tabular math word problem generation. *Preprint*, arXiv:2412.15594.

Wang Ling et al. 2017. Program induction by rationale generation : Learning to solve and explain algebraic word problems. *Preprint*, arXiv:1705.04146.

Jieyi Long. 2023. Large language model guided tree-of-thought.

Sheng Lu et al. 2024. Are emergent abilities in large language models just in-context learning?

Steven Moore et al. 2023. Crowdsourcing the evaluation of multiple-choice questions using item-writing flaws and bloom's taxonomy. In *Proceedings of the Tenth ACM Conference on Learning @ Scale*, L@S '23, page 25–34, New York, NY, USA. Association for Computing Machinery.

Arkil Patel et al. 2021. Are nlp models really able to solve simple math word problems? *Preprint*, arXiv:2103.07191.

Jacob Pfau et al. 2024. Let's think dot by dot: Hidden computation in transformer language models.

Haritz Puerto et al. 2024. Code prompting elicits conditional reasoning abilities in text+code llms.

Alec Radford et al. 2019. Language models are unsupervised multitask learners. *OpenAI*. Accessed: 2024-11-15.

Sumedh Rasal. 2024. Llm harmony: Multi-agent communication for problem solving.

Subhro Roy et al. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

Ensheng Shi et al. 2023. Towards efficient fine-tuning of pre-trained code models: An experimental study and beyond. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, page 39–51, New York, USA. Association for Computing Machinery.

Gemma Team et al. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Harsh Trivedi et al. 2022. Musique: Multihop questions via single-hop question composition. *Preprint*, arXiv:2108.00573.

Lei Wang et al. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models.

Yubo Wang et al. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Preprint*, arXiv:2406.01574.

Jason Wei et al. 2022. Chain-of-thought prompting elicits reasoning in large language models.

Yilin Wen et al. 2024. Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models.

Chengrun Yang et al. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Qwen: An Yang et al. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Shunyu Yao et al. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Yao Yao et al. 2024. Beyond chain-of-thought, effective graph-of-thought reasoning in language models.

Shao Zhang et al. 2025. Leveraging dual process theory in language agent framework for real-time simultaneous human-ai collaboration. *Preprint*, arXiv:2502.11882.

Zhuosheng Zhang et al. 2024. Multimodal chain-of-thought reasoning in language models.

Huaixiu Steven Zheng et al. 2024. Take a step back: Evoking reasoning via abstraction in large language models. *Preprint*, arXiv:2310.06117.

Lianmin Zheng et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.