

Detailed balance in large language model-driven agents

Zhuo-Yang Song ^{1,*} Qing-Hong Cao,^{1,2,†} Ming-xing Luo,^{3,‡} and Hua Xing Zhu^{1,2,§}

¹*School of Physics, Peking University, Beijing 100871, China*

²*Center for High Energy Physics, Peking University, Beijing 100871, China*

³*Beijing Computational Science Research Center, Beijing 100193, China*

Large language model (LLM)-driven agents are emerging as a powerful new paradigm for solving complex problems. Despite the empirical success of these practices, a theoretical framework to understand and unify their macroscopic dynamics remains lacking. This Letter proposes a method based on the least action principle to estimate the underlying generative directionality of LLMs embedded within agents. By experimentally measuring the transition probabilities between LLM-generated states, we statistically discover a detailed balance in LLM-generated transitions, indicating that LLM generation may not be achieved by generally learning rule sets and strategies, but rather by implicitly learning a class of underlying potential functions that may transcend different LLM architectures and prompt templates. To our knowledge, this is the first discovery of a macroscopic physical law in LLM generative dynamics that does not depend on specific model details. This work is an attempt to establish a macroscopic dynamics theory of complex AI systems, aiming to elevate the study of AI agents from a collection of engineering practices to a science built on effective measurements that are predictable and quantifiable.

Introduction. Large language model (LLM)-driven agents are emerging as a powerful new paradigm for solving complex problems [1–15], demonstrating potential in frontier areas such as scientific discovery by combining the generative capabilities of LLMs with external tools and memory systems [16–20]. For instance, FunSearch and AlphaEvolve achieve iterative optimization of solutions by integrating LLMs into evolutionary algorithm frameworks [18, 19]. However, the theoretical understanding and explanation of LLMs often remain at the level of token statistical properties and microscopic generative mechanisms [21–23], making it difficult to explain the macroscopic dynamics of LLMs as complex systems [1, 24]. The behavior of these LLM-driven agents is often viewed as a direct product of their complex internal engineering (such as prompt templates, memory modules, tool calls), and their dynamic characteristics remain a black box [12, 25–27].

The dynamics of LLM generation are quite unique. Compared to traditional rule-based programs, LLM-based generation exhibits diverse and adaptive outputs [16, 17, 25]. At the same time, compared to naive random search, LLM generation shows stronger structure and goal-orientedness [18–20]. Despite the complexity of this hybrid dynamics between random search and deterministic planning, we show that at the agent level (i.e., a coarse-grained description of LLM generative dynamics with standardized agent states as units), LLM generative dynamics exhibit detailed balance similar to equilibrium systems, thereby greatly simplifying the analysis and understanding of LLM generative dynamics [28–31].

To model the dynamic behavior of LLMs, we embed the generative process of LLM within a given agent framework, viewing it as a Markov transition process in its state space [7, 12, 18, 28, 29]. The states are defined by the complete information retained by the agent

at each time step, which may include task objectives, historical summaries, code, file systems, API return values, etc., where the LLM-based generative process is treated as the transition kernel from the current state to a new state. We show that even the states doesn't contain complete historical records, like FunSearch [18], the LLM-based state transitions can still exhibit a directionality towards specific states. We attribute this characteristic to LLMs implicitly learning a potential function V for specific tasks within their vast parameter space, rather than memorizing specific rule sets and strategies. This function evaluates the intrinsic properties of any given state, such as “how far the LLM perceives it to be from the goal”. This global awareness enables LLMs to quickly converge to those optimal states, effectively avoiding repetitive cycles in the state space and achieving stronger generalization capabilities than merely learning strategy sets [32–34].

Based on this model, we propose a method to measure this underlying potential function based on a least action principle [35–37]. By experimentally measuring the transition probabilities between states, we statistically discover a detailed balance in LLM-generated transitions, indicating that LLM generation may not be achieved by generally learning rule sets and strategies, but rather by implicitly learning an underlying potential function that may transcend different LLM architectures and prompt templates. To our knowledge, this is the first discovery of a macroscopic physical law in LLM generative dynamics that does not depend on specific model details. This work is an attempt to establish a macroscopic dynamics theory of complex AI systems, aiming to elevate the study of AI agents from a collection of engineering practices to a predictable and quantifiable science built on effective measurements.

Theory. To formulate the problem rigorously, we con-

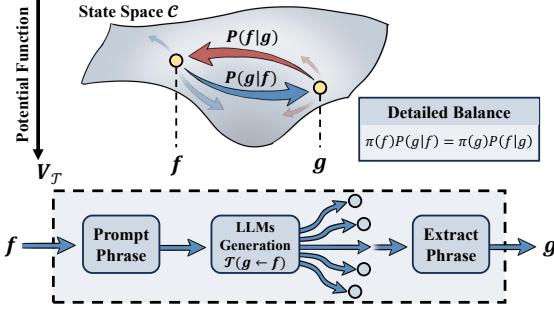


FIG. 1. A schematic of a formalization framework for studying the directionality of LLM generation, illustrating the state space and possible transitions. LLMs are embedded within an agent, which transitions from state f to state g with probability $\mathcal{T}(g \leftarrow f) = P(g|f)$. The underlying potential function V_T quantifies the agent's global ordering of each state, satisfying the detailed balance condition at equilibrium.

sider an agent whose core is comprised of one or more LLMs. The agent takes its current state f as input. Through a series of deterministic steps, it organizes and evaluates this state to generate a relevant prompt. This prompt is then fed into one or more LLMs, whose structured output is parsed to get a new state g . This state is the minimal unit for studying LLM dynamics. This generative process can be viewed as a Markov transition process in the state space \mathcal{C} with a transition kernel $P(g|f)$, retaining the diversity and adaptability of LLM generation. The states are defined by the complete information retained by the agent at each time step, which should include all the information required for the agent to carry out a continuous reasoning or analogical process [18, 20, 38, 39]. In this Letter, the agent contains only a single generation step of LLM, and we denote $\mathcal{T}(g \leftarrow f) = P(g|f)$ as the probability of the agent transitioning from a template containing state f to an output containing state g through LLM generation. A schematic diagram is shown in Fig. 1.

LLM-based agents are characterized by their state transitions not being entirely random but exhibiting a certain structured preference. Specifically, agents tend to transition from the current state f to states g that are “better” from the agent’s perspective. To capture this phenomenon, we hypothesize the existence of an underlying potential function $V_T : \mathcal{C} \rightarrow \mathbb{R}$, which assigns a scalar value to each state, reflecting its “quality”. Since a specific potential function is often difficult to compute directly, we propose a method to effectively estimate the potential function.

Given a global potential function V , we define the violation of the agent’s given transition $\mathcal{T}(g \leftarrow f)$ to the potential function as $K(V(f) - V(g))$, where $K(x)$ is a convex function that describes the extent to which the transition from state f to state g violates the ordering

of the potential function V . To quantify the overall mismatch between the agent’s behavior and the potential function, we weight by the transition kernel $\mathcal{T}(g \leftarrow f)$ and define the action \mathcal{S} as the global average violation:

$$\mathcal{S} = \int_{f \in \mathcal{C}} \int_{g \in \mathcal{C}} \mathcal{T}(g \leftarrow f) K(V(f) - V(g)) Df Dg, \quad (1)$$

where Df, Dg are measures on the state space. In this Letter, we choose $K(x) = \exp(-\beta x/2)$ as the convex function describing the violation of the given state transition from f to g in the ordering of the scalar function V . The action \mathcal{S} or the distribution shape of $\beta V(f)$ can represent the agent’s global cognition ability within this state space \mathcal{C} .

We propose that to quantify the behavior of LLMs using a potential function, one can seek such a potential function that minimizes the overall mismatch between the agent’s transitions and the potential function [36, 37]. Therefore, the most suitable potential function V_T for describing an LLM-based agent \mathcal{T} in a given state space is the one that minimizes the action \mathcal{S} [35, 38, 40, 41].

This implies that the action satisfies the variational principle with respect to the potential function V_T ¹ [35]:

$$\delta \mathcal{S} = 0. \quad (2)$$

The variational condition is equivalent to V_T satisfying the following equilibrium condition:

$$\begin{aligned} & \int_{g \in \mathcal{C}} \mathcal{T}(g \leftarrow f) K'(V_T(f) - V_T(g)) Dg \\ & - \int_{h \in \mathcal{C}} \mathcal{T}(f \leftarrow h) K'(V_T(h) - V_T(f)) Dh = 0, \end{aligned} \quad (3)$$

holding for all $f \in \mathcal{C}$, where $K'(x) = \frac{dK}{dx}$.

Specifically, if for all transitions $\mathcal{T}(g \leftarrow f) > 0$, $V(f) \geq V(g)$ holds, it indicates that the agent’s state transitions are completely ordered, and in this case, V serves as a Lyapunov function [42, 43].

It is worth noting that if \mathcal{T} describes the transition of an equilibrium system, its state transitions satisfy the detailed balance condition, i.e., for all state pairs (f, g) , the following holds [29, 31]:

$$\pi(f)P(g|f) = \pi(g)P(f|g), \quad (4)$$

where $\pi(f)$ denotes the equilibrium distribution of the system at state f , and $P(g|f)$ denotes the transition kernel. In this case, there exists a potential function V that can explicitly express the detailed balance as

$$\log \frac{\mathcal{T}(g \leftarrow f)}{\mathcal{T}(f \leftarrow g)} = \beta V(f) - \beta V(g). \quad (5)$$

¹ We show in Supplemental Material A that the variational principle is equivalent to the least action principle under the condition that $K(x)$ is a convex function.

Substituting into Eq. (3), it can be verified that this potential function $V = V_T$ satisfies the least action principle (see Supplemental Material B). This indicates that for equilibrium systems, if the detailed balance condition exists, the corresponding underlying potential function can be estimated through the least action principle. In general cases, the least action merely seeks the most ordered arrangement of the potential function, minimizing the violations of this arrangement by the agent's state transitions [41].

The main point of this Letter is that we point out that LLM-based agents often behave like an equilibrium system in their LLM-generated state space, which is coarse-grained compared to the complete generation sequence of LLMs [24, 44]. The existence of this phenomenon suggests a universal macroscopic law in LLM generative dynamics that does not depend on specific model and task details. It indicates that despite being seemingly unrelated, there are underlying connections between different LLM generative processes, allowing us to describe the global ordering in LLM generation through the potential function V_T , thereby providing explanations for the internal dynamics of LLMs.

Experiments. We conducted experiments on three different models, including GPT-5 Nano, Claude-4, and Gemini-2.5-flash. Each model was prompted to generate a new word based on a given prompt word such that the sum of the letter indices of the new word equals 100. For example, given the prompt “WIZARD(23+9+26+1+18+4+19=100)”, the model needs to generate a new word whose letter indices also sum to 100, such as “BUZZY(2+21+26+26+25=100)”. The transition kernel between two prompt words can be estimated through sampling as:

$$\mathcal{T}(g \leftarrow f) \approx \frac{N(g \leftarrow f)}{N_0(f)}, \quad (6)$$

where $N(g \leftarrow f)$ denotes the number of times the model generated the word g from the prompt word f . $N_0(f)$ represents the number of sampling attempts starting from the prompt word f . Each model performed 20,000 generations. More details of the experiments are provided in Supplemental Material C.

The three models exhibited two different behaviors, demonstrating directionality and certain diversity in actual LLM generative dynamics. Claude-4 and Gemini-2.5-flash demonstrated rapid convergence, with generated words quickly concentrating on a few high-frequency words. For instance, in 20,000 generations, Claude-4 generated only 5 valid prompt words, while Gemini-2.5-flash generated 13 valid prompt words. In contrast, GPT-5 Nano exhibited stronger exploration, producing as many as 645 different valid prompt words in 20,000 generations. This difference reflects the exploration-exploitation trade-off in LLM generative dynamics, mak-

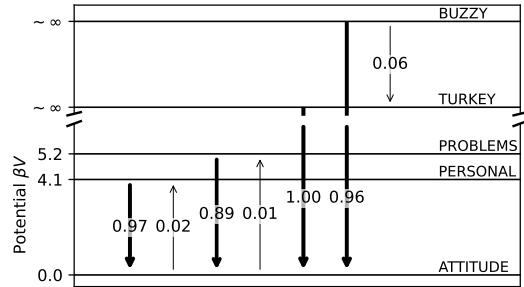


FIG. 2. In the Conditioned Word Generation task, the Claude-4 model exhibits directionality. Transition process of the Claude-4 model in the prompt word state space, ordered by the potential function V_T . Transitions tend to move towards states with lower potentials. States with $\beta V(f) \gg \log(20000) \sim 10$ are those where the equilibrium condition cannot be strictly satisfied; a detailed analysis is provided in Supplemental Material B.

ing them suitable for task scenarios with varying demands for exploration and stability.

In Claude-4 and Gemini-2.5-flash, the solution to the variational condition Eq. (2) can be calculated analytically. In this case, we can plot the transition process of Claude-4 ordered by the potential function as shown in Fig. 2, with specific calculations provided in Supplemental Material B.

Since Claude-4 and Gemini-2.5-flash exhibited high convergence, the equilibrium condition Eq. (3) is almost equivalent to the detailed balance condition Eq. (5). It is worth noting that we observed Claude-4 starting from the prompt word “ATTITUDE”, which has the lowest potential function, began to attempt some invalid words, while Gemini-2.5-flash oscillated between the two lowest potential functions “ATTITUDE” and “DISCIPLINE”, losing exploration. This behavior is similar to the low-temperature trapping phenomenon in physical systems [31, 45], suggesting that controlling the potential function may provide a feasible path to avoid model convergence. A more detailed discussion is provided in Supplemental Material D.

A key example of interest is the GPT-5 Nano model. GPT-5 Nano generated a large number of prompt words due to its strong exploration, allowing us to directly test the detailed balance condition within the state space. We note that according to detailed balance, the sum of potential changes along any closed path should be zero. Specifically, considering a closed path $f_1 \rightarrow f_2 \rightarrow \dots \rightarrow f_n \rightarrow f_1$, according to the detailed balance condition, we have:

$$\sum_{i=1}^n \log \frac{\mathcal{T}(f_{i+1} \leftarrow f_i)}{\mathcal{T}(f_i \leftarrow f_{i+1})} = \sum_{i=1}^n \beta (V(f_i) - V(f_{i+1})) = 0, \quad (7)$$

Fig. 3 counts all triplets in the experimental data, where transitions between each pair of the three data

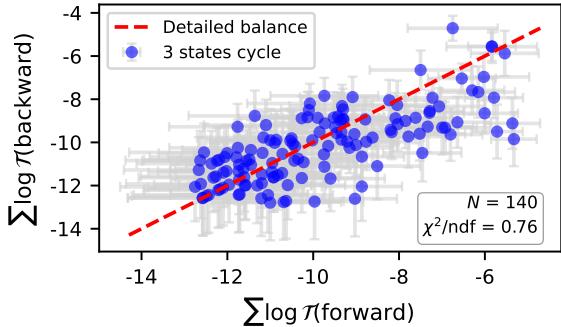


FIG. 3. In the task of Conditioned Word Generation without a reasoning chain, verification of detailed balance through closed paths in the state transition graph of the GPT-5 Nano model. Each point represents a triplet, with all different triplets found in the experimental data. The error bars directly arise from sampling errors.

TABLE I. Examples of some states of the agent $\mathcal{T}_{\text{real}}$ and their potentials. The independent variable $\log_{\text{v.k.nu}}$ in the fitting task is abbreviated as x .

states f	Potential
$\text{param1} * \tanh(\text{param2} * x + \text{param3}) + \text{param4}$	5.70
$\text{param1} - (\text{param2} / (x + \text{param3}))$	0.88
$\text{param1} * x / (1 + \text{param2} * \log(x + 1))$	-0.57
$\text{param1} * \tanh(\text{param2} * x) + \text{param3}$	-1.57
$\text{param2} + \text{param1} * (1 - \exp(-x))$	-3.30

points were measured, totaling 140 different triplets. Each point represents a comparison of the sum of the logarithms of the forward and reverse transition kernels for a triplet, thereby verifying the detailed balance condition. The measurement points cluster around the diagonal line, indicating that within the error range, the two sums are approximately equal, consistent with the detailed balance condition.

To further validate the universality of detailed balance in LLM generation, we now construct an agent $\mathcal{T}_{\text{real}}$ with a long reasoning chain, whose states are strings that can be parsed into specific expression trees (implementation details are provided in Supplemental Material C). We recorded 50,228 state transitions executed by this agent, constructing a database containing 21,697 different transitions and 7,484 different states. Some example states are shown in Table I. By analyzing these transitions, we can statistically verify the detailed balance condition and estimate its underlying potential function $V_{\mathcal{T}_{\text{real}}}$ through the least action principle. This agent involves multiple different LLMs and prompt templates, so its potential function characteristics may reflect typical behaviors of LLM generation in practical applications. More experimental details are provided in Supplemental Material C.

Similar to the measurement described above, we

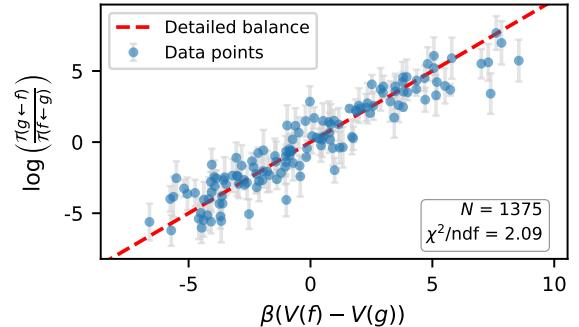


FIG. 4. In the task of Symbolic Fitting with a long reasoning chain, verification of the detailed balance condition for the agent $\mathcal{T}_{\text{real}}$. The error estimates only include root mean square statistical errors, excluding unknown systematic errors. Measurement includes state pairs with at least one measured transitions between the two states. Points with $|V_{\mathcal{T}}(f) - V_{\mathcal{T}}(g)| > \log 50288$ are excluded. One-tenth of the data points are displayed for clarity.

counted all triplets in the experimental data, where transitions between each pair of the three data points were measured, and for all different triplets, we observed the establishment of the detailed balance condition within the measurement error range. The results are provided in Supplemental Material B.

To further validate detailed balance, we now estimate the underlying potential function $V_{\mathcal{T}_{\text{real}}}$ through the least action principle. In a discrete state space, the integrals are replaced by sums over states, so the action (Eq. (1)) becomes, normalized by the number of states in the database:

$$S = \frac{\sum_{g \leftarrow f} K(V_{\mathcal{T}_{\text{real}}}(f) - V_{\mathcal{T}_{\text{real}}}(g))}{\sum_f 1}. \quad (8)$$

By numerically minimizing this action, we can estimate the potential function values $V_{\mathcal{T}_{\text{real}}}(f)$ for each state. According to this estimation, the minimum value of the action is much smaller than $K(0)$, indicating that the state transitions of the agent $\mathcal{T}_{\text{real}}$ indeed exhibit directionality. In Supplemental Material D, we show that this optimized action value can be used to quantify the density distribution of states with respect to the underlying potential function, thereby providing direct guidance for designing more effective LLM generation strategies in practice.

By estimating this potential function, we can verify whether the potential function is consistent with the detailed balance condition. Specifically, the detailed balance condition requires that for all state pairs (f, g) , Eq. (5) holds.

Fig. 4 shows a comparison of the left and right sides of Eq. (5), indicating that the detailed balance condition is largely satisfied.

With the estimation of the potential function, we further discuss the specific meaning of the potential function

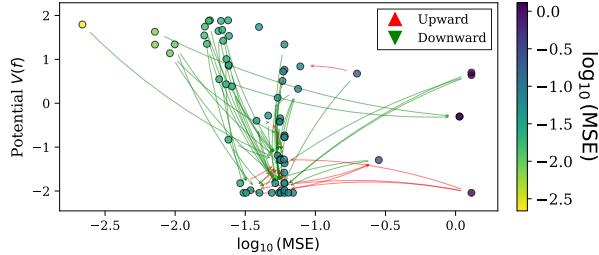


FIG. 5. illustration of the ability of the potential function discovered using IdeaSearch [46] to predict the directionality of state transitions. Each point represents a transition pair from state f to state g . The figure shows a subgraph composed of 70 states selected from the database, displaying transitions with high transition kernel $\mathcal{T}_{\text{real}}(g \leftarrow f) > 0.05$. Red and green lines represent transitions with increasing and decreasing potential functions, respectively. The horizontal axis represents the mean square error of the expression corresponding to the state in the symbolic fitting task, while the vertical axis represents the potential function.

in this problem to reveal the intrinsic cognitive characteristics in LLM generative dynamics. To this end, we use the action as an optimization objective and employ a workflow based on IdeaSearch [46] to find a potential function with an explicit functional form that maps the expression strings corresponding to state f to scalar potential function values $V_{\mathcal{T}}(f)$. The best potential function found in 4000 rounds of search contains 49 parameters, capturing various features of state f at the expression level, such as complexity, syntactic validity, and structural affinity with domain-specific patterns, without capturing string-level information. The magnitude of the corresponding parameter values directly reflects the importance that LLMs attach to these features during the generation process. The potential function and its specific analysis are provided in Supplemental Material E.

Fig. 5 shows the transition patterns between some states sorted by this potential function. In the database, there are a total of 9,769 transitions with high transition kernel $\mathcal{T}_{\text{real}}(g \leftarrow f) > 0.05$, and the corresponding potentials $V(f), V(g)$ are calculated using the potential function. Among them, 6,795 (69.56%) exhibit a decrease in the potential function, 2,523 (25.83%) exhibit an increase in the potential function, and 451 (4.62%) exhibit no change in the potential function. This indicates that the potential function partially captures the intrinsic directionality in LLM generative dynamics. It is worth emphasizing that LLMs overall tend to choose states with relatively low potential function values as the next state, even though they may not necessarily perform better on actual data. In this way, the potential function can reveal the differences between the intrinsic cognition of LLMs and real data.

Conclusion and Outlook. In this Letter, we have proposed a framework based on the least action principle

to describe and analyze the generative dynamics of LLM-based agents in their LLM-generated state space. Through experimental validation on multiple different models and tasks, we have found that the state transitions of these agents largely satisfy the detailed balance condition, indicating that their generative dynamics exhibit characteristics similar to equilibrium systems. We have further estimated the underlying potential function through the least action principle and revealed its important role in capturing the intrinsic directionality in LLM generative dynamics.

This Letter provides a preliminary exploration of the possibility for discovering macroscopic laws in LLMs generative dynamics. Future work can further expand this framework and explore the application potential of more tools from equilibrium and near-equilibrium systems in understanding and optimizing LLM generation processes. For instance, studying the degree of deviation from equilibrium may help us understand a model's level of overfitting, as overfitted models may learn more localized strategy sets rather than global generative patterns governed by potential functions [33, 34]. Additionally, optimization methods based on potential functions may also provide new ideas for improving the quality and diversity of LLM task-related generation, such as adjusting the action to different magnitudes based on varying safety and exploration requirements.

Acknowledgements. We would like to thank Zeyu Cai, Jiashen Wei, Shi Qiu, Shutao Zhang, Jichen Pan and Zikang Lin for useful discussions. This work is supported by National Natural Science Foundation of China under contract No. 12425505, 12235001, U2230402.

Data and Code Availability. The code used to perform the analysis in this Letter is publicly available on GitHub at <https://github.com/SonnyNondegeneracy/detailed-balance-lm> under the MIT License. The data discussed in this Letter are available on Hugging Face at <https://huggingface.co/datasets/Nondegeneracy/detailed-balance-lm> under the Creative Commons Attribution 4.0 (CC BY 4.0) license.

* zhuoyangsong@stu.pku.edu.cn

† qinghongcao@pku.edu.cn

‡ mingxingluo@csrc.ac.cn

§ zhuhx@pku.edu.cn

- [1] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, in *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22* (Curran Associates Inc., Red Hook, NY, USA, 2022).
- [2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, React: Synergizing reasoning and acting in language models (2023), arXiv:2210.03629 [cs.CL].
- [3] 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 (2023), arXiv:2305.10601 [cs.CL].
- [4] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, Self-consistency improves chain of thought reasoning in language models (2023), arXiv:2203.11171 [cs.CL].
- [5] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models (2023), arXiv:2305.04091 [cs.CL].
- [6] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, Toolformer: Language models can teach themselves to use tools (2023), arXiv:2302.04761 [cs.CL].
- [7] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, Voyager: An open-ended embodied agent with large language models (2023), arXiv:2305.16291 [cs.AI].
- [8] W. Chen, X. Ma, X. Wang, and W. W. Cohen, Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks (2023), arXiv:2211.12588 [cs.CL].
- [9] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, in *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 202 (PMLR, 2023) pp. 10764–10799.
- [10] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, Code as policies: Language model programs for embodied control (2023), arXiv:2209.07753 [cs.RO].
- [11] D. J. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, and A. e. a. Ahern, *Nature* **618**, 257 (2023).
- [12] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23 (Association for Computing Machinery, New York, NY, USA, 2023).
- [13] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gaminazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczek, and T. Hoeffler, *Proceedings of the AAAI Conference on Artificial Intelligence* **38**, 17682 (2024).
- [14] M. Mohammadi, Y. Li, J. Lo, and W. Yip, in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25 (ACM, 2025) p. 6129–6139.
- [15] R. Sapkota, K. I. Roumeliotis, and M. Karkee, Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges (2025), arXiv:2505.10468 [cs.AI].
- [16] D. A. Boiko, R. MacKnight, B. Kline, and G. Gomes, *Nature* **624**, 570 (2023).
- [17] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller, Chemcrow: Augmenting large-language models with chemistry tools (2023), arXiv:2304.05376 [physics.chem-ph].
- [18] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi, *Nature* **625**, 468 (2024).
- [19] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. R. Ruiz, and A. M. et. al., Alphaevolve: A coding agent for scientific and algorithmic discovery (2025), arXiv:2506.13131 [cs.AI].
- [20] Z.-Y. Song, Z. Cai, S. Zhang, J. Wei, J. Pan, S. Qiu, Q.-H. Cao, T.-J. Hou, X. Liu, M. xing Luo, and H. X. Zhu, Iterated agent for symbolic regression (2025), arXiv:2510.08317 [physics.comp-ph].
- [21] N. Bhattacharya, N. Thomas, R. Rao, J. Dauparas, P. K. Koo, D. Baker, Y. S. Song, and S. Ovchinnikov, in *Pacific Symposium on Biocomputing*, Vol. 27 (World Scientific, Singapore, 2022) pp. 34–45.
- [22] Y. Sun and B. Haghight, Phase transitions in large language models and the $o(n)$ model (2025), arXiv:2501.16241 [cs.LG].
- [23] Z. Liu, Y. Liu, J. Gore, and M. Tegmark, Neural thermodynamic laws for large language model training (2025), arXiv:2505.10559 [cs.LG].
- [24] E. Hoel, *Entropy* **19** (2017).
- [25] J. J. Hopfield, *Proceedings of the National Academy of Sciences* **79**, 2554 (1982), <https://www.pnas.org/doi/pdf/10.1073/pnas.79.8.2554>.
- [26] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, *Transactions on Machine Learning Research* (2022), survey Certification.
- [27] R. Schaeffer, B. Miranda, and S. Koyejo, in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23 (Curran Associates Inc., Red Hook, NY, USA, 2023).
- [28] A. A. Markov (2006) uRL <https://api.semanticscholar.org/CorpusID:126339706>.
- [29] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *The Journal of Chemical Physics* **21**, 1087 (1953).
- [30] M. Mézard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond* (World Scientific, Singapore, 1987).
- [31] D. V. Schroeder, *An introduction to thermal physics* (Oxford University Press, 2020).
- [32] R. S. Sutton, A. G. Barto, et al., *Reinforcement Learning: An Introduction*, Vol. 1 (MIT Press, Cambridge, MA, 1998).
- [33] N. Tishby and N. Zaslavsky, 2015 IEEE Information Theory Workshop, ITW 2015 (2015).
- [34] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, ArXiv [abs/1609.04836](https://arxiv.org/abs/1609.04836) (2016).
- [35] J. Safko, H. Goldstein, and C. Poole, *Classical mechanics* (2002).
- [36] K. Friston, *Nature Reviews Neuroscience* **11**, 127 (2010).
- [37] A. Tschantz, M. Baltieri, A. K. Seth, and C. L. Buckley, in *2020 International Joint Conference on Neural Networks (IJCNN)* (2020) pp. 1–8.
- [38] D. B. West, *Introduction to graph theory* (2001).
- [39] M. Yasunaga, X. Chen, Y. Li, P. Pasupat, J. Leskovec, P. Liang, E. H. Chi, and D. Zhou, Large language models as analogical reasoners (2024), arXiv:2310.01714 [cs.LG].
- [40] T. Joachims, in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02 (Association for Computing Machinery, New York, NY, USA, 2002) p. 133–142.
- [41] X. Jiang, L.-H. Lim, Y. Yao, and Y. Ye, *Mathematical Programming* **127**, 203 (2011).
- [42] A. M. Lyapunov, *International journal of control* **55**, 531 (1992).

- [43] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering* (CRC press, 2015).
- [44] S. Weinberg, Physica A **96**, 327–340 (1979).
- [45] J. C. Schön, Journal of Physics A: Mathematical and General **30**, 2367 (1997).
- [46] I. Collaboration, Ideasearch, <https://github.com/IdeaSearch/IdeaSearch-fit> (2025).
- [47] I. Collaboration, Ideasearchfitter, <https://github.com/IdeaSearch/IdeaSearch-framework> (2025).
- [48] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, BioData mining **10**, 36 (2017).
- [49] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, and S. e. a. Pfohl, Nature **620**, 172 (2023).
- [50] N. J. Szymanski, B. Rendy, Y. Fei, R. E. Kumar, T. He, D. Milsted, M. J. McDermott, M. Gallant, E. D. Cubuk, A. Merchant, H. Kim, A. Jain, C. J. Bartel, K. Persson, Y. Zeng, and G. Ceder, Nature **624**, 86 (2023).
- [51] Z.-Y. Song, T.-Z. Yang, Q.-H. Cao, M. xing Luo, and H. X. Zhu, Explainable ai-assisted optimization for feynman integral reduction (2025), arXiv:2502.09544 [hep-ph].
- [52] Q.-H. Cao, Z.-Y. Hou, Y.-Y. Li, X. Liu, Z.-Y. Song, L.-Q. Zhang, S. Zhang, and K. Zhao, Scalable quantum state preparation via large-language-model-driven discovery (2025), arXiv:2505.06347 [quant-ph].

SUPPLEMENTAL MATERIAL FOR “DETAILED BALANCE IN LARGE LANGUAGE MODEL-DRIVEN AGENTS”

A. EQUIVALENCE OF THE LEAST ACTION PRINCIPLE AND THE VARIATIONAL CONDITION

This supplementary material proves the equivalence of the variational condition Eq. (2) and the least action principle when $K(x)$ is a convex function. The proof is conducted in the discrete case.

Firstly, we prove that the variational condition is a necessary condition for the least action.

Proof 1 Assume that when the least action is satisfied, $V_{\mathcal{T}}$ does not satisfy the variational condition Eq. (2), then there exists at least one state f_0 such that

$$\frac{\delta \mathcal{S}}{\delta V(f_0)} \Big|_{V=V_{\mathcal{T}}} \neq 0. \quad (9)$$

This implies that there must exist another potential function $V'_{\mathcal{T}}$ defined as

$$V'_{\mathcal{T}}(f) = V_{\mathcal{T}}(f) + \epsilon \delta_{f,f_0} \frac{\delta \mathcal{S}}{\delta V(f_0)} \Big|_{V=V_{\mathcal{T}}}, \quad (10)$$

where δ_{f,f_0} is the Kronecker delta, and ϵ is a sufficiently small constant. Expanding the action \mathcal{S} around it yields

$$\begin{aligned} \mathcal{S}[V'_{\mathcal{T}}] &= \mathcal{S}[V_{\mathcal{T}}] + \sum_f \frac{\delta \mathcal{S}}{\delta V(f)} \Big|_{V=V_{\mathcal{T}}} (V'_{\mathcal{T}}(f) - V_{\mathcal{T}}(f)) + O(\epsilon^2) \\ &= \mathcal{S}[V_{\mathcal{T}}] + \frac{\delta \mathcal{S}}{\delta V(f_0)} \Big|_{V=V_{\mathcal{T}}} (V'_{\mathcal{T}}(f_0) - V_{\mathcal{T}}(f_0)) + O(\epsilon^2) \\ &= \mathcal{S}[V_{\mathcal{T}}] + \epsilon \left(\frac{\delta \mathcal{S}}{\delta V(f_0)} \Big|_{V=V_{\mathcal{T}}} \right)^2 + O(\epsilon^2). \end{aligned} \quad (11)$$

By choosing a sufficiently small $\epsilon > 0$, the first-order term dominates, and due to Eq. (9), we have

$$\left(\frac{\delta \mathcal{S}}{\delta V(f_0)} \Big|_{V=V_{\mathcal{T}}} \right)^2 > 0, \quad (12)$$

Thus,

$$\mathcal{S}[V'_{\mathcal{T}}] < \mathcal{S}[V_{\mathcal{T}}]. \quad (13)$$

This contradicts the assumption that $V_{\mathcal{T}}$ is the least action, thus completing the proof.

Next, we prove that when $K(x)$ is a convex function, the potential function $V_{\mathcal{T}}$ satisfying the variational condition is necessarily a global minimum point of the action \mathcal{S} . This can be reduced to proving that the action \mathcal{S} is a convex functional.

Proof 2 Let V_1, V_2 be any two potential functions, and $0 \leq \lambda \leq 1$, then

$$\begin{aligned} &\mathcal{S}[\lambda V_1 + (1 - \lambda)V_2] \\ &= \sum_{f,g} K(\lambda V_1(f) + (1 - \lambda)V_2(f) - \lambda V_1(g) - (1 - \lambda)V_2(g)) \mathcal{T}(g \leftarrow f) \\ &\leq \sum_{f,g} \lambda K(V_1(f) - V_1(g)) + (1 - \lambda) K(V_2(f) - V_2(g)) \mathcal{T}(g \leftarrow f) \\ &= \lambda \mathcal{S}[V_1] + (1 - \lambda) \mathcal{S}[V_2], \end{aligned} \quad (14)$$

where the inequality arises from the convexity of $K(x)$ and the positivity of $\mathcal{T}(g \leftarrow f)$, thus completing the proof.

In summary, when $K(x)$ is a convex function, the variational condition Eq. (2) and the principle of least action are equivalent.

From \ To	ATT.	TUR.	PER.	PRO.	BUZ.	escape
ATT.	0	0	66	20	0	3914
TUR.	4122	0	0	0	0	0
PER.	3879	0	0	0	0	121
PRO.	3558	0	0	0	0	442
BUZ.	3859	238	0	0	0	0

TABLE II. Transition kernel $\mathcal{T}(g \leftarrow f) = \min(N(g \leftarrow f)/4000, 1)$ for the Claude-4 model, where ATT., TUR., PER., PRO., and BUZ. represent the states ATTITUDE, TURKEY, PERSONAL, PROBLEM, and BUZZY, respectively. Each row represents the number of transitions starting from state f , and each column represents the number of transitions to state g . The reason for “escape” is that some transitions are rejected because they are not words or the sum of letters is not 100, or the generated word is still the prompt word, especially for transitions starting from the ATTITUDE state.

B. THE LEAST ACTION PRINCIPLE FOR EXTREME AGENTS

This Supplemental Material proves that the detailed balance condition Eq. (5) is a sufficient condition for the variational principle. It then introduces some techniques for analytically calculating the minimum value of the action, based on which the potential function distributions of the Claude-4 are analyzed.

Detailed balance condition is a sufficient condition for the variational principle

Proof 3 Assume that the agent \mathcal{T} satisfies the detailed balance condition Eq. (5), then for any state pair (f, g) , we have

$$\mathcal{T}(f \leftarrow g) = \mathcal{T}(g \leftarrow f) e^{-\beta(V_{\mathcal{T}}(f) - V_{\mathcal{T}}(g))}. \quad (15)$$

Therefore, substituting this relation into the equilibrium condition Eq. (3), we obtain

$$\frac{1}{2} \sum_{f,g} \left[K'(V_{\mathcal{T}}(f) - V_{\mathcal{T}}(g)) - K'(V_{\mathcal{T}}(g) - V_{\mathcal{T}}(f)) e^{-\beta(V_{\mathcal{T}}(f) - V_{\mathcal{T}}(g))} \right] \mathcal{T}(g \leftarrow f) = 0. \quad (16)$$

Substituting the derivative of $K(x)$ into the above equation, the result is naturally satisfied.

It's worth noting that any $K(x)$ function satisfying

$$K'(x) - K'(-x) e^{-\beta x} = 0, \quad (17)$$

makes the detailed balance condition a sufficient condition for the variational principle. There are no specific requirements for the form of $K(x)$ here.

Analytically calculating the minimum action value

Taking the Claude-4 model as an example, we demonstrate how to analytically calculate the minimum action value to analyze the distribution of the potential function. Since only 5 different prompt words are involved, a relatively accurate estimation method is $N_0(f) = 20000/5 = 4000$, thus estimating the transition kernel $\mathcal{T}(g \leftarrow f)$ as

$$\mathcal{T}(g \leftarrow f) \approx \frac{N(g \leftarrow f)}{N_0(f)} \approx \min \left(\frac{N(g \leftarrow f)}{4000}, 1 \right). \quad (18)$$

where $N(g \leftarrow f)$ is the number of transitions from state f to state g as shown in the table II. We excluded self-loop transitions ($f = g$) and states that were recorded only once ($\sum_{g'} N(g' \leftarrow f) \leq 1$). We also indicate that transitions starting from a given state may “escape” for more detailed discussion, i.e., $\mathcal{T}(\text{escape} \leftarrow f) = 1 - \sum_g \mathcal{T}(g \leftarrow f)$. Here, $N(g \leftarrow f)$ represents the number of transitions from state f to state g .

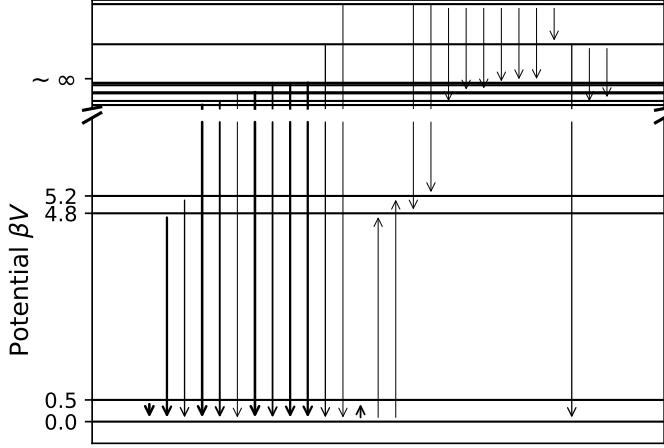


FIG. 6. Transition process of the Gemini-2.5-flash model in the prompt word state space, sorted by the potential function V_T . Transitions tend to move towards states with lower potential functions. States with $\beta V(f) \gg \log(20000) \sim 10$ are those where the equilibrium condition cannot be strictly satisfied. Thick lines represent high-frequency transitions, while thin lines represent low-frequency transitions. Each horizontal line represents a state, arranged in order of increasing potential function.

Now, setting the zero point of the potential function as $V_T(\text{ATT.}) = 0$, for the state PERSONAL, which has transitions only with the ATTITUDE state, the equilibrium condition Eq. (3) degenerates into the detailed balance condition, yielding

$$\beta V_T(\text{PER.}) \approx \log \frac{\mathcal{T}(\text{PER.} \leftarrow \text{ATT.})}{\mathcal{T}(\text{ATT.} \leftarrow \text{PER.})} = \log \frac{66/4000}{3879/4000} \approx 4.1. \quad (19)$$

Similarly, for the state PROBLEM, we have $V_T(\text{PRO.}) \approx 5.2$. For the state BUZZY, there are no transitions into it, meaning that its potential function value cannot be estimated; it can only be judged that it should be much greater than $\sim \log(20000) \sim 10$, which should be the maximum range for accurate measurement of the potential function.

For the state TURKEY, there are two transitions: one from ATTITUDE to TURKEY and another from BUZZY to TURKEY. Since it has been assumed that $V_T(\text{BUZZY}) \rightarrow \infty$, the equilibrium condition Eq. (3) simplifies to

$$K'(\beta V_T(\text{TUR.}) - \beta V_T(\text{ATT.}))\mathcal{T}(\text{TUR.}, \text{ATT.}) = K'(\beta V_T(\text{BUZ.}) - \beta V_T(\text{TUR.}))\mathcal{T}(\text{BUZ.}, \text{TUR.}), \quad (20)$$

thus obtaining

$$\beta V_T(\text{TUR.}) \rightarrow \infty. \quad (21)$$

In summary, the potential function of the Claude-4 model is approximately

$$\beta V_T(f) \approx \begin{cases} 0, & f = \text{ATT.}, \\ 4.1, & f = \text{PER.}, \\ 5.2, & f = \text{PRO.}, \\ \sim \infty, & f = \text{BUZ. or TUR.}, \end{cases}. \quad (22)$$

This is consistent with the results in Fig. 2. It is worth noting that in this case, the potential function is almost self-consistently derived directly from the results of the detailed balance condition.

Similarly, analyzing the Gemini-2.5-flash model yields the transition map shown in Fig. 6.

The potential function of Gemini-2.5-flash is approximately

$$\beta V_T(f) \approx \begin{cases} 0, & f = \text{ATTITUDE}, \\ 0.5, & f = \text{DISCIPLINE}, \\ 4.8, & f = \text{EXCELLENT}, \\ 5.2, & f = \text{BLISSFUL}, \\ \sim \infty, & f = \text{others}, \end{cases}. \quad (23)$$

TABLE III. Database statistics for the two agents, including transition sampling counts, unique state counts, unique transition counts, and the number of states with sampling times greater than 1.

Agent	Transition Samples	Unique States	Unique Transitions	States with Samples > 1
IdeaSearchFitter	50228	7484	21697	2551
Conditioned Word Generation (GPT5-Nano)	19968	645	9473	620

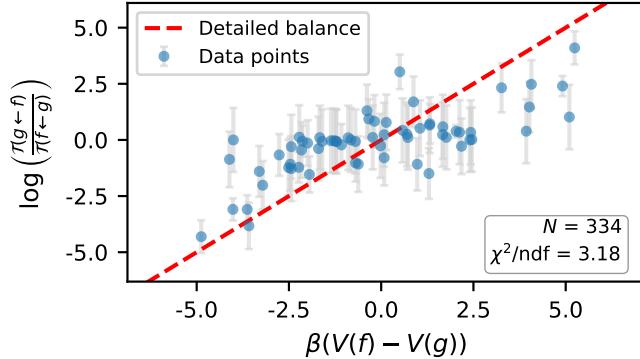


FIG. 7. Verification of the detailed balance condition for the Conditioned Word Generation Agent using GPT5-Nano model. The error estimates only include root mean square statistical errors, excluding unknown systematic errors. Measurement includes state pairs with at least one measured transitions between the two states. The agent’s underlying potential function is consistent with the detailed balance condition, with systematic deviations from detailed balance observed at higher potential function values. Points with $|V_T(f) - V_T(g)| > \log 20000$ are excluded. One-fifth of the data points are displayed for clarity.

C. IMPLEMENTATION OF AGENT

To validate the detailed balance proposed in this Letter, LLMs need to be embedded into an agent framework to standardize their state space and state transitions. This Supplemental Material describes the implementation details of two completely different agent frameworks and supplements more experimental results, including the examination of detailed balance through the potential function for the Constrained Word Generation Agent using GPT5-Nano model and the direct examination of Eq. (7) in the **IdeaSearchFitter** Agent. For the Constrained Word Generation Agent using GPT5-Nano model and the **IdeaSearchFitter** Agent, which involve more states, a relatively appropriate estimate is to ignore “escape” and directly take $N_0(f) = \sum_g N(g \leftarrow f)$. To achieve a more accurate evaluation, we filter out those states f for which $\sum_g N(g \leftarrow f) \leq 1$, as shown in Eq. (24).

The transition sampling counts, unique state counts, unique transition counts, and the number of states with sampling times greater than 1 for the two agents are shown in Table III. The code used to construct the Agents below can be found in the GitHub repository.

Conditioned Word Generation Agent

To examine the generative dynamics of the model, we constructed an agent $\mathcal{T}_{\text{real,I}}$ based on a conditioned word generation task. This agent generates a word through an LLM, requiring that the sum of the indices corresponding to all letters in the word equals 100 (for example, ATTITUDE, EXCELLENT). The state space of this agent consists of all words that satisfy this condition, and the large models used in state transitions are GPT5-Nano, Claude-4, and Gemini-2.5-flash, which read the context containing prompts and given states to generate new words that meet the condition. The specific implementation can be found in the GitHub repository.

The implementation of the Conditioned Word Generation Agent shows two different behavioral patterns, with the Claude-4 and Gemini-2.5-flash models exhibiting significant convergence, while the GPT5-Nano model demonstrates broader exploration capabilities. Fig. 7 presents the results of verifying the detailed balance condition for the Conditioned Word Generation Agent using the GPT5-Nano model.

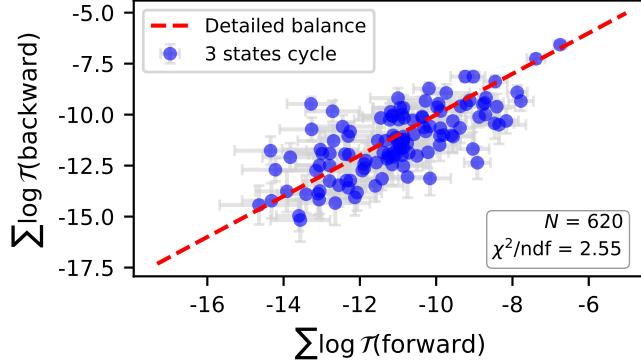


FIG. 8. Verification of detailed balance through closed paths in the state transition graph of the complex agent $\mathcal{T}_{\text{real}}$. Each point represents a triplet, with a total of 620 different triplets found in the experimental data, where each transition was detected at least twice. This indicates that within the error range, the sums of the logarithms of the forward and backward transition kernels are roughly equal, consistent with the detailed balance condition. To clearly display the figure, only 1/5 of the data points are shown.

IdeaSearchFitter Agent

To examine the performance of LLM generative dynamics in specific tasks, we constructed an agent $\mathcal{T}_{\text{real}}$ based on the symbol fitting task using **IdeaSearchFitter** [47]. The state space of this agent consists of strings represented as expression trees f , and state transitions are achieved by generating new expression trees through LLMs. The agent runs in expert mode 10 times to obtain the database used in the main text; specifically, “example_num” is set to 1 to simplify the state space to numexpr strings, and “auto_polish” is set to True to test with richer prompts. “sample_temperature” and “model.sample_temperature” are set to 1000.0 to uniformly sample the state space. Each run searches the “nikuradse_2” dataset from PMLB [48] without early stopping conditions. The final dataset contains 50,228 state transitions, involving 21,697 unique transitions and 7,484 unique states, of which 2,551 states were sampled more than once. The implementation can be found in the GitHub repository.

Within this Agent, the method for estimating the transition kernel is:

$$\mathcal{T}(g \leftarrow f) = \begin{cases} \frac{N(g \leftarrow f)}{\sum_{g' \neq f} N(g' \leftarrow f)}, & \sum_{g'} N(g' \leftarrow f) > 1 \text{ and } g \neq f, \\ 0, & \text{otherwise.} \end{cases} \quad (24)$$

We excluded self-loop transitions ($f = g$) and states that were recorded only once ($\sum_{g'} N(g' \leftarrow f) \leq 1$) to make the estimation of the transition kernel more accurate. Here, $N(g \leftarrow f)$ represents the number of transitions from state f to state g .

In the main text, we examined the detailed balance condition of the **IdeaSearchFitter** agent by directly comparing the differences in its potential function and the logarithm of the transition kernel ratios. To further validate detailed balance, we also verified it through closed paths in its state transition graph. Specifically, we searched for all possible triplets (f, g, h) in the experimental data such that transitions exist between each pair. For each triplet, we calculated the sums of the logarithms of the forward and backward transition kernels along the closed path, with the results shown in Fig. 8. A total of 620 different triplets were found in the experimental data, indicating that within the error range, the sums of the logarithms of the forward and backward transition kernels are roughly equal, consistent with the detailed balance condition.

Next, we demonstrate that even when changing the specific form of $K(x)$, as long as it satisfies Eq. (17), the potential function distribution consistent with detailed balance can still be recovered through the principle of least action. Fig. 9 shows the results when using a common function form in transition dynamics, $K(x) = \log(1 + e^{-\beta x})$. It can be seen that the potential function distribution is basically consistent with the results obtained using $K(x) = e^{-\beta x/2}$ in the main text, further supporting the reasonableness of the detailed balance condition.

Finally, in order to further validate the reasonableness of detailed balance, we discuss such pairs where the transition $g \leftarrow f$ was not measured, while the transition $f \leftarrow g$ was measured. Using the same notation conventions as in the

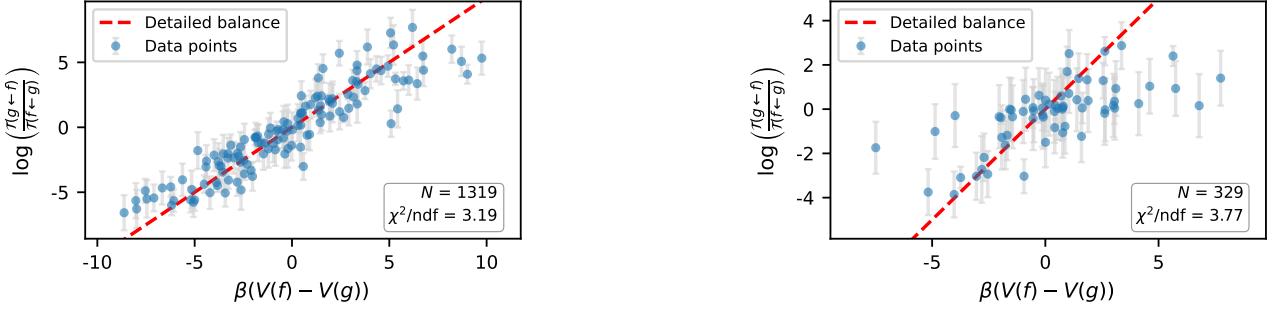


FIG. 9. Verification of the detailed balance condition for both agents using $K(x) = \log(1 + e^{-x})$, with all settings the same as in the main text. (a) Results for the **IdeaSearchFitter** agent. (b) Results for the Conditioned Word Generation Agent.

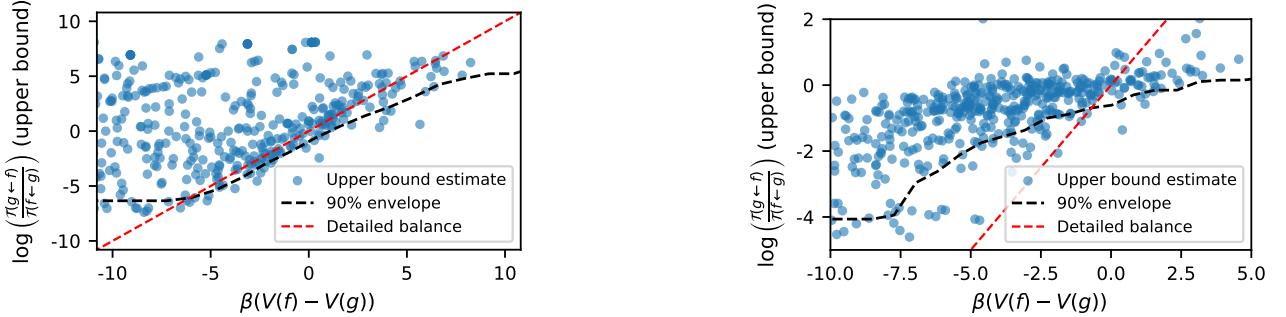


FIG. 10. For pairs in the **IdeaSearchFitter** and Conditioned Word Generation Agent where the transition $f \leftarrow g$ was measured but the transition $g \leftarrow f$ was not, we compare the differences in their potential functions and the logarithm of the transition kernel ratios. The figure shows points corresponding to 500 such pairs. The black dashed line represents the 90th percentile line. It is mostly above the red dashed line representing detailed balance, indicating that the inequality Eq. (25) is basically satisfied, with those points at larger $|\beta V_T(f) - \beta V_T(g)|$ possibly arising from systematic overestimation of the potential function differences. (a) Results for the **IdeaSearchFitter** agent with a total of 18,935 such pairs. (b) Results for the Conditioned Word Generation Agent with a total of 8,805 such pairs. From this figure, it can be directly seen that most of such transition pairs come from cases where $N(f \leftarrow g) = 1$.

main text, we can estimate

$$\beta V_T(f) - \beta V_T(g) = \log \frac{\mathcal{T}(g \leftarrow f)}{\mathcal{T}(f \leftarrow g)} > \frac{1/N(f)}{N(f \leftarrow g)/N(g)}. \quad (25)$$

Here, $N(f)$ represents the total number of transitions from state f . Fig. 10 shows the comparison results for these pairs. It can be seen that these pairs basically satisfy the inequality Eq. (25), which further supports the reasonableness of the detailed balance condition. It is worth emphasizing that since these pairs are not fully included in the convex optimization of the minimum action, the estimates of $|\beta V_T(f) - \beta V_T(g)|$ may sometimes be overestimated.

D. DETAILED DISCUSSION ON THE MEANING OF ACTION

The generative dynamics of LLMs are often highly directional. In the main text, we pointed out that the strength of this directionality can be measured by the size of the minimum action. In this Supplemental Material, we show that the action actually provides a method for estimating how the state density in the LLM-generated state space varies with the potential, and we use Majority Voting as an example to show that although the measurement of the potential function must be performed through the agent, the distribution of the potential function is not sensitive to the specific design of the agent when detailed balance is satisfied.

Taking the **IdeaSearchFitter** agent and the Conditioned Word Generation Agent constructed with GPT5-Nano as examples, we measured the distribution of potential functions for all states in the database that were sampled at least twice, as shown in Fig. 11. The distributions of both agents exhibit significant localized structures, indicating that the

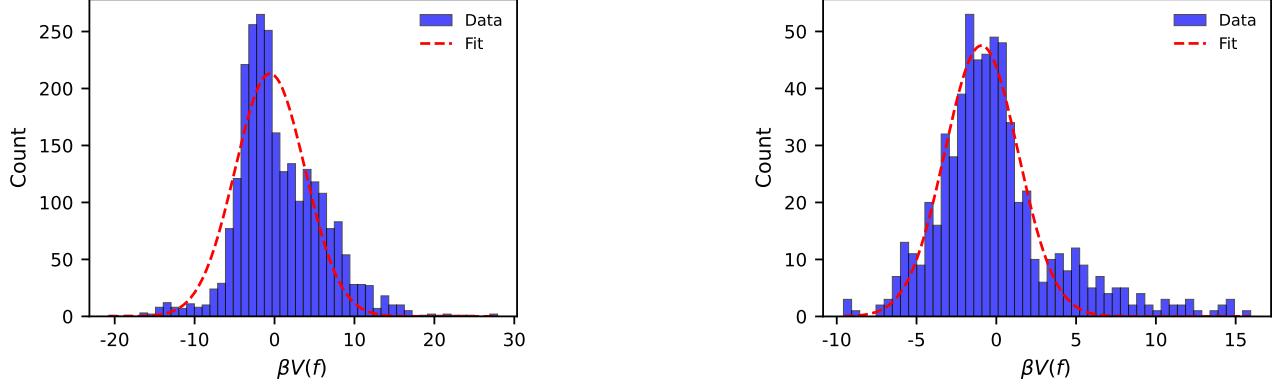


FIG. 11. (a) Distribution of potential functions for all states sampled at least twice in the `IdeaSearchFitter` agent. It exhibits a localized structure and can be fitted with a Gaussian distribution $\mathcal{N}(\mu = -0.56, \sigma = 4.37)$. (b) Distribution of potential functions for all states sampled at least twice in the Conditioned Word Generation Agent. It exhibits a localized structure and can be fitted with a Gaussian distribution $\mathcal{N}(\mu = -0.93, \sigma = 2.30)$.

state density of the agents may be localized; in other words, the long-term behavior of these agents may be insensitive to the sampling temperature in the state space. The distribution can be fitted with a Gaussian distribution $\mathcal{N}(\mu, \sigma)$. The relatively high standard deviation indicates that this potential function distribution is wide, within which the agent can exhibit significant directionality.

To quantify the relationship between action and state distribution, we assume that for a typical transition $g \leftarrow f$, it satisfies $\beta V(f) - \beta V(g) \gg 1$. Considering detailed balance, we can approximately write

$$\mathcal{S} \approx \int \int_{f,g \text{ for } V(f) < V(g)} 2 \exp(\beta(V(f) - V(g))) \mathcal{T}(g \leftarrow f) Df Dg. \quad (26)$$

If we only consider the scaling introduced by detailed balance, without considering the more specific structure of the state transition kernel, we can assume that (taking $\beta = 1$ to simplify the notation):

$$\begin{aligned} \mathcal{S} &\sim \int \int_{V_f < V_g} \frac{2}{2\pi\sigma^2} \exp \left[(V_f - V_g) - \frac{V_f^2}{2\sigma^2} - \frac{V_g^2}{2\sigma^2} \right] dV_f dV_g \\ &= \int_{-\infty}^{+\infty} dV_f \int_{V_f}^{+\infty} dV_g \frac{2}{2\pi\sigma^2} \exp \left[(V_f - V_g) - \frac{V_f^2}{2\sigma^2} - \frac{V_g^2}{2\sigma^2} \right]. \end{aligned} \quad (27)$$

By making the variable substitutions $u = V_f + V_g$ and $v = V_g - V_f$, we can obtain

$$\begin{aligned} \mathcal{S} &\sim \int_0^{+\infty} dv \int_{-\infty}^{+\infty} du \frac{2}{2\pi\sigma^2} \exp \left[-v - \frac{u^2 + v^2}{4\sigma^2} \right] \\ &= \int_0^{+\infty} dv \frac{2}{\sqrt{\pi}\sigma} \exp \left[-v - \frac{v^2}{4\sigma^2} \right] \\ &= 2e^{\sigma^2} \operatorname{erfc}(\sigma), \end{aligned} \quad (28)$$

where erfc is the complementary error function. For large σ , we can approximately write $\mathcal{S} \approx \frac{\mathcal{S}_{\sigma=0}}{\sigma\sqrt{\pi}}$, indicating that the size of the action is inversely proportional to the standard deviation of the state density. Note that the normalization should be chosen to match the measurement as $\mathcal{S}_{\sigma=0} = K(0)$. The comparison of the expected minimum action size through the potential function and the actual minimized action for the two agents is shown in Table IV.

This suggests that the size of the action estimates the characteristic energy scale in the agent's transitions, with smaller actions indicating that the agent's transition dynamics are more directional, while larger actions suggest that it is difficult for the agent's transitions to exhibit a clear directionality.

It is worth noting that measuring action is a more efficient method compared to directly measuring the directional distribution of the entire state space. In practical applications, the agent's transitions can be sampled through limited measurements to estimate the characteristic energy scale corresponding to this generative dynamics, thereby helping

TABLE IV. Comparison of expected minimum action size through the potential function and actual minimized action for two agents.

Agent	σ	Expected min. action	Actual min. action
IdeaSearchFitter	4.38	0.129	0.150
Conditioned Word Generation(GPT5-Nano)	2.30	0.245	0.195

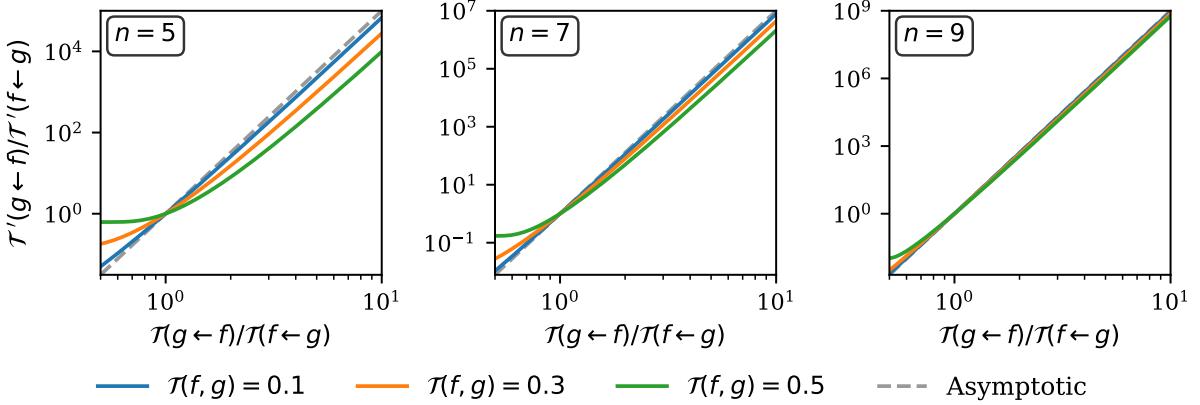


FIG. 12. Numerical comparison of both sides of Eq. (30). The comparison is performed for $M = 10$, with different colors representing different values of $\mathcal{T}(g \leftarrow f)$. The three subplots represent the cases of $n = 5, 7, 9$, respectively.

to improve the design of agent tasks more efficiently. For example, in Fig. 5, it can be seen that when poorer fits are sampled by IdeaSearchFitter, hyperparameters should be controlled to reduce the action and enhance the directionality of the agent's generation. While after reaching a better fit, the directionality of the internal generative dynamics of the LLM no longer aligns with that required by the optimization function, and hyperparameters should be controlled to increase the action, allowing the agent to serve as a mutation core [18, 20] to explore the state space more efficiently without directional constraints.

We next illustrate that when detailed balance is satisfied, the design of the agent often only changes the scale of the potential function rather than its distribution. Therefore, the size of the action may serve as a universal metric for agent design. First, we assume that before each agent transition, instead of directly generating and extracting a new state, M candidate states are generated, and then the state that appears more than $n > M/2$ times among the candidate states is selected as the new state (if no state meets this condition, the transition is rejected). Under this design, assuming the original agent transition kernel is $\mathcal{T}(g \leftarrow f)$, the new transition kernel can be written as

$$\begin{aligned} \mathcal{T}'(g \leftarrow f) &= \sum_{k=n}^M \binom{M}{k} [\mathcal{T}(g \leftarrow f)]^k [1 - \mathcal{T}(g \leftarrow f)]^{M-k} \\ &= I_{\mathcal{T}(g \leftarrow f)}(n, M-n+1), \end{aligned} \quad (29)$$

where $I_x(a, b)$ is the regularized incomplete beta function. Assuming \mathcal{T} is not very large, we have

$$\frac{\mathcal{T}'(g \leftarrow f)}{\mathcal{T}'(f \leftarrow g)} = \frac{I_{\mathcal{T}(g \leftarrow f)}(n, M-n+1)}{I_{\mathcal{T}(f \leftarrow g)}(n, M-n+1)} \approx \left(\frac{\mathcal{T}(g \leftarrow f)}{\mathcal{T}(f \leftarrow g)} \right)^n. \quad (30)$$

Numerical comparison is shown in Fig. 12. It can be seen that when $\mathcal{T}(g \leftarrow f)$ is small, Eq. (30) basically holds. Indicating that simply multiplying $V_{\mathcal{T}}$ by a constant n can estimate the new potential function $V_{\mathcal{T}'} \approx nV_{\mathcal{T}}$, suggesting that the specific design of the agent has little effect on the distribution of the potential function. For this issue, we also realize that $\mathcal{S} \sim \frac{K(0)}{\sqrt{\pi}\sigma} \sim \frac{1}{n}$, indicating that by increasing the selection threshold n , the action can be effectively reduced, thereby enhancing the directionality of the agent's generative dynamics, while increasing M does not significantly affect the size of the action but can improve sampling success rates by increasing the sampling budget when the transition kernel is small.

Generally speaking, an agent performing tasks within the distribution (e.g., in fields such as healthcare, experiments, etc. [16, 49, 50]) should be designed to have a lower action, while an agent performing tasks outside the distribution (e.g., in fields such as exploring the frontiers of mathematics and theoretical physics [18–20, 51, 52]) should be designed to have a higher action.

E. DISCOVERY OF POTENTIAL FUNCTION USING IDEASEARCH

In this Supplemental Material, we describe how to configure IdeaSearch and present the form and meaning of the best potential function discovered. IdeaSearch is an automatic program search method based on LLMs [20, 46], which can iteratively evaluate and optimize programs to solve complex problems by combining LLMs and evolutionary algorithms. We use IdeaSearch to search for the expression form of the potential function. Specifically, we represent the expression of the potential function as a combination of a predefined list of functions and operators, and the goal of IdeaSearch is to find an expression that maps this list to a floating-point number to minimize the corresponding action. IdeaSearch is configured with 16 different models to search for the target potential function. After running for 4000 rounds, the parameters in the best potential function were manually replaced and optimized using a random descent algorithm to minimize the action. The result is shown in Code 1, and the specific configuration parameters, evaluation, and running scripts of IdeaSearch can be found in the GitHub repository.

The best potential function assigns a potential value by tokenizing the input mathematical function string and then evaluating its structure and complexity, normalizing the value to the range [-1, 1]. The function considers not only the syntactic integrity of the input string but also extracts various features, including function usage, parameter structure, and specific substructures. These features are combined to form the final expression. It is important to emphasize that even though the structure of the potential function discovered by IdeaSearch does not capture the non-commutativity of the potential function with respect to strings, it can still be used to estimate the directionality of the agent’s transitions. This indicates that the directionality of agent transitions arising from LLM generation can be observed at different levels of coarse-graining, such as string level or expression level.

The following table lists the optimized parameter values obtained through random descent optimization with the minimum action being 0.47. All values are rounded to 2 decimal places.

Parameter	Value	Parameter	Value
empty_input_potential	-0.85	freq_var_weight	1.82
paren_penalty	1.70	freq_var_cap	10.04
extra_char_penalty	0.43	entropy_bonus	0.60
extra_char_threshold	2.13	log_v_bonus	1.35
length_penalty_divisor	4.00	log_bonus	0.60
max_depth_penalty	0.42	pattern_affinity_bonus	0.15
max_depth_threshold	0.33	pattern_count_divisor	11.67
func_penalty	0.36	linear_logv_weight	0.29
div_pow_penalty	0.42	centered_linear_weight	0.27
abs_penalty	6.50	nonlinear_weight	0.81
trig_penalty	0.75	exp_weight	0.35
nested_expr_penalty	0.54	proximity_cap	3.74
div_zero_risk_penalty	0.54	proximity_bonus	0.14
pow_risk_penalty	1.05	simple_bonus	1.00
sqr_risk_penalty	0.20	simple_length_threshold	77.42
no_params_penalty	1.00	simple_func_threshold	2.00
few_params_penalty	1.50	short_bonus	0.50
few_params_threshold	2.87	short_length_threshold	50.72
optimal_params_min	3.00	max_energy	4.59
optimal_params_max	5.53	K	1.37
optimal_params_bonus	0.43	pattern_affinity_threshold	0.29
excess_params_penalty	1.07	pattern_affinity_adjustment	0.01
excess_params_threshold	-0.48	min_potential	-1.72
		max_potential	0.93
		nan_inf_default	0.00
		overall_factor	2.04

TABLE V: Optimized parameter values for the potential function discovered using IdeaSearch.

The potential function discovered using IdeaSearch is implemented in Python as shown in Code 1.

Listing 1. Potential function discovered using IdeaSearch

```

1 import numpy as np
2 import re
3 import math
4

```

```

5 # Default values for parameters if not provided
6 default_params = {
7     'id_to_token': {
8         0: 'sin', 1: 'cos', 2: 'tan', 3: 'arcsin', 4: 'arccos', 5: 'arctan',
9         6: 'tanh', 7: 'log', 8: 'log10', 9: 'exp', 10: 'square', 11: 'sqrt',
10        12: 'abs', 13: '*', 14: '**', 15: '/', 16: '+', 17: '-', 18: '1',
11        19: '2', 20: 'pi', 21: 'log_v_k_nu', 22: 'param1', 23: 'param2',
12        24: 'param3', 25: 'param4', 26: 'param5', 27: 'param6', 28: 'param7',
13        29: 'param8', 30: 'param9', 31: '(', 32: ')', 33: ' '
14    }
15}
16
17 def potential_optimized_batch(token_ids_list: list, params: dict) -> np.ndarray:
18     """
19     Batch version of potential using numpy vectorization for speed.
20     Follows the exact logic of potential() but processes multiple expressions at once.
21
22     Args:
23         token_ids_list: A list of token_id lists, each representing a mathematical expression.
24         params: A dictionary containing the parameters for calculating the potential.
25
26     Returns:
27         A numpy array of potentials (energies) for all expressions.
28     """
29     # Use provided params, falling back to defaults
30     p = {**default_params, **params}
31
32     n = len(token_ids_list)
33     if n == 0:
34         return np.array([])
35
36     id_to_token = p['id_to_token']
37
38     # Pre-allocate arrays for vectorized operations
39     potentials = np.zeros(n, dtype=np.float64)
40
41     # Process each expression
42     for i, token_ids in enumerate(token_ids_list):
43         # Reconstruct expression string from tokens
44         s = ''.join(id_to_token.get(t, "") for t in token_ids)
45         s = (s or "").strip()
46         s_lower = s.lower()
47
48         # 1) Input validity check
49         if not s_lower:
50             potentials[i] = p['empty_input_potential']
51             continue
52
53         # 2) Syntax completeness check
54         depth = 0
55         max_depth = 0
56         bad_paren = False
57         for ch in s:
58             if ch == '(':
59                 depth += 1
60                 if depth > max_depth:
61                     max_depth = depth
62             elif ch == ')':
63                 depth -= 1
64                 if depth < 0:
65                     bad_paren = True
66                     depth = 0
67             if depth != 0:

```

```

68     bad_paren = True
69
70 # 3) Feature extraction
71 funcs = re.findall(r'\b(?:exp|log|ln|log10|sqrt|tanh|sin|cos|tan|abs|pow|ceil|
72     floor|log_v_k_nu)\b', s_lower)
73 num_funcs = len(funcs)
74
75 num_exp = s_lower.count('exp')
76 num_log = s_lower.count('log') + s_lower.count('ln') + s_lower.count('log10')
77 num_sqrt = s_lower.count('sqrt')
78 num_abs = s_lower.count('abs')
79 num_trig = s_lower.count('sin') + s_lower.count('cos') + s_lower.count('tan')
80 num_div = s_lower.count('/')
81 num_pow = s_lower.count('**') + s_lower.count('^^')
82
83 param_list = re.findall(r'\bparam\d+\b', s_lower)
84 unique_params = sorted(set(param_list))
85 num_params = len(unique_params)
86 param_counts = [param_list.count(p_name) for p_name in unique_params]
87 total_params = sum(param_counts)
88
89 if num_params > 0:
90     mean_params = total_params / num_params
91     freq_var = sum((c - mean_params) ** 2 for c in param_counts) / num_params
92     entropy = -sum((c / total_params) * math.log((c / total_params) + 1e-12) for
93         c in param_counts) if total_params > 0 else 0.0
94     entropy_norm = entropy / (math.log(num_params) + 1e-12) if num_params > 1
95         else 0.0
96 else:
97     freq_var, entropy_norm = 0.0, 0.0
98
99 # Nikuradse-2 related structure recognition
100 has_log_v = 'log_v_k_nu' in s_lower
101 linear_logv = bool(re.search(r'\bparam\d+\s*\*\s*log_v_k_nu\b', s_lower))
102 centered_linear = bool(re.search(r'\bparam\d+\s*\*\s*\(\s*log_v_k_nu\s*[-]\s*
103     param\d+\s*\)\b', s_lower))
104 logistic_present = len(re.findall(r'1\s*/\s*\(\s*1\s*/\s*\+\s*exp', s_lower)) > 0
105 tanh_present = len(re.findall(r'\btanh\s*\(\s*, s_lower)) > 0
106 softplus_present = len(re.findall(r'log\s*\(\s*1\s*/\s*\+\s*exp', s_lower)) > 0
107
108 pattern_count = int(has_log_v) + int(linear_logv) + int(centered_linear) + int(
109     logistic_present) + int(tanh_present) + int(softplus_present)
110 pattern_affinity = pattern_count / p['pattern_count_divisor']
111
112 nested_expr = bool(re.search(r'exp\s*\(\s*, s_lower)) or bool(re.search(r'log\s*\(
113     , s_lower))
114
115 div_zero_risk = '/ in s_lower
116 pow_risk = num_pow > 0
117 sqrt_risk = num_sqrt > 0 and not bool(re.search(r'sqrt\s*\(\s*abs', s_lower))
118
119 # 4) Energy calculation and mapping to [-1, 1]
120 energy = 0.0
121
122 # Syntax completeness penalty
123 if bad_paren:
124     energy += p['paren_penalty']
125     extra_chars = len(re.findall(r'^0-9a-zA-Z_\+\-\*\|\^\.\(\)\,\s*', s_lower))
126     energy += max(0, extra_chars - p['extra_char_threshold']) * p['extra_char_penalty']
127
128     energy += math.log1p(len(s)) / p['length_penalty_divisor']
129     energy += max(0, max_depth - p['max_depth_threshold']) * p['max_depth_penalty']
130
131 # Basic function and operator complexity penalty

```

```

125     energy += num_funcs * p['func_penalty']
126     energy += (num_div + num_pow) * p['div_pow_penalty']
127     energy += num_abs * p['abs_penalty']
128     energy += num_trig * p['trig_penalty']
129
130     # Risk penalty
131     energy += p['nested_expr_penalty'] if nested_expr else 0.0
132     energy += p['div_zero_risk_penalty'] if div_zero_risk else 0.0
133     energy += p['pow_risk_penalty'] if pow_risk else 0.0
134     energy += p['sqrt_risk_penalty'] if sqrt_risk else 0.0
135
136     # Parameter diversity adjustment
137     if num_params == 0:
138         energy += p['no_params_penalty']
139     elif num_params < p['few_params_threshold']:
140         energy += p['few_params_penalty'] * (p['few_params_threshold'] - num_params)
141     elif p['optimal_params_min'] <= num_params <= p['optimal_params_max']:
142         energy -= p['optimal_params_bonus']
143     else:
144         energy += (num_params - p['excess_params_threshold'])
145
146     energy += p['freq_var_weight'] * min(freq_var, p['freq_var_cap'])
147     energy -= p['entropy_bonus'] * entropy_norm
148
149     # Nikuradse-2 prior structure reward
150     if has_log_v:
151         energy -= p['log_v_bonus']
152     elif num_log > 0:
153         energy -= p['log_bonus']
154
155     # Structure matching reward
156     energy -= p['pattern_affinity_bonus'] * pattern_affinity
157
158     # Structure similarity weighted penalty
159     proximity_score = 0.0
160     if has_log_v and num_params > 0:
161         proximity_score = (
162             p['linear_logv_weight'] * int(linear_logv)
163             + p['centered_linear_weight'] * int(centered_linear)
164             + p['nonlinear_weight'] * (int(logistic_present) + int(tanh_present) +
165                 int(softplus_present))
166             + p['exp_weight'] * num_exp
167         )
168         proximity_score = min(p['proximity_cap'], proximity_score)
169     energy -= p['proximity_bonus'] * proximity_score
170
171     # Simplicity preference
172     simple_pattern = re.compile(r'^[0-9a-zA-Z_\\s\\+\\-\\*\\/\\.\\((\\))]+$')
173     truly_simple = bool(simple_pattern.match(s_lower)) and num_funcs <= p['
174         simple_func_threshold'] and num_pow == 0
175     if truly_simple and len(s) < p['simple_length_threshold']:
176         energy -= p['simple_bonus']
177     elif len(s) < p['short_length_threshold']:
178         energy -= p['short_bonus']
179
180     # Avoid unstable operations
181     if '0' in s_lower and ('/' in s_lower or '**' in s_lower):
182         energy += 0 # Final mapping
183
184     if energy < 0:
185         energy = 0.0
186     max_energy = p['max_energy']
187     if energy > max_energy:
188         energy = max_energy

```

```
187     K = p['K']
188     norm = 1 - math.exp(-energy / K)
189     val = -1 + 2 * norm
190
191     # Fine-tuning for key pattern matching
192     if pattern_affinity >= p['pattern_affinity_threshold'] and (logistic_present or
193         tanh_present or softplus_present or has_log_v):
194         val -= p['pattern_affinity_adjustment']
195
196     if math.isnan(val) or math.isinf(val):
197         val = p['nan_inf_default']
198     val = max(p['min_potential'], min(p['max_potential'], val))
199
200     potentials[i] = float(round(val, 5))*p['overall_factor']
201
202 return potentials
```