Towards a Probabilistic Framework for Analyzing and Improving LLM-Enabled Software

Juan Manuel Baldonado
ICC UBA/CONICET and DC, FCEN
Universidad de Buenos Aires
Buenos Aires, Argentina
juanmanuelbaldonado@gmail.com

Flavia Bonomo-Braberman
ICC UBA/CONICET and DC, FCEN
Universidad de Buenos Aires
Buenos Aires, Argentina
fbonomo@dc.uba.ar

Víctor A. Braberman
ICC UBA/CONICET and DC, FCEN
Universidad de Buenos Aires
Buenos Aires, Argentina
vbraber@dc.uba.ar

***Abstract*—Ensuring the reliability and verifiability of large language model (LLM)-enabled systems remains a significant challenge in software engineering. We propose a probabilistic framework for systematically analyzing and improving these systems by modeling and refining distributions over clusters of semantically equivalent outputs. This framework facilitates the evaluation and iterative improvement of Transference Models–key software components that utilize LLMs to transform inputs into outputs for downstream tasks. To illustrate its utility, we apply the framework to the autoformalization problem, where natural language documentation is transformed into formal program specifications. Our case illustrates how distribution-aware analysis enables the identification of weaknesses and guides focused alignment improvements, resulting in more reliable and interpretable outputs. This principled approach offers a foundation for addressing critical challenges in the development of robust LLM-enabled systems.**

***Index Terms*—LLMs, prompt engineering, autoformalization**

I. Introduction

In recent years, Large Language Models (LLMs), such as GPT-4 [13] and Gemini [17], have demonstrated remarkable capabilities across diverse applications. These successes are largely attributed to their instruction-following abilities and in-context learning. By conditioning on task-specific instructions (zero-shot) or a small set of examples (few-shot), LLMs have been shown to perform a wide array of tasks effectively [3]. This adaptability has led to a proliferation of applications leveraging LLMs to elicit various downstream tasks via prompting. The integration of LLMs into software systems is a rapidly growing trend, but it presents numerous challenges [5]. Particularly, phenomena such as hallucination [7] impact what can be guaranteed about such applications. One basic aspect to disciplined engineering of complex systems is how to understand and analyze their behavior. While NLP research acknowledges and leverages at some extent the underlying predictive model on next token distribution (e.g., [4], [18]), research papers on software engineering of LLM-enabled applications typically pinpoint that aspect just as a troublesome source of non-determinism (e.g., flaky tests [5]). Instead, we believe that a disciplined engineering of reliable LLM-enabled applications should be based on both the identification of constituent "transference models" (TMs) –LLM-powered software components transforming inputs into outputs–, and the modeling and analysis of the underlying distribution over classes that comprise (transference)-equivalent outputs (i.e., "meaning classes" [4]). In particular, we propose an evaluation and refinement framework based on the above insights and also on the belief that it is feasible and worthwhile to characterize in which extent and how distribution over meaning-classes satisfies or deviates from expected distribution characteristics. More concretely, we hypothesize that, for inputs in which TM yields concentrated distributions on incorrect meaning-classes (adversarial cases), it is possible to verbalize into task-specific terms the kind of misalignment, and that constitutes a valuable input for re-engineering transference models (e.g., sub-task decomposition, replacement of underlying LLM, or other targeted adjustments informed by such analyses).

To illustrate these ideas, we present a preliminary case study addressing the problem of autoformalization: translating doc-strings that specify the intention of a code snippet into formal pre- and post-conditions (e.g., assertions written in Dafny [9]). This downstream task, which translates natural language into formal languages, is particularly useful in software testing and verification [15]. Our study demonstrates how probabilistic analysis and meaning-class transformations can lead to focused alignment improvements, offering a foundation for principled advancements in LLM-enabled systems.

II. Preliminaries

A. Large Language Models

These predictive models are parameterized functions, whose parameter sets are often initially found by optimizing an objective or loss function for next-token prediction with respect to a training corpus. Thus, at its core, an LLM yields a probability distribution over the set of tokens or vocabulary (i.e., $P(t_k|x)$, the probability of the k-th token being t_k given the context x). LLMs are typically used in its generative role by performing some particular decoding approach using the next-token predictive model [12]. In general, LLM-enabled software leverages them by injecting prompts

Partially supported by UBACyT Grants 20020220300079BA and 20020190100126BA, CONICET PIP 11220200100084CO, and ANPCyT PICT-2021-I-A-00755, A-1-2022-1-173516 IDRC-ANII, SWPERFI UFAM-MOTOROLA RD&I Project "Técnicas de Inteligência Artificial para Análise e Otimização de Desempenho de Software", and Amazon Research Award – Fall 2023 on Automated Reasoning.

that condition continuations/responses (i.e., LLM predicts next-tokens conditioned by the prompt), thus hopefully eliciting the desired downstream tasks. Task-agnostic prompting strategies are frequently used (e.g. [10], [19]).

B. Autoformalization

As indicated, our illustration is based on the problem of autoformalization in software verification. This is the transformation from natural language descriptions (docstrings, typically available in development processes) into some formally correct and automatically verifiable format [15], [16] which enables automated reasoning on program's correctness. In our illustration, we choose Dafny [9] as the target formal language and particularly the pre- and post-condition sections.

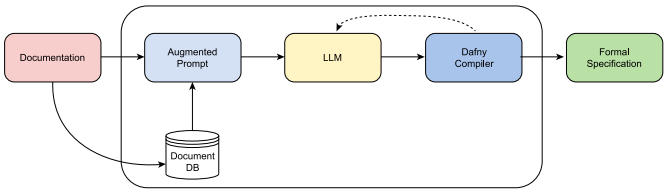

Figure 1. Architecture of the illustrated baseline transference model

C. Transference Models

Following [15], we define "Transference Models" (TMs) as software components that use (typically, prompt-modulated) LLMs to elicit downstream tasks as the key (but not necessarily unique) means for transforming some input data into some output data. Those transformations have potentially stochastic nature given the underlying predictive model and (sampling-based) decoding strategy of the LLMs. Thus, TMs behavior can be formulated as a function of input-output pairs into reals, that is, $T : \mathcal{I} \times \mathcal{O} \to \mathbb{R}$, where $T(i, o)$ denotes the probability that $i \in \mathcal{I}$ is transferred to $o \in \mathcal{O}$, and for $i \in \mathcal{I}$, $T(i, \cdot)$ is a probability distribution over $\mathcal{O}$. As an example, in autoformalization proposals, typically, the TM is a stochastic process that links natural language with annotations as the expected type of input-output pairs (e.g., [2], [15]). Figure 1 shows the architecture of the TM developed for illustration. Similarly to Clover's "doc2anno" TM [15], it elicits a formalization task and, if Dafny compiler detects that the generated annotation is syntactically incorrect, it instructs a corrective version of the formalization task by including compiler's feedback a bounded number of times. We also include RAG-like (few-shot) prompt generation [10] to get annotations for likely similar problems by querying a vector database.

D. Distributions over Semantic Domains

Given that Language Models are density estimation functions that assign probability to every possible string, we concur with others (e.g. [4], [6]) that probability should be deemed assigned to concepts, not strings: there are often many (or even infinite) strings that represent a given idea equally well (e.g., in the autoformalization problem, logically equivalent annotations) and share the same meaning when embedded into the abstract domain $\mathcal{O}$. This is, in theory, the result of summing up the probabilities assigned to strings in each of the equivalence classes.

E. Computing Empirical Distributions of TMs

Given the potential richness of TMs and decoding strategies plus the fact next-token probabilities are not always accessible, for a given input, we approximate the TM probability distribution by the empirical categorical distribution on meaning-classes that are identified and built on-the-fly by clustering generated outputs (e.g., by checking annotations to be SMT-equivalent). More concretely, re-execution of code of the transference model with the same input triggers the (many times stochastic) decoding adapters [12] of the involved LLMs and thus the entire TM behaves as a black box stochastic process (whose behavior depends on hyperparameters and settings of those adapters like temperature, etc.). Yielded outputs are then clustered according to the equivalence relation and empirical distribution on classes is computed (see similar discussion in [4]).

III. The Framework

A. Assumptions, Hypothesis and Rationale

Firstly, we assume that for most LLM-enabled applications, it is possible to understand their LLMs interactions as part of the implementation of one or more TMs. TMs might be implemented straightforwardly by zero-shot instruction prompting that includes some representation of the input. However, TMs could also be much more sophisticated or even generated by automated prompt generation frameworks like [8]. As illustrated above, they could include statically or dynamically orchestrated external tools (e.g., some feedback signal for corrective tasks), general reactive LLM-triggered external computation [21], the chaining of a series of lower-level transference models [8], etc. TMs are the component under analysis in our approach. Thus the second assumption is that those input/output transformations can be recovered and described as –human understandable– downstream tasks either by developers/testers (or LLMs, in the future) (e.g. [2]). This understanding should include the ability to assess and equate tasks results into conceptual classes. Our main hypothesis is that to gain insight on the potential behavior of an LLM-based TM (e.g., when testing it, when evaluating it, when validating it, when engineering it, when over-sighting it, etc.) it is key to embrace the predictive nature of underlying model and thus approximate and characterize the yielded underlying probability distribution of TMs over concepts[1]. More concretely, we believe that concentrated and misaligned cases are the ones to be detected and reduced during development time. The rationale is manyfold: on the one hand, concentrated and misaligned cases pinpoint to (adversarial) inputs that lead to systematic

[1]We also hypothesize that consistent probability assignments for task's equivalent results is likely an emergent behavior of LLMs.

misalignment in the prompt-modulated predictive model while they could potentially deceive certainty-based hallucination detection techniques [4] (i.e., false negatives that would need to be further identified or mitigated somehow in a safe way). Also, it has been shown that assuming TMs to be aligned and concentrated with high probability may provide error-bounded performance guarantees (e.g. [15]). Thus, improvement is defined to be akin refinement or behavioral subtyping used in program verification settings (e.g. [11]): replacing a TM with an improved one will mean less false positives for a certainty-based hallucination guardrail, if deployed, and a reduced likelihood of error. We also require (almost-sure) non-regression on aligned cases.

On the other hand, we also hypothesize that, when a meaning class that gets the most of the probability mass does not match TM's expected result, the nature of failure can be, in general, stated in terms of mistakes humans have already studied and categorized in the corresponding problem domain and could lead to hints regarding improvement. That is, we seek characterizing in *task-specific* terms the nature of *failures*. For instance, for the autoformalization problem studied, "too weak/strong pre/post-conditions" are well-known concepts of formalization mistakes ontology. As we will show later, they are also typically the perfect fit for characterizing TM's failures.

B. Defining Improvement

For a given input, TM's distribution over meaning classes is *aligned* when the class with the largest assigned probability is a correct one, and *misaligned* otherwise. *Concentration* is used here as a practical proxy of certainty on generated concepts (akin semantic entropy of the empirical distribution), and it is defined here as a distribution characterized by the winning meaning class having probability mass greater than or equal to the sum of the rest of the masses. Other definitions are possible (e.g., [15]). Given two alternative t, t' TMs, we say that TM t' pointwise *improves* over t for a test set when t' extends the set of inputs which feature aligned and concentrated distributions and reduces the set of inputs that feature concentrated and misaligned distributions. The generalized and computable definition of almost-sure improvement for the entire domain is possible by assuming some probability distribution D over the domain of inputs (or a set of possible distributions). Although its definition and study is out of scope, it can be informally stated as a) probability of an input sampled according D to be related by t' to a concentrated and aligned distribution is greater than sampling from D an input with such characteristics in t, b) with probability close to one, an input drawn from D that has a concentrated and aligned distribution in t also has an aligned and concentrated distribution in t', and c) probability of an input sampled according D to be related by t' to a concentrated and misaligned distribution is less or equal than sampling from D an input with such characteristics in t.

IV. Illustrative results on Autoformalization

Next we show an anecdotal illustration in line with the stated hypothesis (no claim of generalization). We use an existing dataset of Dafny programs; the CloverBench dataset consists of 62 small hand-written example programs similar to those found in standard computer science textbooks [15]. Each program in the dataset consists of a single method and includes a Dafny implementation, annotations specifying pre-conditions and post-conditions, and a docstring that describes program's intention. We extract from those programs docstrings and method signatures and we use pre- and post-conditions as the groundtruth to assess alignment of the most probable meaning class. We get the empirical approximation by re-execution of the transference model (30 times in this case) and clustering with the assistance of a SMT solver. For the sake of simplicity, we treat each syntactically-invalid result as its own class, rather than assigning probability mass to the edit-distance closest valid class. Compared to collapsing all invalid results into a single class, it may lead to less concentration in verdicts, especially when the model assigns significant probability to diverse invalid outputs. We opted for the *Gemini 1.5 Flash* model with a context of 1000 tokens. Hyperparameters were the default ones for that LLM: top_k with $k = 40$, top_p with $p = 0.95$ and temperature 0.7. This means we work with a generative-process distribution that is more skewed (and tail-truncated) than the underlying prompt-modulated next-token prediction model distribution. Yet, this illustrates a plausible generative-process distribution of the LLM running at the core of the TM whose stochastic behavior one wants to approximate under such decoding settings. More details and data can be found in [1].

TABLE I
Alignment and Concentration of Distributions

Alignment/Concentration	Concentrated	Not Concentrated
Aligned	46	3
Correct class gen. but misaligned distr.	2	3
Correct class not generated	4	4

In Table I, we show a breakdown of the results obtained in terms of alignment and concentration. For a given input (e.g., docstring + signature), we say that a meaning class (i.e., set of equivalent pre/post specifications) is *correct* when it is made up of specs equivalent to groundtruth specification. Firstly, 54 out of the 62 inputs, the empirical distribution features a non-zero probability meaning-class which is the correct one (although not necessarily the winning one). For those 54 inputs, in 49 inputs the obtained empirical distribution is actually aligned. Moreover, in 46 cases, the distribution is *both concentrated and aligned*: a sunny day scenario. It is worth noting that the baseline TM using zero-temp (almost) deterministic single-output regime in the underlying LLM gets 47 aligned results. However, TM's distributions on concepts for some of those inputs are either misaligned or they are high-entropy distributions. Thus, zero-temp disguises potential issues of the underlying model's probability assignment. On the other hand, there are also inputs in which the zero-temp utterance is incorrect but TM's distribution is actually aligned and concentrated. A similar phenomenon has been observed for close-ended selection tasks [6]. We argue that for reliability and

safety concerns one should focus on understanding and reducing cases in which misalignment concurs with high concentration. In fact, Table II yields the in-depth analysis of winning meaning-class in the misaligned cases. As mentioned earlier, how the winning-class differs from the groundtruth can be many times defined naturally in task-specific terms. In fact, 11 out of 14 misalignments were easily matchable to well-known bug types in formalization. When analyzing reasons for misalignment, weak post-condition is the more frequent characteristic of winning meaning-classes. One of the 2 cases in "debugging" focus (misaligned and concentrated) is *LinearSearch*. It is supposed to return the index of the first appearance of an element, but the winning meaning-class does not include into the post-condition that the returned index is indeed the first appearance. Moreover, that formula was actually generated in another context and that led to the idea that the model was not able to detect that part of the intention in the docstring and formalize it in the same inference step. In general, this phenomenon is known as the *compositional gap* [14] in which models can correctly answer all sub-problems but not generate the overall solution. This, in turn, suggested that a structured docstring (i.e., NL pre-conditions followed by the NL post-conditions, both marked with an identification) could constitute a better subdomain for getting better performance in terms of the proposed distribution analysis. This could trigger, for instance, a re-engineering of the TM by adding a sentence close-ended classification task (pre-condition, post-condition, none) to help structuring stripped docstring as a preprocessing component added to the baseline TM.

TABLE II
NATURE OF MISALIGNMENT FOR THE OBTAINED DISTRIBUTIONS.

Formalization Mistakes	Concentrated	Non-Concentrated	Total
Weak post-condition	2	3	6
Incorrect post-condition	1	2	4
Syntax error	1	2	3
Weak pre-condition	1	0	1
Strong pre-condition	1	0	0
Total	6	7	14

In fact, in Table III we show how the modified TM safely improves over the baseline. Vast majority of inputs lead to concentrated probability distributions and the concentration is on the right meaning class. Moreover, the number of misaligned and concentrated cases dropped from 6 to 2. Those two inputs leading to concentrated misalignment (indeed, not even a correct output generated), *modify_2d_array* and *on-line-max*, according to our proposal would be the focus of analysis for a new round of engineering efforts (out-of-scope). In fact, relevance of those inputs in focus can also be backed by the fact that in the baseline TM the resulting distributions were misaligned and non-concentrated (one without generation of the correct result). Moreover, *modify_2d_array* was a paradigmatic case of non-concentration in the baseline TM (20 different meaning classes). On the other hand, both TMs were unable to generate one of the constraints required for the concept of maximum in *on-line-max*. Through this guided analysis, we have identified natural language utterances where the model's assigned probabilities for formalizations are unsatisfactory.

TABLE III
ALIGNMENT AND CONCENTRATION FOR THE MODIFIED TM.

Alignment/Concentration	Concentrated	Not Concentrated
Aligned	57	1
Correct class gen. but misaligned distr.	0	0
Correct class not generated	2	2

V. RELATED WORK

Surface form competition for multiple-choice tasks is introduced in [6] and later discussed in [20]. The hallucination detection method of [4] works approximating entropy on meaning-classes, and equivalence checking for grouping results is done by means of LLMs given the diversity of benchmarked tasks. In [15], transference model and similar notions of alignment and concentration are introduced in their analytical model to mathematically describe the hypothesis required to bound error in their program synthesis approach. In some sense, those works provide evidence on the potential usefulness of such concepts but do not elaborate on a framework to analyze and improve transference models.

VI. CONCLUSIONS AND FUTURE WORK

We illustrate some principled engineering insights gained when probability distributions over meaning-classes are recovered from the stochastic behavior of transference models. This paper pinpoints at concentrated misalignment as key failure for "debugging" but inputs leading to non-concentrated distributions and their characterization might be also relevant. Sensitivity of distributions to input perturbations that are (or not) supposedly equivalent to the transformation under analysis could also be key to understand task-related limitations of the TM. Several RQs and experiments are necessary to claim generalizability and impact of the approach (e.g., In which extent tasks characteristics impact effectiveness of analysis?, In which extent knowing input distribution is key to analysis?, that is, When does improvement on a test set translate into generalized improvement?, Which is the impact of decoding strategy in the analysis?, How robust the framework is to alternative practical distribution/equivalence approximation methods?, etc.). There are also many possible conceptual and automation paths, including: approximation techniques for computing concentration, the definition of a richer language to predicate on inputs and resulting distributions, and notions of compositionality of TMs. Also, we are aware that, for some tasks/transference models, a more structured domain of concepts might be needed (e.g., preference relation) and improvement definitions might need to accommodate potential trade-offs among lowering entropy and aligning distributions.

Last but not least, it is foreseeable the help of AI-based solutions to characterize misalignments and troublesome/adversarial inputs and even the assistance in decomposition, prompt-engineering and hyperparameter tuning.

References

[1] Juan Baldonado. Source code for the experiments. https://github.com/jmbuba/llm-semantic-perf, 2024.

[2] Víctor A. Braberman, Flavia Bonomo-Braberman, Yiannis Charalambous, Juan G. Colonna, Lucas C. Cordeiro, and Rosiane de Freitas. Tasks people prompt: A taxonomy of LLM downstream tasks in software verification and falsification approaches, 2024. arXiv:2404.09384.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language models are few-shot learners. *NeurIPS 2020*.

[4] Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nat.*, 630(8017):625–630, 2024. doi:10.1038/S41586-024-07421-0.

[5] Ahmed E. Hassan, Dayi Lin, Gopi Krishnan Rajbahadur, Keheliya Gallaba, Filipe Roseiro Côgo, Boyuan Chen, et al. Rethinking software engineering in the era of foundation models: A curated catalogue of challenges in the development of trustworthy FMware. *FSE Comp. 2024*. doi:10.1145/3663529.3663849.

[6] Ari Holtzman, Peter West, Vered Shwartz, Yejin Choi, and Luke Zettlemoyer. Surface form competition: Why the highest probability answer isn't always right. *EMNLP 2021*. doi:10.18653/V1/2021.EMNLP-MAIN.564.

[7] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, et al. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38, 2023. doi:10.1145/3571730.

[8] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, et al. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. *ICLR 2024*.

[9] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. *LPAR 2010*. doi:10.1007/978-3-642-17511-4_20.

[10] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *NeurIPS 2020*.

[11] Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994. doi:10.1145/197320.197383.

[12] Clara Meister, Gian Wiher, and Ryan Cotterell. On decoding strategies for neural text generators. *Trans. Assoc. Comput. Linguistics*, 10:997–1012, 2022. doi:10.1162/TACL_A_00502.

[13] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, et al. GPT-4 technical report, 2024. arXiv:2303.08774.

[14] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *EMNLP 2023*. doi:10.18653/V1/2023.FINDINGS-EMNLP.378.

[15] Chuyue Sun, Ying Sheng, Oded Padon, and Clark W. Barrett. Clover: Closed-loop verifiable code generation. *SAIV 2024*. doi:10.1007/978-3-031-65112-0_7.

[16] Christian Szegedy. A promising path towards autoformalization and general artificial intelligence. *CICM 2020*. doi:10.1007/978-3-030-53518-6_1.

[17] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, et al. Gemini: A family of highly capable multimodal models, 2024. arXiv:2312.11805.

[18] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, et al. Self-consistency improves chain of thought reasoning in language models. *ICLR 2023*.

[19] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS 2022*.

[20] Sarah Wiegreffe, Matthew Finlayson, Oyvind Tafjord, Peter Clark, and Ashish Sabharwal. Increasing probability mass on answer choices does not always improve accuracy. *EMNLP 2023*. doi:10.18653/V1/2023.EMNLP-MAIN.522.

[21] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, et al. ReAct: Synergizing reasoning and acting in language models. *ICLR 2023*.