# Working Title

**Anonymous Authors**[1]

## Abstract

## 1. Introduction

## 2. Related Work

Structured reasoning with language models has become a prominent research direction. One foundational approach is *Chain-of-Thought (CoT) prompting*, which elicits intermediate reasoning steps from the model. By having the model "think aloud" through a series of sub-steps, CoT significantly improves performance on complex tasks (Wei et al., 2022). Building on this idea, more advanced frameworks allow branching and search over possible reasoning paths instead of following a single linear chain. For example, *Tree-of-Thoughts (ToT)* generalizes CoT by expanding a tree of potential "thought" steps and using self-evaluation to decide among branches, enabling lookahead and backtracking during inference (Yao et al., 2023a). Such deliberate search over a reasoning tree or DAG can yield far better results on tasks requiring planning or strategic exploration, as the model is not confined to greedy left-to-right generation. Relatedly, alternative decoding strategies like self-consistency have been proposed to improve reliability: instead of taking one pass through the prompt, the model samples multiple diverse reasoning chains and then selects the most consistent answer among them (Wang et al., 2022). This method leverages the intuition that a complex problem may be solved via different logical routes leading to the same answer, and indeed has been shown to greatly boost accuracy on benchmarks (Wang et al., 2022).

Another line of work augments language model reasoning with *external tools or formal executors*. *Program-Aided Language Models (PAL)* exemplify this trend by generating programs (e.g., Python code) as intermediate reasoning steps and offloading their execution to a runtime (Gao et al., 2023). In PAL, the language model's job is to correctly decompose the problem into code, and the Python interpreter handles the actual calculation or logic—an approach that achieved state-of-the-art results on math and symbolic reasoning tasks, even outperforming much larger models that rely on CoT alone (Gao et al., 2023). Beyond code execution, researchers have equipped LMs with a variety of *tool-use abilities*. *Toolformer* showed that an LM can be fine-tuned (in a self-supervised way) to decide when to call external APIs (such as calculators, web search, or translation services) and how to incorporate the results into its text generation (Schick et al., 2023). This allows the model to overcome its weaknesses (e.g., factual lookup, arithmetic) by delegating those subtasks to specialized tools, substantially improving zero-shot performance without sacrificing general language ability. In a similar vein, prompting strategies like *ReAct* intermix textual reasoning with explicit actions. ReAct prompts the model to produce both *Thoughts* (natural language reflections) and *Actions* (commands like queries to a knowledge base or environment) in an interleaved manner (Yao et al., 2023b). By design, the model's "thought" can trigger an external tool (e.g., a Wikipedia lookup) and then incorporate the tool's output before continuing the reasoning. This tight integration of tool usage within the reasoning process helps address issues like hallucination and enables tackling interactive decision-making tasks that pure text generation would struggle with (Yao et al., 2023b). Overall, these tool-augmented systems demonstrate the benefit of extending LMs beyond the text-only paradigm—a theme also reflected in many recent agent frameworks.

Indeed, there has been a surge of interest in treating LMs as *autonomous agents or planners* that can control their own multi-step decision process. Projects such as AutoGPT (Significant-Gravitas, 2023) and BabyAGI (Yohei Nakajima, 2023) (2023) popularized the idea of an "AI agent" that recursively plans and executes sub-goals using an LLM at its core (Shen et al., 2023). While these systems are outside traditional academic publications, they illustrate a trend of wrapping an LM in a loop of plan–act–observe, allowing it to tackle complex, long-horizon tasks. Academic work has explored similar ideas. For instance, *Reflexion* proposes an LLM agent that improves itself through textual self-feedback: after each trial or attempted solution, the agent generates a natural-language reflection on what went wrong or could be improved, stores this in memory,

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

and uses it to inform the next attempt (Shinn et al., 2023). Notably, this is done without gradient updates—the model refines its behavior by reading its own past reflections, a form of verbal self-reinforcement. This procedure led to large gains on tasks like code synthesis by enabling the agent to learn from mistakes in a few-shot manner (Shinn et al., 2023). Another example is *HuggingGPT*, which treats a powerful LM (ChatGPT) as a high-level controller that can delegate subtasks to other specialized models. Given a complex request, HuggingGPT uses the LM to parse the task, generate a structured plan in natural language (identifying which models or tools to use for each subtask), call those models, and then integrate their outputs (Shen et al., 2023). Language serves as the interface between the controller LM and various tools or model "friends," showcasing how an LM's planning can orchestrate an entire tool ecosystem. Broadly, these approaches with learned or explicit controllers (ReAct, AutoGPT (Significant-Gravitas, 2023), Reflexion, etc.) share a motivation with NLEL: they seek to make the model's inference-time behavior more *agentic* and *controlled*, whether through hard-coded loops or by learning policies. NLEL's labeller–tuner architecture can be seen as a principled way to achieve such control, by splitting the reasoning agent into modular roles (one generating natural-language directives, another translating those into execution parameters).

A key aspect of NLEL is its use of *natural-language instructions to modulate model behavior* at each inference step. There is ample prior evidence that LMs can be steered by carefully crafted textual prompts or directives. Kojima et al. (2022) famously showed that simply appending "Let's think step by step" to a query can unlock a latent reasoning mode in GPT-3, turning a zero-shot prompt into a significantly more accurate multi-step solver (Kojima et al., 2022). This highlights that even without any parameter updates, the phrasing of an instruction can trigger qualitatively different behavior from the same model. Follow-up work has generalized this insight into a paradigm of *prompt programming* or *prompt engineering*, where one designs prompts (potentially including fictitious examples, chain-of-thought demonstrations, or high-level directives) to induce the desired problem-solving strategy. For instance, instructing a model to "first outline a plan, then solve the problem" can lead it to generate an explicit plan in natural language and subsequently follow it—effectively using language as an intermediate program. Such techniques illustrate how natural language can function as a flexible control interface for LMs. Beyond prompting alone, researchers have explored methods for dynamic control during decoding. One approach is to adjust decoding parameters or strategies based on the model's uncertainty or the context. Hierarchical search methods like ToT or iterative beam search explore multiple candidate continuations in parallel and use a heuristic or value model to decide which branch to expand. Similarly, guided decoding techniques have been proposed where an auxiliary model or criterion steers the token selection (e.g., biasing the LM away from known incorrect paths or towards factually correct statements). While much of this work is recent and ongoing, the common thread is providing extra guidance signals—often in the form of language—to regulate the LM's generation process in real time. NLEL contributes to this line of research by explicitly generating a natural-language *edge label* (directive) for each reasoning step and mapping it to a set of controllable parameters (via the tuner LM). In effect, it uses a learned textual instruction at each edge to dynamically configure how the next step should be expanded, unifying prompt-based guidance with low-level decoding control.

Finally, our work connects to efforts on leveraging *structured supervision and symbolic scaffolding* to improve model reasoning. Instead of treating the LM as a black-box sequence predictor, these approaches provide intermediate structure or feedback that guides the model towards correct solutions. One example is training LMs with scratchpads: models are asked to "show their work" by outputting intermediate calculations or reasoning steps, which can be compared to ground-truth steps during training (Nye et al., 2021). Even at inference time, the model then tends to produce coherent intermediate justifications, making its reasoning more transparent and often more accurate. This idea was demonstrated by Nye et al. (2021) in tasks like long addition and program execution—when the model is allowed and trained to emit step-by-step computations in a scratchpad, it can handle significantly more complex problems (Nye et al., 2021). Relatedly, incorporating symbolic scaffolds (whether via training or at inference) has shown benefits. PAL's use of a Python interpreter is a prime example: by delegating the actual computation to a symbolic tool, the burden on the language model is reduced and errors can be eliminated or detected (Gao et al., 2023). Even without external tools, one can use verifiers or constraints as a form of scaffold. For instance, Cobbe et al. (Cobbe et al., 2021) (2021) trained a verifier model to judge the correctness of candidate solutions generated by an LM; coupling this verifier with GPT-3 resulted in higher accuracy on math word problems (the original CoT work noted that CoT prompting surpassed even a fine-tuned GPT-3 + verifier system on GSM8K) (Wei et al., 2022). This indicates that having an explicit checker or optimization objective (e.g., consistency with known facts, mathematical validity) can refine the model's outputs by pruning away incorrect reasoning traces. In summary, prior research has explored multiple ways to impose structure on LM reasoning—from supervised intermediate steps to the use of external symbolic systems or auxiliary models for guidance. NLEL aligns with these themes by introducing a structured framework (a labeled tree/DAG with a control-

lable expansion policy) and by using natural-language labels as a form of lightweight symbolic scaffold. Our approach merges the strengths of prompt-based control, learned planning agents, and symbolic guidance: it allows a language model to systematically break down tasks (like CoT), search through alternatives (like ToT), invoke the right operations (like tool-using agents), and tune its generation behavior via instructions—all within a unified, learned *Natural Language Edge Labelling* scheme.

## 3. Preliminaries and Problem Setup

**Reasoning structure.** We model inference as expansion of a directed tree (or a DAG with tie-breaking) $G = (V, E)$. Each node $v \in V$ is a *reasoning step* with textual content $x_v$; the root $v_0$ holds the task statement. Each edge $e = (u \to v) \in E$ carries a natural-language label $L_e$ and induces a control vector $\Pi_e$ used to expand the child $v$. We distinguish two roles: a *labeller* LM $\Lambda$ that proposes edge labels, and a *tuner* LM $\Psi$ that emits control, with mappings

$$L = \Lambda(P, C), \qquad \Pi = \Psi(P, L, C).$$

Here $P$ denotes the parent node text (and any exposed metadata), and $C$ denotes a compact context.

**Context $C$.** We keep $C$ compact and measurable. In our setting, $C$ may include:

- **Frontier uncertainty:** summaries such as the median $\sigma$ across candidate values;
- **Novelty:** nearest-neighbor distances among frontier candidates (embedding or lexical);
- **Depth:** distance from the root;
- **Sibling/frontier summaries:** best $(\mu, \sigma)$ among siblings;
- **Raw label history:** the most recent edge labels as *strings* (from siblings and, optionally, a short frontier window);
- **Budgets:** token usage, retrieval calls, and verification outcomes.

**Control schema $\Pi$.** The tuner controls a task-agnostic set of fields:

- **Decoding:** temperature, top_p, maximum tokens, repetition penalty;
- **Generation:** `gen_count` $\in \mathbb{N}^+$ (bundle size under this label);
- **Search:** branch quota, exploration coefficient $\beta$;
- **Retrieval:** mixture weights $w$ over indices or corpora;
- **Verification:** number and strictness of checks;

Given $\Pi$, a downstream selector (agnostic to NLEL) can use scores such as $S = \mu + \beta\,\sigma$ or a standard Tree-of-Thought (ToT) culling operator.

**Edge labels.** Labels are produced by $\Lambda$ from $(P, C)$.

**Problem instances.** An instance consists of a task $T$, root $v_0$ text, and an evaluation function producing $(\mu, \sigma)$ for partial answers. Unless noted, we treat $G$ as a tree; extension to DAGs is straightforward by merging isomorphic textual states.

**Notation summary.**

| Symbol | Meaning |
|---|---|
| $P$ | parent node content (text + exposed metadata) |
| $L$ | natural-language edge label |
| $C$ | compact context features (bulleted above) |
| $\Lambda$ | labeller LM mapping $(P, C) \to L$ |
| $\Psi$ | tuner LM mapping $(P, L, C) \to \Pi$ |
| $\Pi$ | control vector (decoding, search, retrieval, verification) |
| $\mu, \sigma$ | value / uncertainty estimates used by the selector |
| $w$ | retrieval mixture weights over indices/corpora |
| $\beta$ | exploration coefficient in selection |
| $c_e$, $C_t$ | per-edge and cumulative compute cost |
| `gen_count` | generation bundle size (per edge label) |

## 4. Method

### 4.1. Overview

We propose *Natural Language Edge Labelling* (NLEL), a control layer for structured language-model (LM) reasoning in which each edge carries a natural-language label that specifies *how* the next step should proceed (e.g., "seek a counterexample", "work backward", "apply an anthropological lens; probe for defeaters"). A dedicated *tuner* LM reads a tuple $(P, L, C)$—the parent node $P$, the edge label $L$, and the current context $C$—and maps it directly to a control vector $\Pi$ that configures decoding, search, retrieval, and verification for the next expansion.

### 4.2. Inputs, Outputs, and Mapping

**Inputs.** $P$ is the current parent state (text and optional structure). $L$ is a free-form natural-language directive for the edge. $C$ denotes the remaining state, which can include the partial tree/graph, concise summaries of the frontier and siblings, budget trackers, and verifier configuration.

**Output.** A control vector $\Pi$ whose fields actuate the reasoning stack. A task-agnostic schema can include:

- **Decoding:** temperature, top_p, max_tokens, repetition penalty;
- **Search:** branch quota, exploration coefficient $\beta$;
- **Generation:** number of candidates `gen_count` per label;
- **Retrieval:** mixture weights $w$ over indices or corpora;
- **Verification:** number and strictness of checks.

**Mapping.** Let $\Psi : (P, L, C) \mapsto \Pi$ denote the tuner mapping. In our prompt-only instantiation (Section 4.4), $\Psi$ is realized by a JSON parameter emitter that respects a schema with bounds and learns from a compact in-prompt ledger of historical expansions.

### 4.3. Expansion Procedure

We expand the structure at a parent $p$ in four steps: label emission; bundle generation; selection; and state update.

1. **Emit labels.** Use the labeller to obtain a set of edge labels for $p$: $\mathcal{L}_p = \{L_1, \ldots, L_m\}$, where each $L_i = \Lambda(P, C)$. The number of labels may be governed by a search quota or policy.
2. **Generate bundles under each label.** For each $L \in \mathcal{L}_p$, obtain control $\Pi = \Psi(P, L, C)$ and generate a bundle of `gen_count` candidate children under $L$ using $\Pi$.
3. **Select children (ToT).** Let $\mathcal{B}(L)$ denote the bundle generated under label $L$. Form the union of all candidates for the parent, $\mathcal{C}_p = \bigcup_{L \in \mathcal{L}_p} \mathcal{B}(L)$, and apply the standard ToT child-selection operator to $\mathcal{C}_p$. We inherit ToT's selector as-is.
4. **Update state.** Add survivors to the frontier and update $C$ (budgets, summaries, raw label history strings).

*Notation:* We write $P_p$ for the parent content of node $p$, so mappings like $\Lambda(P, C)$ and $\Psi(P, L, C)$ are instantiated as $\Lambda(P_p, C)$ and $\Psi(P_p, L, C)$ when the current parent is $p$.

---

### 4.4. Prompt-Only JSON Parameter Emitter (JPE)

The tuner LM receives three ingredients in the prompt: (i) a concise *schema* that specifies control fields and bounds; (ii) a *historical ledger* of $(P_i, L_i, C_i) \mapsto \Pi_i$ with outcomes, where rows are tagged as *Pareto* or *dominated* to provide contrastive signals about efficient trade-offs; and (iii) the *current case* $(P, L, C)$. It emits a single JSON object $\Pi$ that must validate against the schema. The ledger can be curated with a lightweight objective that balances task success against compute usage and verification reliability (e.g., success@compute with penalties for excessive tokens or failed checks).

### 4.5. Context Features

To keep $C$ compact and measurable, we surface a small set of features that capture the state of search:

- **Frontier uncertainty:** median $\sigma$ across candidate downstream values (from ensembles, bootstraps, or dropout estimates);
- **Novelty:** median nearest-neighbor distance among frontier candidates (embedding or lexical);
- **Depth:** distance from root (enables exploration annealing and quota schedules);
- **Sibling/frontier summaries:** best $(\mu, \sigma)$ among siblings; raw label history (strings); budget usage.

### 4.6. Downstream Selection (Agnostic to NLEL)

We inherit the standard ToT child-selection operator and apply it once to the union of all candidates produced for a parent (across labels) We use the score $S = \mu + \beta\sigma$ within this ToT culling step..

### 4.7. Stability and Safety

We employ non-intrusive guards: (i) strict schema/bounds validation for emitted JSON; (ii) projection into a trust region around safe defaults to prevent pathological jumps; and (iii) depth-annealed exploration so late-depth expansions remain conservative.

### 4.8. Design Notes

NLEL is compatible with a non-reasoning tuner or a reasoning tuner (e.g., CoT/ToT) used *only* as a controller. The child reasoner can be held fixed to cleanly attribute outcomes to the edge label and the control vector $\Pi$.

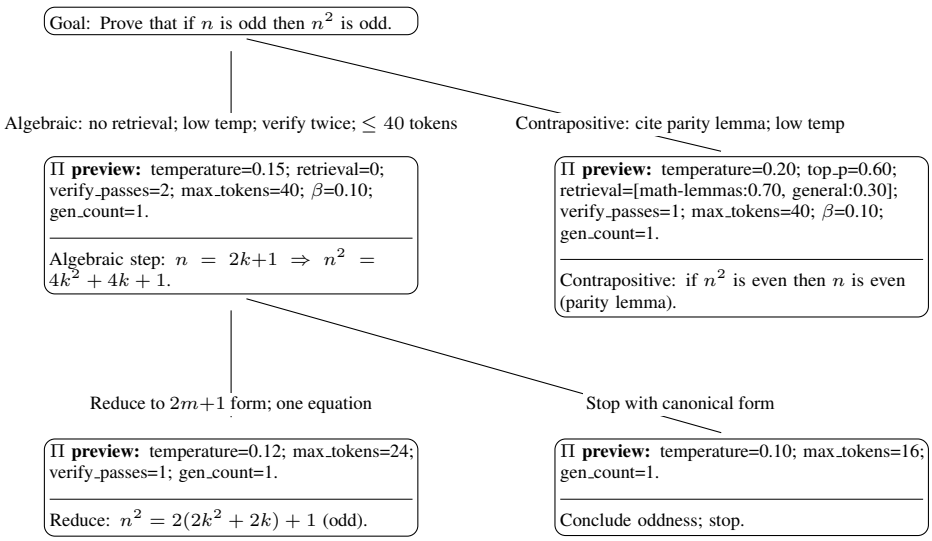


*Figure 1.* A synthetic example of NLEL being used in a ToT setting. For simplicity, gen_count is set to one for all $\Pi_i$.

## 5. Theory (Optional)

## 6. Experiments

## 7. Limitations

## 8. Conclusion

## Impact Statement

## References


Cobbe, K. et al. Training verifiers to solve math word problems. In *Advances in Neural Information Processing Systems*, 2021. Please replace with full, accurate bibliographic details if available.

Gao, L., Madaan, A., Zhou, Y., Alon, U., Liu, P., Yang, Y., and Neubig, G. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2023.

Kojima, T., Gu, S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

Nye, M., Andreas, J., Gur-Ari, G., Tenenbaum, J. B., Lake, B. M., and Linzen, T. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raunak, V., Havrilla, A., Scialom, T., Chen, X., Niehues, J., Schulte, S., Zettlemoyer, L., Schmid, M., and Petroni, F. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Shen, Y., Liu, K., Chen, W., Xu, X., Wang, F., Zhang, Z., Liang, X., Huang, J., Chen, Z., and Zhang, Y. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *arXiv preprint arXiv:2303.17580*, 2023.

Shinn, N., Labash, T., and Madaan, A. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Significant-Gravitas. Autogpt. `https://github.com/Significant-Gravitas/AutoGPT`, 2023. Accessed 2025-10-05.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E. H., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Welleck, S., and Radev, D. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023a.

Yao, S., Zhao, J., Yu, D., Wang, N. D., and Radev, D. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023b.

Yohei Nakajima. Babyagi. `https://github.com/yoheinakajima/babyagi`, 2023. Accessed 2025-10-05.

## A. Additional Experimental Details

## B. Proofs

## C. Extra Results