Semantics-Based Analysis and Repair of Neural Language Models

Jian Gu <jian.gu@monash.edu>

Abstract

Foundation models, and neural language models (LMs) in particular, are increasingly deployed as long-lived components of software systems, coding assistants, retrieval pipelines, and agentic workflows. In these settings, they must not only perform well at deployment, but remain reliable and maintainable as requirements, interfaces, and operating constraints evolve. Failures such as outdated knowledge, brittle reasoning, unsafe tendencies, and recurring task-specific errors therefore become maintenance problems rather than isolated benchmark mistakes. This thesis studies how such models can be maintained at the model level, rather than only through prompts, wrappers, filters, or repeated broad retraining. Its central claim is that semantics provides the most practical bridge between external requirements and internal computation: engineers specify failures semantically, models realize them through latent states and parameterized transformations, and effective maintenance must therefore connect semantic specification to internal mechanism. On this basis, the thesis develops a unified program of semantics-based LM analysis, repair, and meta-repair. LM analysis is to make internal computation legible in semantic terms; LM repair is to correct concrete failures through targeted, low-cost interventions; and LM meta-repair is to improve the semantic organization and measurement of models so that later intervention becomes easier, safer, and more systematic. Viewed in this way, language models are treated not merely as trained predictors, but as maintainable engineering artifacts whose internal behavior can be examined, modified, and organized over time.

The thesis establishes methods and empirical evidence for each part of this maintenance perspective. First, it develops semantics-based analysis tools that treat hidden states as semantically organized objects rather than opaque vectors. Using semantic transition and internal attribution, it shows how semantically meaningful computation can reveal where adaptation effort should be concentrated, enabling computation-efficient fine-tuning through semantic-aware layer freezing. It then extends this analytical perspective across models by introducing semantics-first latent alignment for parametric knowledge transfer, showing that aligned layer outputs can support robust cross-scale transfer more effectively than parameter-space matching alone. Building on these foundations, the thesis proposes two repair frameworks for trained LMs: a neuron-level hot-fix method that localizes and patches a very small number of influential neurons from semantic differences between incorrect and desired outputs, and a semantics-guided optimization method that repairs multiple interacting failures while controlling effectiveness, efficiency, sparsity, and side effects such as interference, overspecialization, and forgetting. Finally, the thesis advances a meta-repair perspective by introducing semantic reference frames, semantic coordinates, semantic dynamics, and semantic traces as a principled basis for comparing residual-stream computation across depth. Together, these contributions provide a coherent methodology for analyzing, repairing, and characterizing language models, and support the broader view that foundation models should be engineered as reusable, reliable, and maintainable software artifacts.

1 Motivation

Recent advances in foundation models have substantially reshaped the role of neural systems in modern software. Large pretrained models can now be adapted to a wide range of downstream tasks, domains, and interaction settings with comparatively modest additional supervision, prompting, or tuning [1–3]. For language models (LMs), this shift is especially consequential. They are no longer used only as benchmarked text predictors, but increasingly as coding assistants, review tools, retrieval-augmented components, and engines for agentic workflows embedded in real development and decision-making pipelines [4, 5]. In these settings, the model is not a disposable experimental artifact. It is a persistent component of a broader software ecosystem whose behavior must remain useful, reliable, and maintainable over time. This thesis therefore treats language models not merely as objects of evaluation, but as *living engineering artifacts* whose capabilities, limitations, and maintenance costs matter in practice.

Once an LM is integrated into real workflows, its strengths and weaknesses become operational concerns. Beneficial capabilities can be reused across tasks and interfaces, but so can outdated knowledge, brittle reasoning patterns, unsafe tendencies, and hard-to-maintain behaviors [1]. A model that hallucinates an obsolete API today may reproduce the same defect tomorrow in code synthesis, later in automated review, and again in an agentic repair loop. Likewise, a model that exhibits unstable reasoning or undesirable behavioral drift can propagate those problems across downstream services. The engineering problem is therefore not limited to training and deployment. It also includes understanding the model, maintaining it after deployment, and intervening when recurring failure modes begin to affect real use.

Engineering Foundation Models as Maintainable Artifacts Software engineering provides a natural starting point for this problem. Over several decades, it has developed systematic methods for specifying software, analyzing its structure, testing its behavior, localizing faults, and maintaining deployed systems [6]. Research on foundation models has likewise produced essential surrounding practices, including data and training pipelines, evaluation harnesses, deployment infrastructure, and MLOps lifecycle support [7]. These practices are indispensable, but they primarily organize work *around* the model. By contrast, this thesis focuses on the next step: treating the model itself as a maintainable artifact that can be analyzed, repaired, and made easier to maintain over time.

The need for explicit model-oriented maintenance becomes clearer when foundation models are compared with conventional software. A software component is grounded in explicit artifacts such as source code, interfaces, requirements, tests, and maintenance history. These do not eliminate complexity, but they provide a relatively stable basis for diagnosis, validation, and repair. A foundation model is different. Its functional content is learned rather than directly written: what the model knows, how it tends to reason, and which behavioral tendencies it exhibits are distributed across parameters, activations, and latent states [1, 8]. Benchmarks, task suites, and safety datasets provide important evidence about behavior, but they do not fully specify what the model has learned, how a particular output is produced, or how a local intervention will affect

other capabilities. This limitation is the central *specification gap* between engineering conventional software and engineering learned models.

If language models are to be treated as important engineering artifacts, this specification gap must be addressed at the model level rather than merely worked around. Prompts, wrappers, filters, retrieval modules, and broad retraining remain important parts of the engineering toolbox, but they do not eliminate the need for direct maintenance of the model itself. When such maintenance is feasible, it offers at least three practical advantages. First, it improves *reusability*: a repair applied once to the model can benefit many downstream applications, interfaces, and workflows simultaneously, rather than being reimplemented separately in each application layer. Second, it improves *reliability*: recurring reasoning failures, unsafe tendencies, and persistent error patterns can be treated as maintainable system defects rather than tolerated as isolated bad outputs. Third, it improves *maintainability*: organizations must repeatedly accommodate changing APIs, local policies, domain conventions, proprietary knowledge, and evolving risk constraints, and targeted model-oriented interventions can often be updated and validated more systematically than repeated rounds of broad retraining. As foundation models become embedded in increasingly consequential workflows, their maintenance can no longer remain an informal afterthought.

LM Analysis, LM Repair, and LM Meta-repair This thesis approaches model-oriented maintenance through three linked activities: *LM analysis*, *LM repair*, and *LM meta-repair*. LM analysis seeks mechanistic understanding of model behavior that can guide intervention. LM repair seeks targeted correction of concrete failures or recurring failure patterns. LM meta-repair seeks to improve the semantic and structural conditions under which later analysis and repair are carried out, so that maintenance becomes easier, more local, and more reliable. As a whole, these activities address three connected engineering questions: what is happening inside the model, how should the model be changed, and how can future changes be made easier?

As shown in Figure 1, the thesis adapts the logic of conventional program maintenance to the learned and distributed setting of language models. In conventional software maintenance, engineers inspect a program to diagnose behavioral or structural defects, intervene by constructing targeted repairs, and improve the system through refactoring or evolution so that future changes are less costly. Semantics-based LM maintenance follows the same engineering logic, but the maintained artifact and the substrate of intervention differ: the object is a learned model rather than a written program; analysis targets internal semantic structure, representations, and circuits rather than source-level control or data-flow structure; and repair modifies responsible mechanisms while constraining interference with unrelated capabilities. The final step, LM meta-repair, mirrors program evolution at a different level: it aims to stabilize and reorganize the semantic conditions under which future analysis and repair are performed.

An intuitive framing comes from the tripartite model of cognition, which distinguishes the *autonomous* mind, responsible for fast and default responses; the *algorithmic* mind, which provides the computational resources for deliberate reasoning and for maintaining decoupled or hypothetical representations; and the *reflective* mind, which sets goals, evaluates whether default responses

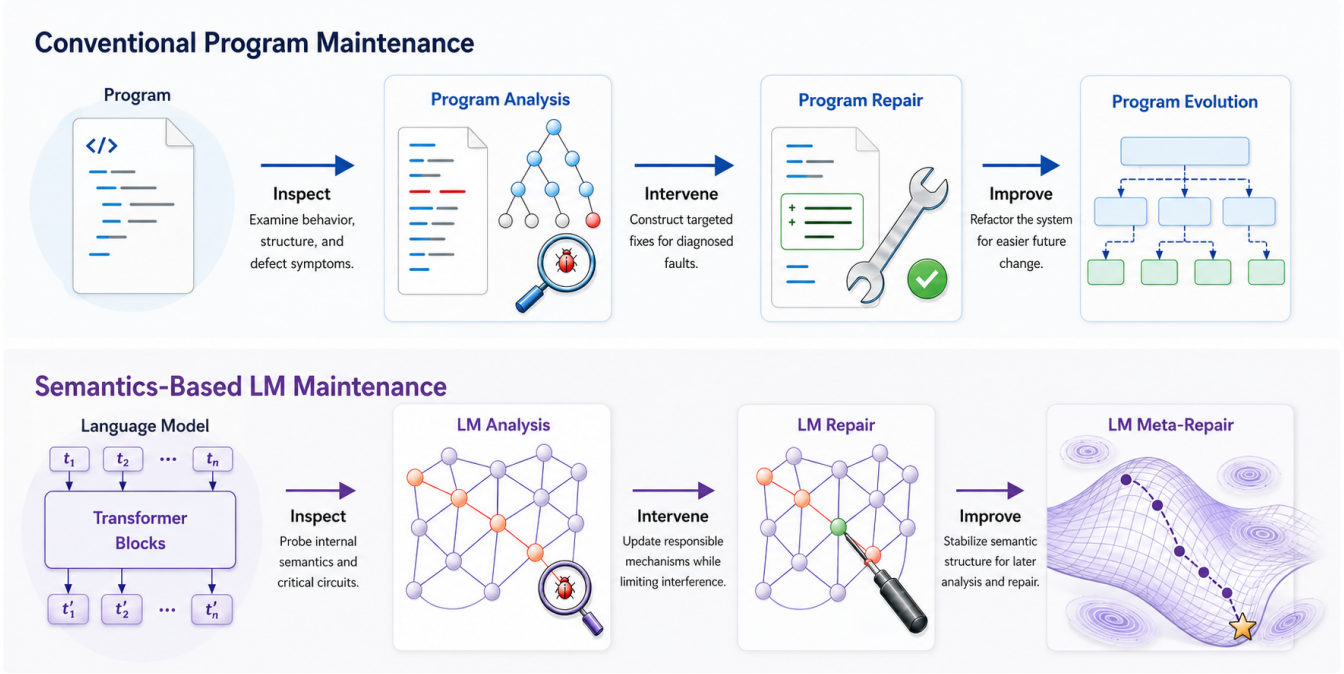

Figure 1: Analogy between program maintenance and language-model maintenance.

should be questioned, and determines when analytic intervention should be initiated [9, 10]. The analogy is heuristic rather than literal. It does not imply that language models instantiate human cognition in any straightforward sense. Rather, it offers a useful conceptual scaffold for thinking about maintenance. At an abstract level, much LM behavior can be treated as a set of default computational tendencies distributed across parameters and representations. Analysis aims to make those tendencies legible. Repair aims to alter them when they become undesirable. Meta-repair aims to improve the conditions under which later diagnosis and intervention are performed. On this view, the tripartite framing organizes the thesis around three connected concerns: default behavior, targeted intervention, and improved capacity for future intervention.

Semantics as the Operational Medium To make this maintenance chain operational, the thesis treats *semantics* as the medium through which external requirements can be related to internal model computation. Engineers typically formulate failures in semantic terms: the model should know a fact, distinguish concepts, sustain a valid line of reasoning, avoid risky behavior, or produce code that satisfies particular constraints. The model, however, realizes these requirements through token interfaces, activations, residual states, attribution paths, gradients, and parameters. Because foundation models do not expose program-like specifications, semantics provides a practical medium for connecting external engineering requirements to internal model computation. In this sense, semantics is not merely a way of describing model behavior after the fact; it is the operational medium through which analysis, intervention, and validation can be aligned.

Accordingly, LM analysis in this thesis is not generic post-hoc explanation, but semantically grounded and mechanistic inquiry that links observed failures to internal computation. Recent work in mechanistic interpretability has made such links increasingly concrete through neuron-

level attribution, path tracing, circuit analysis, and residual-stream investigation [8, 11, 12]. This semantic grounding also makes repair actionable, because an intervention must do more than locate a signal in the network; it must identify that signal's semantic role so that the resulting change remains targeted, interpretable, and compatible with surrounding capabilities. On this basis, LM repair can be defined more precisely. It is neither broad retraining, generic fine-tuning, nor the narrow task of factual knowledge editing [13, 14], but a targeted maintenance process for an already trained model that aims to correct a specific failure while preserving as much unrelated capability as possible. This is difficult because LM behavior is distributed across redundant representations, layered residual computation, and large matrix transformations. Yet the same structure also creates opportunities, as recent work on secure automatic programming suggests that internal vulnerabilities and failure-related features can sometimes be identified and modified through targeted intervention [15]. For this reason, analysis and repair must be developed together. LM meta-repair extends this logic by asking what repeated maintenance reveals about the model itself: semantic entanglement, weak comparability across layers, interference among updates, and poor separation between functions all make later intervention more difficult. This direction is closely related to intrinsic interpretability, which aims to build transparency into model architectures, representations, or computations instead of relying only on post-hoc explanatory surrogates [16]. LM meta-repair is the maintenance-oriented counterpart: it feeds repair requirements back into the design of foundation models, training objectives, analytical frameworks, and internal organization, with the aim of making subsequent analysis and repair more modular, decomposable, stable, and efficient.

A Motivating Example Modern software development is becoming more conversational and increasingly shaped by tools built on foundation models, especially code language models. These systems are used not only for code completion, but also for synthesis, refactoring, review, debugging, test generation, patch construction, and the coordination of complex development workflows [17, 18]. Recent research increasingly emphasizes *agentic coding*, where AI systems take a more autonomous role in planning, execution, testing, and iterative refinement [18–20]. This trend is useful here because it clarifies a broader move from AI-assisted programming to software workflows that are increasingly mediated by language models. Code generation therefore offers a particularly visible instance of a broader maintenance problem. Its outputs are often directly executable, integrate easily into IDE and pipeline workflows, and tend to fail in recurring semantic ways, including API misuse, flawed control flow, inadequate error handling, insecure implementation patterns, and brittle assumptions about program state or environment. When a model repeatedly produces buggy, outdated, or risky code, repairing each generated program after the fact is not enough. Conventional program repair operates on individual artifacts: it may fix a specific output, but it does not change the model behavior that reproduces similar failures across tasks. When the source of the problem lies in the language model itself, for example in stale API knowledge, brittle reasoning about program logic, or a persistent tendency to emit insecure or fabricated implementations, LM repair becomes a direct and practically necessary way to improve code quality at its source.

This example also clarifies why the thesis studies analysis, repair, and meta-repair together, and

why it characterizes failures in terms of *knowledge*, *cognition*, and *behavior*. Analysis asks what aspects of the model should be examined: not only outputs and benchmark scores, but also semantic features, neuron activations, attribution paths, residual trajectories, layerwise roles, and cross-model semantic correspondences when these help explain computation. Repair asks what should be changed: outdated or distorted knowledge, brittle reasoning processes, and undesirable behavioral tendencies. In code generation, these correspond to incorrect or obsolete facts, such as deprecated APIs or incompatible dependency constraints; flawed intermediate reasoning about program semantics, specifications, or execution; and persistent behaviors such as overconfident fabrication, insecure default choices, or the recurrent production of vulnerable code. These categories are not intended as a rigid ontology, but as a practical engineering vocabulary for recurring failures. The same categories also arise beyond coding, for example when models must update stale knowledge, resist retrieval pollution, suppress dangerous behavioral drift, or remain reliable in increasingly agentic deployments [21, 22]. Meta-repair then asks what should be improved to make subsequent intervention easier: semantic coordinates, diagnostics, organizational priors, training-time constraints, and other aspects of model structure that determine whether maintenance is well posed in the first place.

Thesis Overview The thesis operationalizes the above motivation through a progressive research program: it first analyzes how language models encode and transform semantics, then uses this understanding to repair model failures, and finally studies how such repair processes can themselves be made more systematic and reusable.

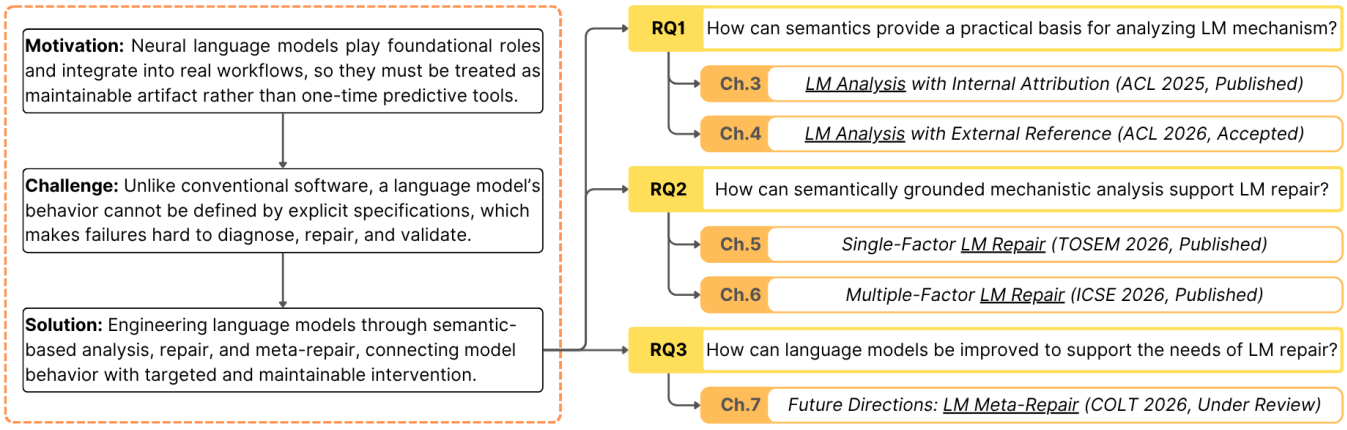

Figure 2: Overview of the thesis structure.

As summarized in Figure 2, this thesis develops a semantics-based maintenance program for language models. The program consists of three connected parts. Chapters 3–4 study *LM analysis*: how semantic information is represented, transformed, and aligned within and across language models. Chapters 5–6 study *LM repair*: how insights from semantic analysis can be used to identify and correct recurring model failures through targeted interventions. Chapter 7 studies *LM meta-repair*: how reusable semantic structures can make future diagnosis and repair more stable and systematic. Together, these parts establish language models as maintainable artifacts and consolidate a series of publications that advance this research direction.

References

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[2] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[3] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, vol. 10, 2022.

[4] Q. Zhang, C. Fang, Y. Xie, Y. Zhang, Y. Yang, W. Sun, S. Yu, and Z. Chen, "A survey on large language models for software engineering," *arXiv preprint arXiv:2312.15223*, 2023.

[5] J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang, and Y. Lou, "Large language model-based agents for software engineering: A survey," *ACM Transactions on Software Engineering and Methodology*, 2024.

[6] A. Abran, P. Bourque, R. Dupuis, and J. W. Moore, *Guide to the software engineering body of knowledge-SWEBOK*. IEEE Press, 2001.

[7] N. Hewage and D. Meedeniya, "Machine learning operations: A survey on mlops tool support," *arXiv preprint arXiv:2202.10169*, 2022.

[8] D. Rai, Y. Zhou, S. Feng, A. Saparov, and Z. Yao, "A practical review of mechanistic interpretability for transformer-based language models," *arXiv preprint arXiv:2407.02646*, 2024.

[9] K. E. Stanovich, "Distinguishing the reflective, algorithmic, and autonomous minds: Is it time for a tri-process theory," *In two minds: Dual processes and beyond*, pp. 55–88, 2009.

[10] K. Stanovich, *Rationality and the reflective mind*. Oxford University Press, 2011.

[11] Z. Yu and S. Ananiadou, "Neuron-level knowledge attribution in large language models," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3267–3280, 2024.

[12] J. Ferrando and E. Voita, "Information flow routes: Automatically interpreting language models at scale," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 17432–17445, 2024.

[13] Y. Yao, P. Wang, B. Tian, S. Cheng, Z. Li, S. Deng, H. Chen, and N. Zhang, "Editing large language models: Problems, methods, and opportunities," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10222–10240, 2023.

[14] S. Wang, Y. Zhu, H. Liu, Z. Zheng, C. Chen, and J. Li, "Knowledge editing for large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–37, 2024.

[15] A. El Husseini, Y. Izza, B. Genest, and A. Roychoudhury, "Repairing llm executions for secure automatic programming," in *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*, 2026.

[16] Y. Gao, Q. Meng, Y. Zhou, and L. Pan, "Towards intrinsic interpretability of large language models: A survey of design principles and architectures," *arXiv preprint arXiv:2604.16042*, 2026.

[17] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *ACM Transactions on Software Engineering and Methodology*, vol. 35, no. 2, pp. 1–72, 2026.

[18] A. E. Hassan, H. Li, D. Lin, B. Adams, T.-H. Chen, Y. Kashiwa, and D. Qiu, "Agentic software engineering: Foundational pillars and a research roadmap," *arXiv preprint arXiv:2509.06216*, 2025.

[19] R. Sapkota, K. I. Roumeliotis, and M. Karkee, "Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic ai," *arXiv preprint arXiv:2505.19443*, 2025.

[20] Y. Ge, L. Mei, Z. Duan, T. Li, Y. Zheng, Y. Wang, L. Wang, J. Yao, T. Liu, Y. Cai, *et al.*, "A survey of vibe coding with large language models," *arXiv preprint arXiv:2510.12399*, 2025.

[21] E. Turner, A. Soligo, M. Taylor, S. Rajamanoharan, and N. Nanda, "Model organisms for emergent misalignment," *arXiv preprint arXiv:2506.11613*, 2025.

[22] A. Lynch, B. Wright, C. Larson, S. J. Ritchie, S. Mindermann, E. Hubinger, E. Perez, and K. Troy, "Agentic misalignment: How llms could be insider threats," *arXiv preprint arXiv:2510.05179*, 2025.