LLMs as Mold Makers and Not 3D Printers

Harsha Kokel
IBM Research
San Jose, CA 95120

Abstract

This abstract proposes a paradigm shift in Large Language Model (LLM) utilization, advocating for a "**moldmaker**" approach over the current "**3D printer**" mode. Currently, LLMs frequently operate like 3D printers, generating each piece from scratch and necessitating that users engineer every response. This leads to considerable time and token costs and inconsistent output formats, tones, and logic. Further, even minor variations in expected outputs result in high compute costs and redundant efforts. An alternative approach is to use LLMs as MoldMakers, building once and using many times. In this paradigm, the LLM is leveraged to create reusable templates, or generate code that serve as "molds". Molds that can be used to solve multiple problems. This "build once, use many times" approach mirrors the efficiency of historical general-purpose solvers, such as SAT Solvers or Domain-independent Planners, which were "built once" to "solve many problems" in a modular, composable, and computationally efficient manner. Adopting the MoldMaker paradigm for LLMs offers substantial benefits, including faster inference, improved consistency and reliability, and lower costs. Ultimately, this work argues for transitioning from workflows that use LLMs to generate small pieces for every problem to those that use LLMs to generate once piece but use that piece multiple times across problems.

Large Language Models (LLMs) are profoundly reshaping the computing landscape, establishing themselves as a pivotal force driving the next computing paradigm. They are quickly becoming the natural language interface to interact with code [7, 4], APIs [17, 15], web applications [14] as well as storage systems [16, 11, 21, 13]. Despite this progress, challenges around accuracy, reliability, and high cost continue to hinder their widespread adoption in enterprise and high-stakes environments. Consequently, two key desiderata for effective integration of LLMs into workflows are: (1) verifiability of generated outputs to ensure correctness and reliability, and (2) reusability of outputs to amortize development and deployment costs. This work contends that adopting the MoldMaker paradigm—rather than treating LLMs as one-off 3D printers—is crucial to achieving both verifiability and reusability in LLM-driven workflows.

A **mold maker** is a specialist (or machine/process) that designs and fabricates molds—rigid tools used to shape materials into specific forms. Typically molds are designed to efficiently produce many identical parts. Molds are extremely useful to scale up the production in reliable and consistent fashion. This approach emphasizes the principle of "Build Once, Use Many Times". A **3D printer** is a machine that builds objects layer-by-layer from digital models. They are typically used during prototyping and design iteration. They are slow and typically used to produce bespoke and unique pieces for individual customers. Essentially, Mold making is for efficient, large-scale, repeatable production whereas 3D printing is for flexibility, customization, and early-stage design.

Traditionally, AI research has prioritized the development of general-purpose solvers—such as SAT solvers [5], theorem provers [19], and domain-independent planners [9]—"built once" but "applied broadly" across a range of problem instances. These systems exemplify modularity, composability, and computational efficiency. A similar philosophy underpins various assessment tools which distill patterns from data into lightweight, scalable models capable of fast and reliable inference in high-

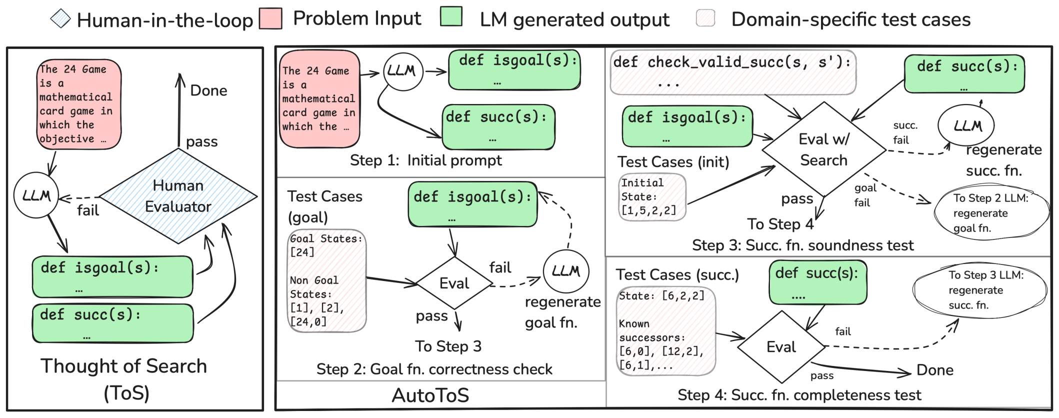

Figure 1: An overview of ToS and AutoToS approach that uses LLMs to generate successor function and goal function from description of a domain; which can be subsequently invoked multiple times across problem by search algorithm. An Example of MoldMaker Approach.

stakes settings [10, 6, 1]. These approaches can be likened to Mold Makers as they promote efficiency, consistency, and scalability through reusability.

However, current trends use LLMs to generate each piece of content. This incurs time, token costs, and redundant efforts. Further, even after all the engineering, the result is an inconsistent format, tone, and logic. This can be likened to using LLMs as 3D printers, where each generation incurs significant cost and generated output is bespoke and unique everytime. This abstract appeals to use LLMs as mold-makers insteal. With LLMs as mold-makers, we create molds—reusable templates, frameworks, code, logic flows. Since these molds are well-defined modules, they can be verified for consistency and reliability. As molds are only generated once, they provide significant cost advantage. Most importantly, once the molds are reliable, we can use them to scale and be more efficient.

There are several compelling examples that illustrate the effectiveness of applying the MoldMaker paradigm in LLM-driven workflows across diverse domains [18, 3, 8]. Notably, Thought of Search (ToS) demonstrates how LLMs can be leveraged to generate reusable search components–such as goal conditions and successor functions–from natural language task descriptions [12]. Rather than relying on prompting LLMs for each problem instance [20], ToS promotes a structured methodology that can be verified and applied across muliple problems in that domain, significantly improving consistency and efficiency. Building upon this idea, AutoToS [2] introduces a further refinement by automating the verification and validation process through the use of unit tests and iterative prompting (see Fig. 1). This enables the system to generate reusable and sound search components with minimal human involvement, thereby reducing both computational cost and human oversight. Together, these systems exemplify how LLMs, when used as mold makers rather than one-off generators, can deliver scalable and reliable performance in complex reasoning and decision-making workflows.

In summary, the MoldMaker paradigm presents a compelling framework for addressing the limitations of current LLM workflows, particularly with respect to cost, consistency, and scalability. By shifting from bespoke, per-instance generation toward reusable, verifiable artifacts, this approach aligns with longstanding principles in AI system design—emphasizing modularity, composability, and efficiency. Through case studies such as ToS and AutoToS, we observe that LLMs can be systematically harnessed to construct problem-solving templates that generalize across tasks, reducing redundancy and improving reliability. As LLMs continue to play a central role in software, API, and knowledge system interfaces, adopting the MoldMaker mindset may be key to building sustainable, trustworthy, and scalable AI infrastructure.

References

[1] Azra Bihorac, Tezcan Ozrazgat-Baslanti, Ashkan Ebadi, Amir Motaei, Mohcine Madkour, Panagote M Pardalos, Gloria Lipori, William R Hogan, Philip A Efron, Frederick Moore, et al. Mysurgeryrisk: development and validation of a machine-learning risk algorithm for major

complications and death after surgery. *Annals of surgery*, 269(4):652–662, 2019.

[2] Daniel Cao, Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi. Automating thought of search: A journey towards soundness and completeness (student abstract). In *AAAI*, pages 29328–29330. AAAI Press, 2025.

[3] Augusto B Corrêa, André G Pereira, and Jendrik Seipp. Classical planning with llm-generated heuristics: Challenging the state of the art with python code. *arXiv preprint arXiv:2503.18809*, 2025.

[4] Cursor. Cursor: The ai-powered code editor. `https://www.cursor.sh/`, 2023.

[5] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[6] Hassan M. Elbiss and Fikri M. Abu-Zidan. Artificial intelligence in gynecologic and obstetric emergencies. *International Journal of Emergency Medicine*, 18(1):20, 2025. doi: 10.1186/s12245-025-00820-8. URL `https://doi.org/10.1186/s12245-025-00820-8`.

[7] GitHub. Github copilot. `https://github.com/features/copilot`, 2021.

[8] Yilun Hao, Yang Zhang, and Chuchu Fan. Planning anything with rigor: General-purpose zero-shot planning with llm-based formalized programming. In *ICLR*. OpenReview.net, 2025.

[9] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[10] Julia Hippisley-Cox, Carol Coupland, and Peter Brindle. Development and validation of qrisk3 risk prediction algorithms to estimate future risk of cardiovascular disease: prospective cohort study. *BMJ*, 357, 2017.

[11] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *CoRR*, abs/2406.08426, 2024.

[12] Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi. Thought of search: Planning with language models through the lens of efficiency. In *NeurIPS*, 2024.

[13] Georgia Koutrika. Natural language data interfaces: A data access odyssey (invited talk). In *ICDT*, volume 290 of *LIPIcs*, pages 1:1–1:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[14] Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiaoyong Wei, Shanru Lin, Hui Liu, Philip S. Yu, and Qing Li. A survey of webagents: Towards next-generation AI agents for web automation with large foundation models. *CoRR*, abs/2503.23350, 2025.

[15] OpenAI. Function calling and api calling with gpt models. 2023. Accessed: 2025-08-04.

[16] Abdul Quamar, Vasilis Efthymiou, Chuan Lei, and Fatma Özcan. Natural language interfaces to data. *Found. Trends Databases*, 11(4):319–414, 2022.

[17] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

[18] Ion Stoica, Matei Zaharia, Joseph Gonzalez, Ken Goldberg, Koushik Sen, Hao Zhang, Anastasios Angelopoulos, Shishir G. Patil, Lingjiao Chen, Wei-Lin Chiang, and Jared Quincy Davis. Specifications: The missing link to making the development of LLM systems an engineering discipline. *CoRR*, abs/2412.05299, 2024.

[19] Geoff Sutcliffe and Christian B. Suttner. Evaluating general purpose automated theorem proving systems. *Artif. Intell.*, 131(1-2):39–54, 2001.

[20] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

[21] Weixu Zhang, Yifei Wang, Yuanfeng Song, Victor Junqiu Wei, Yuxing Tian, Yiyan Qi, Jonathan H. Chan, Raymond Chi-Wing Wong, and Haiqin Yang. Natural language interfaces for tabular data querying and visualization: A survey. *IEEE Trans. Knowl. Data Eng.*, 36 (11):6699–6718, 2024.