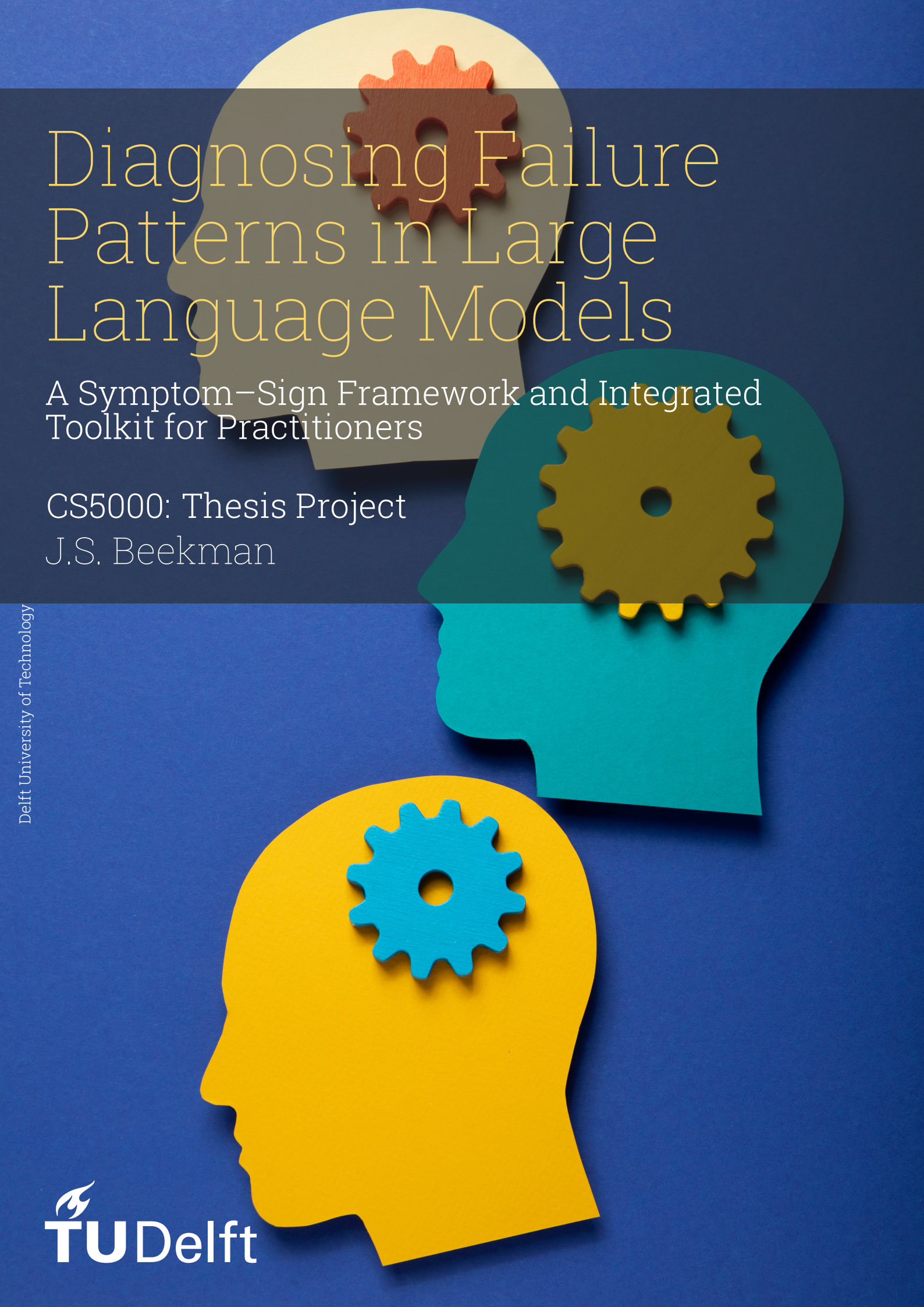

Diagnosing Failure Patterns in Large Language Models

A Symptom–Sign Framework and Integrated Toolkit for Practitioners

CS5000: Thesis Project

J.S. Beekman

Delft University of Technology

TU Delft

Diagnosing Failure Patterns in Large Language Models

A Symptom–Sign Framework and Integrated Toolkit for Practitioners

by

J.S. Beekman

to obtain the degree of Master of Science in Computer Science
at the Delft University of Technology,
to be defended publicly on Monday, April 20, 2026, at 15:00.

Student Number:	5885116
Project Duration:	February 2025 - April 2026
Thesis Committee:	Dr. J. Yang; TU Delft, Associate Professor
	Dr. K.W. Song; TU Delft, Associate Professor
Daily Co-Supervisor:	L. Corti; TU Delft, PhD Candidate
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover:	Top view people head shapes and cogwheels design by Freepik
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Abstract

Large language models (LLMs) are increasingly deployed in consequential settings, yet their failures remain challenging to understand. Unlike traditional software bugs, such undesirable behavior emerges from distributed, context-dependent interactions that resist straightforward debugging. While XAI methods can surface signals about individual predictions, they do not directly support the hypothesis-driven investigative process that characterizes diagnosis in practice: forming expectations, gathering evidence, and identifying recurring failure patterns.

This thesis addresses this gap by introducing (1) a diagnostic framework that structures diagnosis through symptoms (observed undesirable outputs), signs (evidence from interpretability methods), and failure patterns (recurring, explainable combinations of symptoms and signs), complemented by a Should-Know/Really-Know lens that distinguishes task expectation from actual model knowledge; and (2) a prototype diagnostic toolkit that operationalizes this framework through integrated evaluation, run comparison, and state externalization.

An evaluation study with eight practitioners using codebook thematic analysis reveals three core findings. First, practitioners universally adopt a baseline-first strategy, building diagnostic confidence through initial evaluation before deeper probing. Second, they triangulate across samples, metrics, and interpretability outputs rather than relying on single signals, using comparison as a central sensemaking operation for hypothesis testing. Third, diagnostic depth is systematically gated by three factors: interpretation friction (insufficient guidance on what methods reveal and how to act on their outputs), missing workflow glue (the absence of affordances for iterative refinement), and execution constraints (opaque platform limits that disrupt sustained diagnostic progress).

These findings reframe diagnostic tooling as infrastructure for iterative, hypothesis-driven reasoning, extending beyond the provision of isolated analytical methods. Effective diagnostic support must scaffold the full investigative cycle: from expectation formation and baseline calibration through evidence triangulation, hypothesis testing via comparison, and state externalization. This scaffolding must also account for the gating factors that shape the depth of diagnostic progress. This positions diagnosis as a knowledge and workflow challenge, with implications for tooling design, framework development, and empirical research into practitioner diagnostic workflows.

Contents

List of Figures

List of Tables

Glossary

Term	Definition
(Self-)Attention	A mechanism for assigning importance to each token relative to others in the input
Diagnosis	An iterative, hypothesis-driven process of investigating a model to identify, explain, and possibly localize failure patterns
Explainability	Providing human-understandable justifications for why a model produced specific outputs
Failure Pattern	A recurring combination of Symptoms and Signs that consistently indicate undesired model behavior
Interpretability	Understanding a model's internal mechanisms and representations (how decisions are computed)
Language Model (LM)	A model that predicts the next token in a sequence given the preceding tokens (also called Large Language Model (LLM) for very large versions)
Sign	An indirectly observable indicator of a potential model weakness or failure; likely requires an instrument or expert knowledge to surface
Symptom	A directly observable indicator of a potential model weakness or failure; does not (necessarily) require an instrument or expert knowledge to surface
Token	A sequence of characters treated as a single unit
Transformer	A deep learning architecture that uses a (multi-head) attention mechanism

Note on Interpretability and Explainability: While the foundational XAI literature often uses these terms interchangeably [62, 30], this thesis distinguishes them to clarify different diagnostic activities. Interpretability refers to understanding *how* a model works internally (mechanisms, representations, attention patterns), while explainability refers to understanding *why* a model produced specific behaviors or outputs (justifications, attributions). Both fall under the umbrella of Explainable Artificial Intelligence (XAI) and are relevant to model diagnosis.

1

Introduction

Large language models (LLMs) have been rapidly adopted across various domains, including software engineering, education, healthcare, and legal services [64, 119, 63, 7]. This adoption is driven by demonstrated task-level utility and the maturation of deployment infrastructure, as indicated by surveys showing sustained growth in enterprise LLM integration [1, 92, 96]. As these models are increasingly embedded in workflows and decision-making processes, their errors have meaningful consequences. Understanding and managing such errors, therefore, becomes an essential part of working with LLMs. In practice, this diagnostic responsibility falls primarily to practitioners (developers, researchers, and engineers who deploy and maintain these models), who must determine why a model behaves undesirably and how to address the issue. This difficulty motivates a central concern of this thesis: understanding and investigating LLM failures.

To address this challenge, one might turn to Explainable Artificial Intelligence (XAI), which provides tools for understanding model behavior, encompassing both interpretability methods (which reveal internal mechanisms and representations) and explainability methods (which provide human-understandable justifications for specific outputs) [99, 122]. Such methods can offer valuable insights into how a model arrives at a particular output and can play a crucial role in diagnosis. Yet, on their own, they do not directly address a different and equally important need: identifying why a model behaves incorrectly in specific situations. Understanding models requires diagnosis, a process that builds on explanatory tools but moves beyond them to reason about behavior or underlying causes.

In this thesis, diagnosis is understood as an iterative, hypothesis-driven process of investigating a model to identify, explain, and possibly localize failure patterns. This involves forming expectations about model behavior, probing model responses through interpretability methods and metrics, and connecting observations to potential underlying causes. It requires a structured workflow rather than reliance on a single explanatory artifact. A more diagnostic approach requires, for example, examining what a model should know to perform a task, what it actually knows in practice, and how discrepancies between the two may arise.

Existing research has begun to explore aspects of model diagnosis, proposing various frameworks and methods for investigating model failures [10, 120, 81]. These approaches demonstrate the value of examining a model through a diagnostic lens, with studies showing successful failure identification, model comparison, and the detection of critical failures in deployed systems. However, they remain fragmented and are often tailored to specific fields or types of failure. Practitioners working across different contexts must therefore piece together disconnected methods, requiring significant expertise to determine which approaches apply to their situation and how to integrate findings into coherent diagnostic insights. Moreover, these methods are rarely integrated into practitioner-facing tools and do not yet provide a structured, generalized diagnostic workflow.

Given these limitations, practitioners working with LLMs rely heavily on manual inspection, domain knowledge, and ad hoc experimentation when diagnosing failures, with studies showing that failure handling workflows are typically performed manually and without formal ML-specific methods [9, 82,

27]. Despite the proliferation of interpretability and explainability methods, these tools often remain underutilized in real diagnostic workflows due to challenges in tooling, evaluation, and integration into existing practices [27]. A formative study conducted at a software company[1] corroborated this pattern: participants reported that formal XAI tools were infrequently used during diagnosis, and that explanations were often disconnected from the practical questions they faced. Instead, they emphasized the need for contextual information and structured guidance to reason about *why* failures occurred. Combined with the fragmentation of diagnostic research identified above, these findings highlight a gap between existing approaches and practitioners' diagnostic needs.

This thesis addresses this gap by introducing a diagnostic framework and accompanying toolkit for transformer-based language models. The framework structures diagnosis as a progression from symptoms (observed undesirable outputs) through signs (evidence from interpretability methods) towards failure patterns (recurring, interpretable weaknesses). To support this process, the framework integrates a Should-Know/Really-Know lens, adapted from prior diagnostic work, to distinguish task expectations from actual model knowledge [93, 10]. The toolkit operationalizes this framework, providing a structured way for practitioners to explore and understand undesirable behavior. Together, these artifacts aim to support more systematic and accessible model diagnosis.

The work follows a Design Science Research (DSR) approach, guided by the guidelines of Hevner et al. [42]. The framework and toolkit are treated as research artifacts, grounded in prior work and practitioner needs, and evaluated through a user study. While the design process involved iterative refinement, the primary knowledge contribution emerges from the artifacts themselves and their empirical evaluation, consistent with DSR's focus on building and evaluating solutions to identified problems.

While the challenges outlined above suggest a need for better diagnostic support, three key gaps remain. First, the challenges and limitations shaping how practitioners diagnose LLM failures have not been examined within the context of diagnostic workflows. Second, it remains unclear what forms of diagnostic support would effectively address practitioners' needs in practice. Third, the extent to which integrated tooling can complement or improve upon current ad hoc approaches has not been empirically evaluated. This thesis addresses these gaps through the following research question:

Main Research Question

What kinds of tooling support practitioners in diagnosing undesirable behavior in large language models?

To address this question, the following sub-questions (SQ) have been defined:

- **SQ1:** What challenges and limitations shape how (large) language models are diagnosed?
- **SQ2:** What kinds of diagnostic support help practitioners identify and understand failure patterns?
- **SQ3:** How does the toolkit complement or improve upon current diagnostic practices used by practitioners?

Contributions

The primary contributions of this thesis are:

- A **diagnostic framework for LLM failures** that structures diagnosis through symptoms, signs, and failure patterns;
- A **prototype diagnostic toolkit** that operationalizes the framework and provides integrated support for diagnostic workflows;
- **Empirical insights into practitioner diagnostic behavior** revealing baseline-first strategies, three gating factors for diagnostic depth, and design implications emphasizing scaffolding the full investigative cycle rather than providing isolated methods.

[1] https://resolver.tudelft.nl/700ce316-5047-4182-ae25-e47db8ca4054

Structure

The remainder of this thesis is structured as follows: Chapter 2 provides an overview of the background and related work in LLM diagnosis, reviews the existing tooling landscape, and identifies systematic gaps in support for diagnostic workflows. Chapter 3 introduces the diagnostic framework that structures diagnostic reasoning and organizes evidence around symptoms, signs, and failure patterns. Chapter 4 presents the system architecture and implementation of the diagnostic toolkit. Chapter 5 describes the evaluation study assessing the effectiveness of the toolkit in real-world diagnostic scenarios. Chapter 6 presents the empirical findings from the evaluation study. Chapter 7 interprets the findings at a higher level of abstraction, situates them within existing research, and derives implications for diagnostic tooling. Finally, Chapter 8 synthesizes the contributions and reflects on future directions for diagnostic practice and research.

2

Background and Related Work

This chapter provides the conceptual foundation for understanding undesirable behavior in large language models (LLMs), the need for diagnostic support, and the challenges practitioners face in investigating model failures. It reviews key properties of transformer-based language models, existing failure taxonomies, prior work on model explanation and diagnosis, and the landscape of tools available for diagnostic work. The chapter concludes by identifying systematic gaps in existing tooling that inform the diagnostic framework and toolkit introduced in Chapters 3 and 4.

2.1. Transformer-Based Language Models

Transformer-based language models have become the dominant architecture in modern natural language processing due to their scalability, strong empirical performance, and ability to generalize across tasks [104, 20]. Model families such as GPT, DeepSeek, and Claude underpin many contemporary LLM systems [70, 25, 6]. While transformers demonstrate strong scalability, empirical performance, and generalization, the architectural characteristics underlying these capabilities introduce challenges that complicate the diagnosis of undesirable model behavior.

A central characteristic of transformer models is their reliance on deep, multi-layer self-attention mechanisms [104]. Self-attention enables a model to dynamically weight the relationships between tokens in an input sequence, thereby capturing long-range dependencies and contextual interactions. Formally, self-attention computes a weighted combination of *values* for each token, where the weights are determined by the compatibility between that token's *query* and the *keys* of all other tokens in the sequence. Intuitively, the query reflects what a token is searching for in context, the key encodes what each token offers, and the value carries the information that is passed forward. Given input representations packed into matrices $\mathbf{Q}$ (queries), $\mathbf{K}$ (keys), and $\mathbf{V}$ (values), the attention output is computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

where d_k is the dimensionality of the key vectors, introduced as a scaling factor to prevent excessively large dot products that would push the softmax into regions of very small gradients [104]. The softmax normalizes the resulting compatibility scores into a probability distribution, yielding a weighted sum over the values. In practice, this operation is applied in parallel across multiple independent *heads* (referred to as multi-head attention), allowing the model to simultaneously attend to information from different representational subspaces, such as syntactic structure in one head and semantic relatedness in another [104]. Figure 2.1 illustrates how attention and feed-forward layers are stacked within each encoder or decoder block, and Figure 2.2 makes the resulting token-to-token weights concrete.

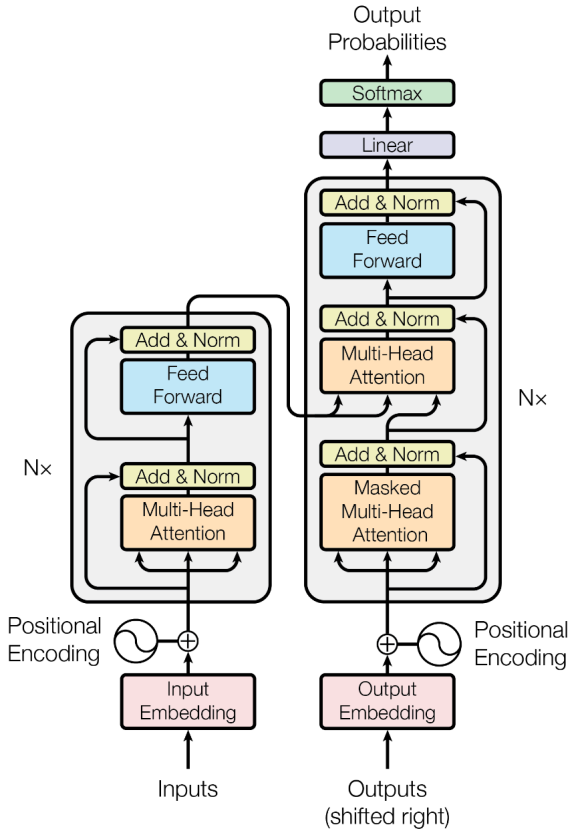

Figure 2.1: The transformer architecture, consisting of stacked encoder and decoder blocks each containing multi-head (self-)attention and feed-forward sub-layers [104]. Encoder layers process the full input sequence, and decoder layers additionally attend to encoder outputs.

	I	love	you
je	0.94	0.02	0.04
t'	0.11	0.01	0.88
aime	0.03	0.95	0.02

Figure 2.2: An example attention alignment matrix showing cross-attention weights between source tokens (e.g., "I love you") and target tokens (e.g., "je t'aime"). Each cell indicates the weight assigned by one target token (row) to a source token (column), corresponding to the normalized $\mathbf{QK}^\top$ scores in the formula above. Note that this example illustrates *cross-attention* between two sequences, whereas self-attention operates within a single sequence, but the same underlying computation applies in both cases. Also note that high attention weight does not necessarily indicate causal importance for the model's output. This is a limitation discussed further in Section 2.3 [91].

In practice, transformer models consist of many stacked attention layers combined with nonlinear transformations, where each layer progressively refines the internal representations. As information is repeatedly transformed and redistributed across layers, the contribution of individual inputs or intermediate representations becomes increasingly diffuse [12]. Consequently, model decisions emerge from complex, distributed interactions across many layers, making them difficult to trace to localized, human-interpretable rules. Diagnosing undesirable behavior in such models, therefore, requires reasoning about aggregate patterns and distributed signals, as failures typically cannot be traced to isolated components or decision paths in the way traditional debugging strategies (e.g., step-through debugging, unit tests) rely on [5].

Training objectives further contribute to this challenge. Common objectives, such as autoregressive next-token prediction or masked language modeling, optimize models to predict linguistic patterns across large corpora rather than to explicitly encode task-specific knowledge [26, 20, 77]. As a result, models learn to exploit statistical regularities across training data. While this enables broad generalization, it can also lead models to rely on patterns that do not align with intended task semantics [14]. While this allows strong performance across diverse tasks, it can also create ambiguity about what a model is expected to know in a particular context. A model may produce correct outputs for reasons that do not align with intended task semantics, or fail in cases where the required capability was never explicitly reinforced during training. This disconnect complicates diagnosis, as failures cannot always be attributed to a clearly missing or mislearned function.

Together, these architectural properties mean that diagnosis cannot rely on isolating individual causes or inspecting discrete decision rules. Instead, practitioners must reason about aggregate behavioral patterns, form testable hypotheses about model weaknesses, and iteratively refine their understanding through structured probing. This motivates the diagnostic approaches and support mechanisms discussed in subsequent sections.

2.2. Understanding and Categorizing Failures in LLM Systems

Before reviewing how failures in language models are categorized, it is necessary to clarify what constitutes a *failure* in machine learning systems as a specific form of undesirable behavior. Following prior work on failure handling in deep learning systems, a *model failure* can be understood as an externally observable behavior that deviates from requirements or expectations [9]. Failures may arise from various underlying causes, including data quality issues, limitations in model architecture or training, misalignment between task requirements and learned representations, or interactions across the deployment pipeline. Importantly, failures may occur even when the implementation code executes correctly, and conversely, correct outputs do not guarantee the absence of problematic underlying mechanisms.

This distinction is particularly salient for large language models. Due to their scale, distributed representations, and broad generalization, undesirable behavior in LLMs often manifests as subtle or context-dependent failures rather than explicit errors. As a result, a growing body of work has sought to move beyond aggregate performance metrics and develop methods for characterizing, surfacing, and handling failures in more systematic ways.

2.2.1. Failure Characterization

One prominent strand of work approaches failure categorization through *behavioral testing*. Ribeiro et al. [81] introduce CheckList, which organizes tests into categories such as invariance, directional expectations, and minimum functionality, providing a systematic way to specify and probe undesirable model behaviors. Subsequent work has expanded this approach by introducing semi-automated or fully automated behavioral testing pipelines to reduce reliance on manual test construction [45, 122, 35]. Other studies focus on domain-specific failure characterization, for example, in clinical applications or hate speech detection, where task context plays a central role in defining what constitutes undesirable behavior [103, 83, 43].

While effective at exposing brittle or inconsistent behavior, behavioral testing approaches place demands on practitioner expertise [81]. Defining meaningful tests requires prior knowledge of potential failure modes, and even extensive test suites cannot guarantee comprehensive coverage of a model's

behavior. Moreover, such methods primarily identify *where* a model fails, offering limited support for understanding *why* it fails.

Other work focuses on narrow but relevant aspects such as prompt engineering defects, highlighting how seemingly minor design choices can lead to systematic failure patterns [102, 110]. Surveys of prompt engineering techniques show how prompt design can mitigate issues such as hallucination, reasoning errors, or misinterpretations of user intent [86]. While these techniques are often effective in practice, they function primarily as mitigation strategies. They adjust the model's behavior at inference time, but do not directly address underlying model limitations or provide insight into the causes of undesirable behavior.

Beyond black-box behavioral testing, some approaches incorporate internal model information to better characterize failures. For instance, Sekhon et al. [89] propose a suite of white-box[1] coverage metrics that can be used alongside behavioral tests to identify underexplored regions of model behavior. These methods aim to improve test effectiveness but, on their own, do not prescribe how practitioners should interpret or act on observed failures.

Recent work has proposed explicit failure taxonomies for LLM-based systems to structure reasoning about undesirable behavior. For instance, Vinay [106] distinguishes between model-level, system-level, and operational failures, acknowledging that undesirable behavior may arise from interactions across the pipeline rather than solely from model inference. This perspective aligns with earlier work on machine learning, which has highlighted how data dependencies, configuration choices, and pipeline complexity can accumulate into forms of technical debt, even when individual components appear to function as intended [88]. While such taxonomies broaden the scope of failure analysis, they often operate at an abstract level, making direct diagnostic action challenging.

Taken together, existing approaches provide valuable tools for identifying and categorizing failures in LLM systems across behavioral, architectural, and system-level perspectives. However, they are primarily designed for failure identification rather than sustained diagnostic investigation. How practitioners should move from observing a failure to understanding its underlying cause by integrating evidence across methods, forming hypotheses, and iterating remains largely unspecified. Across these approaches, three recurring properties of LLM failures emerge that complicate diagnosis: failures are diverse in form, spanning hallucinations, instruction neglect, distributional brittleness, and harmful outputs; they are context-dependent, defined relative to task requirements, user expectations, and deployment setting; and they are rarely attributable to a single deterministic cause, arising instead from complex interactions across model architecture, training data, and context. This gap motivates the diagnostic framework and toolkit introduced in subsequent chapters, which aim to provide structured support for this diagnostic process.

2.3. Explainability and Diagnosis

Explainable Artificial Intelligence (XAI) aims to make the behavior of complex machine learning models more transparent and understandable. This section reviews the landscape of XAI methods, their adaptation to language models, and their limitations in the context of diagnosis.

2.3.1. Explainable AI: Concepts and Methods

XAI methods seek to provide insight into how and why a model produces a particular output. Commonly cited goals include transparency, accountability, and support for human decision-making [122]. Across machine learning more broadly, XAI techniques are often categorized along several dimensions, such as local versus global explanations, model-agnostic versus model-specific methods, and black-box versus white-box approaches [66].

Widely used explainability techniques include feature attribution methods (e.g., LIME [80], SHAP [57]), saliency-based approaches (e.g., Grad-CAM [90], Integrated Gradients [98]), and example-based explanations (e.g., influence functions [53], counterfactuals [107]). These methods aim to identify which input features or model components are most influential for a given prediction. While such techniques

[1] White-box methods refer to techniques that leverage internal model information, such as activations or weights, as opposed to black-box methods that rely solely on input-output behavior [66].

have been applied successfully in settings with relatively interpretable features, their explanatory value becomes more ambiguous in high-dimensional, complex models, such as LLMs [122]. This is due in part to the distributed nature of learned representations in deep neural networks and the difficulty of mapping low-level features to human-understandable concepts.

A prominent critique in the XAI literature concerns the distinction between plausibility and faithfulness. A *plausible* explanation appears intuitive or convincing to practitioners, while a *faithful* explanation accurately reflects the model's actual decision-making process. Critically, these properties do not necessarily align: many XAI methods optimize for plausibility, producing explanations that seem reasonable yet fail to capture the model's actual causal mechanisms [47, 85]. In addition, interpreting explanations often requires substantial expertise, and there is limited consensus on how to evaluate the quality or usefulness of explanations across tasks and practitioners [49]. These challenges complicate the use of XAI as a standalone tool for understanding model failures.

2.3.2. Interpretability and Explainability for Language Models

Applying XAI methods to language models introduces additional challenges. Unlike domains with continuous input features, language models process discrete token sequences via learned embeddings, producing high-dimensional representations distributed across multiple nonlinear layers. As a result, traditional feature attribution techniques may yield explanations that are difficult to interpret or misleading in practice [71].

Attention-based methods are among the most widely used interpretability techniques for transformer-based models [71]. These methods visualize attention weights to indicate which tokens a model attends to when producing an output, revealing the internal mechanisms by which the model processes information. However, prior work has shown that attention weights often do not reliably correspond to causal importance and do not provide reliable explanations of why a model produces specific outputs [91, 71]. Similar concerns apply to other attribution methods. Gradient-based approaches (e.g., Integrated Gradients [57]) and perturbation-based approaches (e.g., LIME [80]) may produce inconsistent explanations, yielding different attributions for similar inputs or across repeated analyses [4].

Recent empirical studies evaluating XAI techniques for language models further highlight these limitations. For example, comparative analyses show that different methods can produce divergent insights, and that their effectiveness varies across tasks such as robustness assessment or human reasoning agreement [61]. Moreover, XAI outputs that are informative at a token or sentence level may fail to capture higher-level behavioral patterns relevant for diagnosing failures [49].

These challenges are further amplified in specialized settings, such as multilingual or domain-specific language models, where XAI methods may not transfer cleanly across languages or contexts [79]. Together, these findings suggest that while XAI methods can surface useful signals, they offer only a partial, and sometimes unreliable, view of LLM behavior.

2.3.3. Diagnosis of Undesirable Behavior

While XAI focuses on rendering aspects of a model's behavior more transparent, it primarily answers descriptive questions about model behavior. Diagnosis, by contrast, involves answering contrastive questions such as *"Why did the model produce an incorrect output in this instance, as opposed to a correct one?"* This requires moving beyond explanatory artifacts to form and test hypotheses about recurring failure patterns, integrate evidence across methods, and reason about expected versus observed behavior. A separate body of work has examined diagnosis as such a process, distinct from explanation.

An empirical study of machine learning practice highlights that diagnosing model failures differs fundamentally from traditional software debugging. Amershi et al. [5] show that practitioners debugging machine learning systems must reason about data, learned behavior, and uncertainty rather than deterministic control flow. Failures often arise from complex interactions among model architecture, training data, and deployment context, making them difficult to localize with conventional debugging techniques. As a result, practitioners rely on exploratory experimentation, iterative testing, and domain knowledge to form and refine hypotheses about potential causes.

Zhang et al. [120] explicitly conceptualize diagnosis as a structured investigative workflow, introducing a

model-agnostic framework that supports diagnosis through interactive exploration, contrastive analysis, and hypothesis testing. Rather than treating explanations as end products, their approach emphasizes diagnosis as a process of asking *why* questions, such as why a model behaves differently across similar inputs, and using targeted probes to explore these differences. This framing aligns diagnosis with scientific inquiry, where observations motivate hypotheses that are subsequently tested and refined.

In the context of natural language processing, Wu et al. [116] propose Errudite, a system designed explicitly for hypothesis-driven error analysis. Errudite enables practitioners to construct, test, and revise hypotheses about model failures by comparing contrasting examples, inspecting error slices, and iteratively refining diagnostic questions. Importantly, this approach treats errors not as isolated events but as manifestations of broader behavioral patterns, reinforcing the notion that diagnosis operates at a level of recurring failures rather than individual predictions.

Balayn et al. [10] investigate how practitioners employ XAI techniques when identifying bugs in deep learning (vision) models. Their findings suggest that XAI outputs are rarely sufficient on their own. Instead, they function as diagnostic probes that provide partial evidence for or against emerging hypotheses. Practitioners combine interpretability and explainability methods with behavioral testing, domain knowledge, and iterative experimentation to understand the root causes of failures. This reinforces the view that diagnosis subsumes XAI rather than being reducible to it.

Taken together, this literature characterizes diagnosis as a process with several recurring properties: it is hypothesis-driven, grounded in expectations about model behavior, and unfolds through structured, iterative exploration rather than isolated analyses. Diagnosis draws on a variety of tools, including interpretability and explainability methods, but also requires structure, guidance, and contextual reasoning to connect observations to potential underlying causes. Despite growing recognition of its importance, existing diagnostic approaches remain fragmented and are rarely integrated into cohesive, practitioner-facing workflows.

2.4. Empirical Insights into Practitioners' Needs for Diagnosis

In addition to methodological work on explainability and diagnosis, several empirical studies have examined how practitioners investigate and reason about behavior in machine learning systems. These studies provide context for understanding how existing tools and methods are used in practice, and where mismatches arise between research assumptions and real-world workflows.

Across multiple empirical studies, a consistent pattern emerges: practitioners working with machine learning models rely heavily on manual inspection, ad hoc experimentation, and domain knowledge when investigating failures, with limited use of formalized diagnostic workflows [5, 9, 82, 27]. Failure handling workflows are typically performed manually and without ML-specific methods, even as the proliferation of interpretability and explainability techniques might suggest otherwise [9, 27]. As Amershi et al. [5] document, diagnosing machine learning systems fundamentally differs from traditional software debugging: practitioners must reason about data quality, learned representations, and statistical uncertainty rather than deterministic control flow, yet often lack structured support for doing so systematically.

A qualitative study conducted at a software company provides detailed insight into these patterns in the context of language model development [68]. Based on semi-structured interviews with NLP data scientists, the study reports that model investigation in practice is largely exploratory and iterative. Practitioners frequently rely on manual inspection, small-scale experiments, and comparisons across prompts or configurations to understand model behavior, rather than following formalized analytical pipelines. The study further observes that while XAI methods are generally known to practitioners, they are not consistently integrated into everyday diagnostic workflows. Participants reported difficulties in determining when XAI techniques are appropriate, interpreting their outputs reliably, and translating them into concrete next steps. As a result, XAI tools were often used opportunistically, in response to specific questions, rather than as part of a systematic investigative process.

These patterns extend to how practitioners interact with XAI tools more broadly. Kaur et al. [50] conducted a contextual inquiry and large-scale survey of data scientists using XAI tools such as SHAP and generalized additive models. Their results indicate that practitioners may over-trust XAI outputs,

misunderstand visualizations, or apply XAI tools in ways that do not align with their intended semantics. This suggests that access to XAI outputs alone does not guarantee correct or effective use, particularly in complex diagnostic scenarios.

These empirical observations resonate with broader conceptual critiques of XAI methods. Rudin [85], in a position paper arguing for inherently interpretable models, argues that explanations of black-box models are often inherently misleading, particularly in high-stakes settings, because they do not faithfully reflect the actual decision mechanisms of the models. From this perspective, misinterpretation of XAI outputs is not merely a matter of insufficient expertise, but a consequence of relying on explanatory artifacts that lack causal or semantical grounding. This critique underscores a broader concern relevant to this thesis: XAI methods, when treated as substitutes for understanding, can foster false confidence rather than support reliable reasoning about model behavior.

A recent scoping review that synthesizes research on XAI frameworks from a human-centered perspective corroborates these empirical and conceptual findings [24]. Analyzing a broad set of XAI frameworks, the review highlights recurring gaps between framework design and practical use, including limited support for iterative workflows, insufficient attention to cognitive load, and a lack of guidance on integrating XAI outputs into decision-making processes. While not specific to language model diagnosis, these patterns reinforce the importance of considering practitioner context when designing XAI and diagnostic support.

Taken together, this body of empirical work suggests that practitioners already engage in diagnostic reasoning when investigating model behavior, but often do so informally and in a loosely structured manner. Existing XAI tools and frameworks provide potentially useful signals but offer limited support for sustained reasoning about recurring failures, hypothesis formation, and integrating evidence across experiments. The following section examines the concrete tooling landscape practitioners navigate during diagnostic work, surveying the available tool categories and the systematic gaps that emerge across them.

2.5. Tooling Landscape for Diagnostic Work

Practitioners diagnosing undesirable behavior in large language models rarely rely on a single method, tool, or interface. Instead, diagnostic work unfolds across a spectrum of environments and tools that vary in structure, flexibility, and intended use. Many of these tools were not designed explicitly for diagnosis, yet can be appropriated by practitioners as part of diagnostic workflows. Understanding this tooling landscape is essential for identifying where existing support is effective and where additional structure may be beneficial.

At one end of the spectrum, diagnostic work is highly exploratory and loosely structured. Practitioners experiment with inputs, inspect outputs, and iteratively refine their understanding of model behavior using general-purpose programming environments. At the other end, more specialized tools provide predefined abstractions for testing, evaluation, and model interpretation. Movement along this spectrum reflects a trade-off between flexibility and structure: less structured environments allow for open-ended exploration, while more structured tools support reproducibility, comparison, and systematic analysis.

Within this spectrum, several categories of tools and interfaces are available during diagnostic work. Each category offers distinct affordances (the diagnostic actions it enables) and limitations in supporting diagnostic reasoning. The following subsections survey these categories, highlighting their roles in relation to practitioner workflows. The complete list of tools discussed here is provided in Appendix A.

2.5.1. General-Purpose Environments for Exploratory Diagnosis

General-purpose environments such as Jupyter notebooks and custom scripts are widely used during model development and debugging [52, 76]. These environments allow practitioners to probe model behavior, test hypotheses, and interactively inspect intermediate results with much control. Their flexibility makes them well-suited for early-stage investigations, particularly when failure patterns are poorly understood, highly context-dependent, or unknown. Key features such as inline code execution, rich media rendering, and open integration with libraries facilitate iterative exploration and experimentation.

However, the lack of inherent structure in general-purpose environments imposes the risk of significant

overhead that accumulates across diagnostic sessions. Each new investigation requires practitioners to manually reconstruct their analytical infrastructure: loading datasets, configuring metrics, setting up comparison baselines, etc. [115]. This setup burden is compounded when working across multiple projects or when collaborating, as there are no standardized conventions for capturing or sharing diagnostic configurations.

The cognitive load of maintaining diagnostic state (tracking which hypotheses have been tested, which experiments yielded insights, or how findings relate to one another) falls heavily on the practitioner. As diagnostic complexity increases, insights can become fragmented across notebooks, scripts, logs, and ad hoc documentation, making it progressively more challenging to maintain a coherent view of emerging failure patterns. While version control systems can track code changes, they do not capture the reasoning process or evolving understanding of model behavior. Additionally, reproducing past diagnostic sessions can require reconstructing context from sparse comments or commit messages when returning to a problem after time has passed or when documentation is incomplete. Thus, while exploratory environments excel at initial hypothesis generation, their lack of structure can create friction that scales poorly with diagnostic complexity and collaboration needs.

2.5.2. NLP and Model Analysis Libraries

A wide range of NLP and machine learning libraries provides reusable components for model analysis, evaluation, and experimentation. These include libraries for computing metrics (e.g., scikit-learn [75], NLTK [13]), inspecting embeddings (e.g., spaCy [32], Gensim [78]), analyzing attention patterns (e.g., BertViz [105], Captum [54]), or running controlled perturbations (e.g., TextAttack [65]). Such tools can substantially reduce implementation effort and support more systematic experimentation compared to fully ad hoc approaches.

While libraries offer powerful analytical capabilities, their atomized design creates integration and interpretation challenges that extend beyond individual analyses. Each library typically follows its own API conventions, data formats, and output structures, requiring practitioners to write substantial glue code to chain analyses together or compare results across different methods. This fragmentation means that insights remain siloed: an attention analysis from BertViz exists independently from perturbation results from TextAttack, with no built-in mechanism for relating findings or detecting patterns across methods [105, 65].

The expertise barrier can also be substantial. Practitioners must understand not only how to invoke each library's functions, but also the theoretical foundations and limitations of the methods themselves—a challenge that extends the interpretation difficulties identified for XAI tools in Section 2.3. Misapplication or misinterpretation of complex techniques (e.g., using attention weights as explanations [48, 114]) can lead to misleading conclusions [50, 85]. Yet libraries rarely guide when methods are appropriate or how to validate their outputs, placing the burden of analytical design entirely on the practitioner.

2.5.3. Behavioral Testing and Evaluation Frameworks

Behavioral testing and evaluation frameworks introduce additional structure by enabling practitioners to define test suites that probe model behavior under specific conditions. Tools such as CheckList [81], Robustness Gym [38], and TextFlint [112] facilitate systematic evaluation across dimensions, including robustness, invariance, and sensitivity to perturbations. These frameworks are particularly effective at revealing brittle or inconsistent behavior that may not be apparent from aggregated metrics [81]. Practitioners can leverage predefined test templates or create custom tests to target suspected failure patterns, facilitating more focused investigation.

However, several practical limitations constrain their utility in broader diagnostic contexts. First, test suite design requires substantial upfront investment: practitioners must anticipate potential failure patterns, craft appropriate test cases, and establish clear evaluation criteria before gaining any diagnostic insight, which can be misaligned with exploratory diagnostic work where failure patterns are unknown and evolve as understanding deepens. Second, these frameworks are typically designed for batch evaluation rather than continuous investigation. They excel at comparing model versions or validating fixes, but offer limited support for iterative hypothesis refinement during active diagnostic work. Third, the results they produce (e.g., pass/fail counts, robustness scores) require interpretation and contextualization that the framework itself does not always provide. A model failing 30% of invariance tests

may indicate a systematic problem, a dataset artifact, or a test design issue; disambiguating these possibilities requires additional investigation beyond the framework's scope. From a diagnostic perspective, behavioral testing frameworks are valuable for confirming suspected issues or measuring specific behavioral dimensions, but do not scaffold the exploratory reasoning needed to discover root causes.

2.5.4. Specialized Diagnostics and XAI Interfaces

More specialized interfaces aim to support inspection and analysis through visualizations, attribution methods, or interactive exploration. Interpretability toolkits and error analysis systems (e.g., LIT [100], Ecco [3], AllenNLP Interpret [109]) can help practitioners examine model behavior from multiple perspectives, such as input relevance, error distribution, or contrastive examples. These tools often integrate various analyses into a unified interface, facilitating cross-referencing and comparison.

However, specialized interfaces face several recurring challenges that limit their practical adoption and effectiveness. First, they often demand substantial expertise: understanding which visualizations or attribution methods are appropriate for a given question, recognizing when outputs may be misleading, and knowing how to validate findings. This expertise barrier is compounded by insufficient guidance within the tools themselves. Second, the outputs generated (attribution heatmaps, counterfactual examples, attention visualizations) answer specific, narrow questions but offer little guidance on what to investigate next. A practitioner may discover that a model pays close attention to particular tokens. Yet, the interface provides no framework for translating this observation into actionable diagnostic steps or connecting it to other evidence. Third, these tools exist as isolated analytical endpoints rather than components of an integrated diagnostic ecosystem. Insights from a single interface must be manually transcribed, contextualized, and linked to findings from other analyses. As a result, specialized interfaces can deepen understanding of specific behavioral aspects but do not scaffold the broader process of diagnostic reasoning: forming hypotheses, designing multi-faceted investigations, integrating diverse evidence, and arriving at coherent explanations of model failures.

2.5.5. Synthesis: Systematic Gaps in the Tooling Landscape

Across this spectrum, existing tools provide essential building blocks for diagnosing undesirable model behavior. However, several systematic challenges emerge that extend beyond the limitations of individual tool categories.

First, there is a common **expertise and interpretation gap**. Each tool category demands substantial specialized knowledge: understanding when gradient-based attribution is reliable, how to design meaningful behavioral tests, or which analytical methods are appropriate for a given diagnostic question. Yet, this expertise is insufficiently scaffolded by the tools themselves. Practitioners are left to navigate complex analytical landscapes with minimal guidance about when methods are applicable, how outputs should be interpreted, or what their limitations are.

Second, the tooling landscape suffers from **fragmentation and integration costs**. Each tool category operates in isolation with its own conventions, data formats, and conceptual models. Insights generated by one tool cannot easily be related to findings from another. Practitioners must write substantial glue code, manually transfer context between environments, and maintain their own infrastructure for synthesizing evidence across analyses, meaning the cognitive burden of constructing coherent diagnostic narratives falls on individual practitioners rather than being supported by the tooling ecosystem.

Third, there is a fundamental mismatch between **tool affordances and diagnostic workflows**. General-purpose environments excel at exploration but impose heavy setup and maintenance overhead. Evaluation frameworks provide rigor but require upfront investment that may be misaligned with exploratory diagnostic work. XAI interfaces generate sophisticated analyses but offer little to no guidance for what to investigate next or how to integrate findings into broader reasoning processes. Across categories, tools are optimized for isolated analytical tasks rather than the continuous, hypothesis-driven process that characterizes diagnostic work in practice.

These three challenges inform both the diagnostic framework introduced in Chapter 3 and the design of the toolkit presented in Chapter 4.

3

Diagnostic Framework

As established in the preceding chapter, undesirable behavior in transformer-based language models is diverse in form, context-dependent in its manifestation, and rarely attributable to a single deterministic defect. These properties collectively make diagnosis a non-trivial investigative challenge: practitioners often begin from an intuition that a model behaved unexpectedly and must then translate this intuition into a structured investigation across examples, tests, and competing explanations. This chapter introduces a diagnostic framework intended to scaffold that reasoning process.

The framework is designed to provide *structure without rigidity*. It does not prescribe a fixed diagnostic algorithm, nor does it claim exhaustive coverage of all possible failure modes. Instead, it provides a set of concepts and relationships that help practitioners (i) articulate what is observed, (ii) gather and organize diagnostic evidence, and (iii) form and refine hypotheses about recurring failure patterns.

3.1. Introduction

The framework introduced in this chapter has three central properties.

First, it is **flexible**. The sequence from symptoms through signs to failure patterns represents a suggested default that practitioners may enter or exit at any step, depending on what evidence is already available. Diagnostic work in practice is iterative and opportunistic: hypotheses shift as evidence accumulates, and the most useful next probe depends on what has already been learned.

Second, it is **extensible**. Undesirable behavior is defined relative to task context, user expectations, and practitioner intent. New failure manifestations can emerge as contexts change. The symptoms, signs, and failure patterns provided in this work are intended as starting points. Practitioners are encouraged to identify additional concepts relevant to their specific context.

Third, the framework provides a **shared vocabulary coupled to a workflow**. The vocabulary names the concepts used to describe observations and evidence. The workflow describes how these concepts can be applied together to guide diagnosis.

These three properties distinguish the framework from existing approaches reviewed in Chapter 2. Behavioral testing approaches such as CheckList [81] provide structured test categories but do not offer a diagnostic workflow or a vocabulary for organizing evidence across methods. Abstract failure taxonomies [106] categorize undesirable behavior at a conceptual level but do not guide practitioners through the investigative process from observation to hypothesis. Prior work applying the SK/RK distinction [10, 93] (covered in detail in Section 3.4) provides a useful interpretive lens but does not operationalize it into a practitioner-facing vocabulary and workflow for transformer-based language models specifically. This framework integrates these elements into a cohesive structure that supports the practical needs of diagnosis in this domain.

3.2. Core Concepts

The framework is organized around three core concepts: *symptoms*, *signs*, and *failure patterns*. This structure reflects how diagnosis typically unfolds from directly observable mismatches, through instrumented evidence gathered via diagnostic methods, to plausible hypotheses about recurring failure patterns. This progression mirrors medical diagnosis in an abstract sense: clinicians begin with patient-reported complaints and visible anomalies, perform examinations and tests, and then formulate and revise diagnostic hypotheses. The analogy is not intended to claim equivalence between biological and model mechanisms, but rather to motivate a familiar, pragmatic structure for LLM diagnosis. One meaningful difference is that clinical symptoms are typically self-reported by a patient, whereas model symptoms are identified by an external observer. The parallel is therefore structural rather than mechanistic.

3.2.1. Symptoms

Definition. A symptom is a directly observable indicator that a model's behavior mismatches contextual expectations. Symptoms can typically be identified from model inputs and outputs without specialized instrumentation.

Rationale. Practitioners rarely begin the diagnostic process from internal model mechanisms. Instead, diagnosis usually starts with observable mismatches between expected and produced behavior: an incorrect output, a refusal that should not occur, a summary that omits key information, or an unjustifiably confident answer. Such observations are often sufficient to trigger an investigation but insufficient to explain *why* the model failed. Symptoms, therefore, serve as starting points, motivating diagnostic questions and aiding in the selection of initial probes. Importantly, symptoms are *descriptive* rather than explanatory: they state what appears to be wrong, not why it is wrong.

Examples. Table 3.1 lists the symptoms used in this work. This list is non-exhaustive and intended as a starting point. Practitioners are encouraged to identify additional symptoms relevant to their context.

Symptom	Description
Incorrect Output	For cases in which there is a definitive ground truth, e.g., mathematical problems.
Poor Performance	The model does not satisfy the metrics of interest. This is application-dependent.
Repetitive Output	The model starts repeating output tokens at any point during generation.
Instruction Negligence	An instruction (or a series thereof) is not followed properly, or ignored altogether. When a series of instructions is provided, the order is ignored as well.
Exaggerated Safety	Safe prompts are refused because, for instance, they follow the same structure as unsafe ones or mention sensitive topics.
Harmful Output	The outputs are toxic, reflect societal biases or stereotypes, or promote unethical practices. This is application-dependent.
Vague Output	The output does not provide an answer to the query, is phrased ambiguously, lacks detail, or a combination thereof.

Table 3.1: Symptoms of undesirable model behavior (non-exhaustive).

3.2.2. Signs

Definition. A sign is an indirectly observable indicator of potential model weakness that is surfaced through diagnostic methods, instruction, or expert analysis. Signs typically require an explicit probe or measurement to observe.

Rationale. To deepen the investigation beyond what symptoms alone reveal, practitioners employ diagnostic methods that produce instrumented evidence: attribution signals, consistency checks, perturbation tests, slice analyses, or benchmark results. These outputs are not always directly visible in ordinary model use and often require specialized methods or expertise to interpret. Signs differ from

symptoms in that they provide evidence about the model's behavior under probing and are therefore helpful in evaluating competing hypotheses about what might explain observed symptoms.

Examples. Table 3.2 lists the signs used in this work. As with symptoms, this list is non-exhaustive, and additional signs may be relevant depending on the diagnostic task and available instrumentation.

Sign	Description
Fragility to Natural Alterations	The model output changes given data samples that are semantically equivalent and grammatically correct.
Intrinsic Hallucination	The generated output contradicts the source content.
Extrinsic Hallucination	The generated output can neither be supported nor contradicted by the source content.
Mis-calibrated Model	The output probabilities are not consistent with its output. This can manifest as either under-confident or over-confident models.
Wrong Task Decomposition	The model is either unable to decompose tasks properly or does so incorrectly.
Misplaced Attention	Inspection (at selected layers) of attention scores reveal that the model incorrectly attends to input tokens (e.g., irrelevant ones).
Misplaced Token Attribution	Token-level attribution scores (output $\rightarrow$ input) reveal that the model incorrectly attends to input tokens (e.g., irrelevant ones).

Table 3.2: Signs surfaced through diagnostic methods or instrumentation (non-exhaustive).

3.2.3. Failure Patterns

Definition. A failure pattern is a recurring and interpretable combination of symptoms and signs that suggests a plausible explanation for undesirable behavior across multiple instances.

Rationale. Once symptoms and signs have been gathered, practitioners attempt to integrate this evidence into a coherent, recurring explanation. Failure patterns are hypotheses and are not treated as definitive root causes or mechanistic diagnoses. Instead, failure patterns summarize what appears to be happening repeatedly and guide further testing and potential remediation, which falls beyond the scope of this thesis.

Examples. Table 3.3 lists the failure patterns used in this work. As with symptoms and signs, this list is non-exhaustive, and practitioners may identify additional failure patterns as their diagnostic context evolves.

Failure Pattern	Description
Shortcut Reliance	The model relies on spurious or shallow input cues (e.g., text structure or specific words) to generate its output.
Ignoring Input	The model ignores relevant parts of the input. These can be both instruction-related and content-related.
Stereotyping	The model relies on, amplifies, or generates text that contains stereotypes. These can be harmful ones or not.
Mode Collapse	The model produces repeated responses (or template-like outputs) regardless of the selected decoding strategy.
Reasoning Instability	The model is able to resolve input queries correctly, but its reasoning (and steps, if necessary) is inconsistent or incorrect.

Table 3.3: Failure patterns as recurring hypotheses supported by symptoms and signs (non-exhaustive).

Illustrative Mapping

Symptom: Instruction negligence.
Candidate probes: paraphrase the instruction; reorder instruction lists; introduce distractors; vary formatting.
Possible signs: fragility to natural alterations; misplaced token attribution (attribution mass on irrelevant spans); wrong task decomposition.
Plausible failure pattern hypotheses: ignoring input (instruction-related); reasoning instability (if the model sometimes complies via different intermediate steps).

3.3. Diagnostic Workflow

The framework does not prescribe deterministic mappings, such as ”Symptom A implies Failure Pattern B.” Instead, it provides a structure for organizing evidence and maintaining traceability between observations and hypotheses. Figure 3.1 gives a high-level overview.

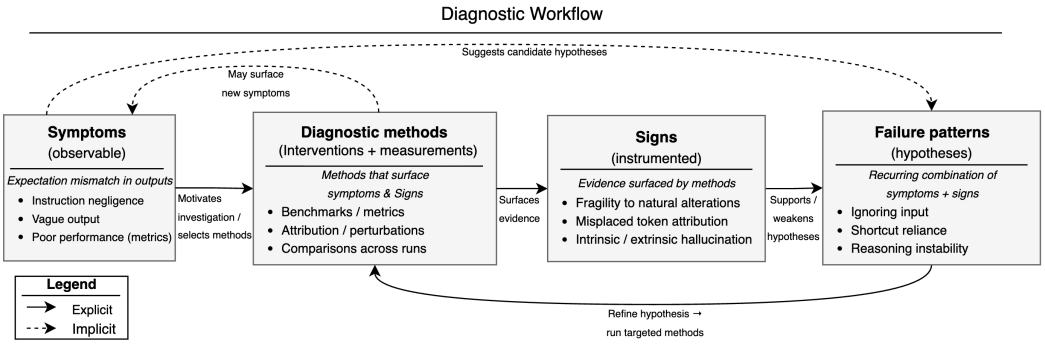

Figure 3.1: High-level structure of the diagnostic framework: symptoms motivate methods that surface signs, symptoms, and signs jointly support hypotheses about failure patterns.

In practice, diagnosis unfolds as an iterative, hypothesis-driven sequence of activities rather than a rigid pipeline; practitioners may move non-linearly depending on context and evidence:

1. **Trigger: observe unexpected behavior.** Diagnosis often begins when a practitioner identifies an expectation mismatch in model behavior (a symptom) with a specific task context.
2. **Articulate expectations and propose an initial hypothesis.** The practitioner specifies what the model should do or know in this context and forms an initial hypothesis about a plausible failure pattern.
3. **Select diagnostic probes to elicit evidence.** The practitioner uses diagnostic methods to surface signs relevant to the hypothesis. Method choice is guided by the type of evidence required (e.g., reliance, stability, sensitivity, distributional slice behavior).
4. **Interpret symptoms and signs jointly.** The practitioner evaluates whether the observed signs support, contradict, or refine the current hypotheses. This step often involves comparing instances, baselines, or controlled variants.
5. **Iterate: refine hypotheses through additional testing.** Based on accumulated evidence, the practitioner adjusts the hypothesis, runs additional probes, and searches for recurrence across examples until a plausible failure pattern stabilizes.
6. **Converge on a failure pattern hypothesis.** The primary output of the diagnostic process in this thesis is the identification of a plausible failure pattern that explains the observed undesirable behavior and can inform remediation outside the scope of this work. Partial localization, attributing the pattern to a specific model reliance or behavioral tendency, may also emerge from certain methods, but it is not guaranteed.

3.4. The Role of Should-Know and Really-Knows

The diagnostic framework is complemented by the distinction between what a model *should know* and what it *really knows*, adapted from prior work on model diagnosis [93, 10].

Should-Know (SK) refers to the knowledge, skills, or task understanding a model is expected to possess to satisfy contextual requirements. In practice, SK is expressed through task specifications, acceptance criteria, and expectations encoded in datasets, metrics, and behavioral tests.

Really-Knows (RK) refers to what the model demonstrably relies on as evidenced by its behavior under probes and across variations of inputs and conditions. RK is therefore established empirically, for example, through perturbation tests, attribution analyses, or controlled comparisons.

SK/RK functions as a diagnostic enabler, providing referential grounding that sharpens each step of the workflow. Without SK, a symptom is harder to interpret as an expectation mismatch because there is no explicit expectation against which to recognize it as a deviation, nor a clear basis for forming an initial hypothesis (workflow step 2). Without RK, a sign is harder to evaluate as evidence of genuine weakness because there is no empirical account of what the model actually relies on, making it difficult to assess whether the sign reveals a real problem or an artifact of the probe (workflow step 3). And without the gap between SK and RK, a failure pattern has no principled characterization: it is the observable divergence between what the model should do and what it demonstrably does that transforms a recurring observation into a testable diagnostic hypothesis (workflow steps 4–6).

3.4.1. Rationale for the Should-Know/Really-Know Distinction

The separation between SK and RK serves a critical diagnostic function: it prevents practitioners from conflating task requirements with model capabilities. Without this distinction, diagnosis often stalls at the observation that ”the model failed,” without clarifying *why*. By explicitly separating what a model is expected to do (SK) from what it demonstrably does (RK), practitioners can formulate more precise diagnostic questions:

- Does the model lack the required capability entirely? (SK exists, but no corresponding RK)
- Does the model possess the capability but fail to apply it? (RK exists in some contexts but not others)
- Does the model rely on spurious or unintended cues? (RK diverges from SK)

This framing transforms vague diagnostic intuition (”something is wrong”) into testable hypotheses about specific gaps or misalignments. It also structures the investigative sequence: first, articulate expectations explicitly (SK); then, probe empirically to reveal actual reliance (RK); and finally, interpret the relationship between the two.

3.4.2. Should-Know/Really-Know as a Diagnostic Enabler

Should-Knows/Really-Know is not an additional vocabulary concept alongside symptoms and signs. Rather, it operates as the interpretive layer that anchors the vocabulary and workflow, making their concepts more precisely actionable. It supports two core diagnostic activities that correspond directly to workflow steps 2 and 3:

- **Expectation articulation (SK):** making explicit what correct behavior requires in a given context, enabling symptoms to be recognized as expectation mismatches rather than isolated anomalies. This involves translating task requirements into testable criteria (e.g., ”the model should faithfully represent document content in summaries”), which in turn anchors the initial hypothesis formed in step 2.
- **Evidence gathering (RK):** using diagnostic methods such as perturbation tests, attribution analyses, or controlled comparisons to empirically establish what the model actually relies on when producing outputs (step 3). This reveals whether the model’s behavior aligns with the articulated expectations or reflects an alternative, potentially spurious reliance pattern.

Together, SK and RK determine the diagnostic significance of observed evidence: symptoms mark the visible surface of an SK/RK gap, signs characterize its nature, and failure patterns name the recurring gap itself.

3.4.3. Illustrative Example

Consider a summarization task where a model produces outputs that omit key entities mentioned in the source document (a symptom). A practitioner articulates the SK expectation: ”the model should rely primarily on document content rather than on position heuristics.” To test this, the practitioner applies attribution analysis (an RK probe) and observes that the model’s attention concentrates overwhelmingly on the document’s opening sentences, with minimal weight on later context containing the omitted entities (a sign). This reveals an SK/RK gap: while the task requires comprehensive content coverage (SK), the model demonstrably relies on position-based shortcuts (RK). The practitioners can then characterize this as a *shortcut reliance* failure pattern and test whether it recurs across examples.

This example illustrates how the SK/RK lens structures diagnosis: the expectation (SK) motivates a specific probe (RK method), whose results reveal a gap that explains the observed symptom and sign, and supports a failure pattern hypothesis.

3.5. Assumptions and Implications

The framework makes several assumptions.

First, it assumes that practitioners can recognize expectation mismatches and interpret diagnostic evidence meaningfully. While the framework aims to scaffold reasoning, it does not eliminate the need for practitioner judgment, particularly when evidence is ambiguous or conflicting.

Second, the framework assumes that mappings between symptoms, signs, and failure patterns are heuristic rather than deterministic. The goal of diagnosis is to surface and evaluate candidate causal explanations for observed behavior, and the framework provides starting points for that search, not prespecified causal rules. Evidence supports or weakens hypotheses; it rarely proves a single explanation.

Third, the framework focuses on model-level diagnosis. In practice, undesirable behavior may also arise from system-level factors (e.g., retrieval failures, prompt routing, post-processing). The framework can still be used to surface and organize evidence that suggests such issues, but it does not aim to model system-level failure mechanisms exhaustively.

These assumptions have direct design implications for tooling, each corresponding to a structural property of the diagnostic workflow described above:

- **Mixed expertise support:** diagnostic tooling should accommodate varying levels of XAI expertise through appropriate interaction design, providing scaffolding for practitioners who lack knowledge without restricting those who have it (reflecting steps 2–3, where expertise gaps are most likely to surface).
- **Iterative evidence accumulation:** tools should support repeated probing, comparison across runs, and persistent organization of findings (reflecting the iterative refinement loop in step 5).
- **Non-linear navigation:** tools should not enforce a strict sequence, since diagnostic reasoning often loops and branches (reflecting the flexible entry and exit points described throughout the workflow).

3.6. Summary

This chapter introduced a diagnostic framework for investigating undesirable model behavior through three concepts: symptoms, signs, and failure patterns. The framework structures diagnosis as an iterative process of observation, probing, and hypothesis refinement rather than as a prescriptive mapping. The SK/RK distinction complements the framework by separating the articulation of expectations from empirical evidence about model reliance. Together, these elements motivate the operationalization of the toolkit presented in Chapter 4.

4

Toolkit Architecture and Operationalization

This chapter presents the diagnostic toolkit developed to operationalize the framework introduced in Chapter 3. The toolkit is designed as a *research instrument*: a concrete instantiation of the framework that enables practitioners to perform diagnosis with structured support for running XAI methods, comparing results, and preserving intermediate reasoning through annotations, making reasoning traceable across iterations. The toolkit is implemented as a containerized system with a Django-based workspace layer and a FastAPI-based execution layer, supporting a range of diagnostic methods and metrics.

The toolkit targets transformer-based models and datasets integrated through Hugging Face. In terms of access assumptions, the workflow supports both black-box and white-box diagnostic methods (see Chapter 2). Most implemented methods operate in a black-box manner, requiring only input and output access. A subset of attribution-based methods (AttnLRP, SHAP, and Integrated Gradients) requires white-box access to the internal model. The access requirements of each method are detailed in Section 4.5.

4.1. Design Goals and System Requirements

The toolkit design is grounded in the diagnostic needs and tooling gaps synthesized in Chapter 2. In particular, prior work highlights that diagnostic practice is iterative, exploratory, and hypothesis-driven, yet existing tools often provide isolated analyses with limited support for organizing evidence, preserving diagnostic state across iterations, or guiding next investigative steps. A further gap is the absence of in-tool, structured support for distinguishing task expectations from actual model behavior (SK/RK) and for reasoning about outputs in context rather than in isolation. The toolkit, therefore, aims to address these gaps by embedding structure without enforcing a rigid process. The following five goals were derived from these gaps to guide the design:

- **Support structured diagnostic reasoning**: provide explicit representations for symptoms, signs, and failure patterns to help practitioners externalize hypotheses and evidence.
- **Enable SK/RK-driven investigation**: support both expectation specification (Should-Know) and empirical probing (Really-Know), and make their relationship inspectable.
- **Reduce cognitive overhead**: preserve diagnostic state across iterations via persistent jobs, recorded configurations, and traceable findings.
- **Support non-linear workflows**: allow branching, revisiting prior runs, and opportunistic probing consistent with real diagnostic practice.
- **Contextual interpretation**: support interpretation through comparison views, tagging, and lightweight reporting rather than isolated analytical outputs.

These goals informed both high-level architectural decisions and specific user-experience (UX) features, as described in the following sections. The design process involved iterative refinement, with early prototypes evaluated against these goals to ensure alignment with practitioner needs and diagnostic realities. The resulting toolkit is intended as a proof-of-concept that demonstrates how structured support can enhance diagnostic workflows, while leaving room for future extensions and refinements based on user feedback and evolving diagnostic challenges.

4.1.1. System Requirements Hierarchy

To translate these goals into implementable targets, system requirements were specified through a tiered set of user stories and accompanying functional/non-functional requirements.

Tiered user stories. User stories were written at three levels of granularity: (i) *epics* capturing very broad practitioner goals, (ii) *feature-level stories* describing key system capabilities, and (iii) *task-level stories* describing concrete UI or service interactions. This tiered structure was used to maintain traceability between practitioner goals, architectural decisions, and specific interactions supported by the toolkit. The alignment of these tiers with the architectural documentation method is described in Section 4.2. The list of user stories can be found in Appendix B, organized by tier.

Functional and non-functional requirements. In addition to user stories, the system was specified using functional requirements (FRs) and non-functional requirements (NFRs), which covered core behaviors (e.g., test execution, result inspection, comparison, reporting, and model/dataset handling) and quality attributes (e.g., usability, reliability, privacy, and portability). During development, requirement coverage was tracked to distinguish implemented, partially implemented, and out-of-scope requirements. The complete set of FR/NFR specifications is included in Appendix B.

4.2. Architecture

The toolkit architecture was designed and documented using the **C4 model** for software architecture diagramming [18, 19]. The C4 model organizes architecture documentation into four progressive levels of abstraction: system context (C1), containers (C2), components (C3), and code (C4). This allows each diagram to target a different audience and level of detail. It was selected here because its hierarchical structure both supports communicating system boundaries and internal component interactions without requiring a single, overloaded diagram. It also enables traceability between practitioner goals, architectural decisions, and concrete interactions.

The three tiers of user stories defined in Section 4.1 map directly onto the C4 levels: epics correspond to the context level (C1), feature-level stories to the container level (C2), and task-level stories to the component level (C3). This alignment ensured that requirement coverage could be tracked against the same structural hierarchy used to document the system. Specifically, a context, container, and component diagram were produced to describe the system at increasing levels of detail. The code-level view (C4) was not produced, as C4 treats this level as on-demand detail typically accessible through the implementation and development environment.

In this chapter, we present the container-level view to communicate the main system boundaries and responsibilities. The context and component diagrams are included in Appendix C.

4.2.1. Container-Level Architecture

At a high level, the toolkit is implemented as a containerized system with separation between (i) a workspace layer responsible for user interaction and persistence of diagnostic artifacts, and (ii) an execution layer responsible for running inference and diagnostic methods. Figure 4.1 shows the container diagram, which illustrates the main components and their interactions.

Workspace layer (Django + SQLite). A Django-based application provides the user interface and project workspace. Project artifacts (e.g., registered models, dataset configurations, and executed job metadata) are stored in a SQLite database. This lightweight database provides robust local persistence while meeting the resource constraints of the target virtual machine (VM) environment.

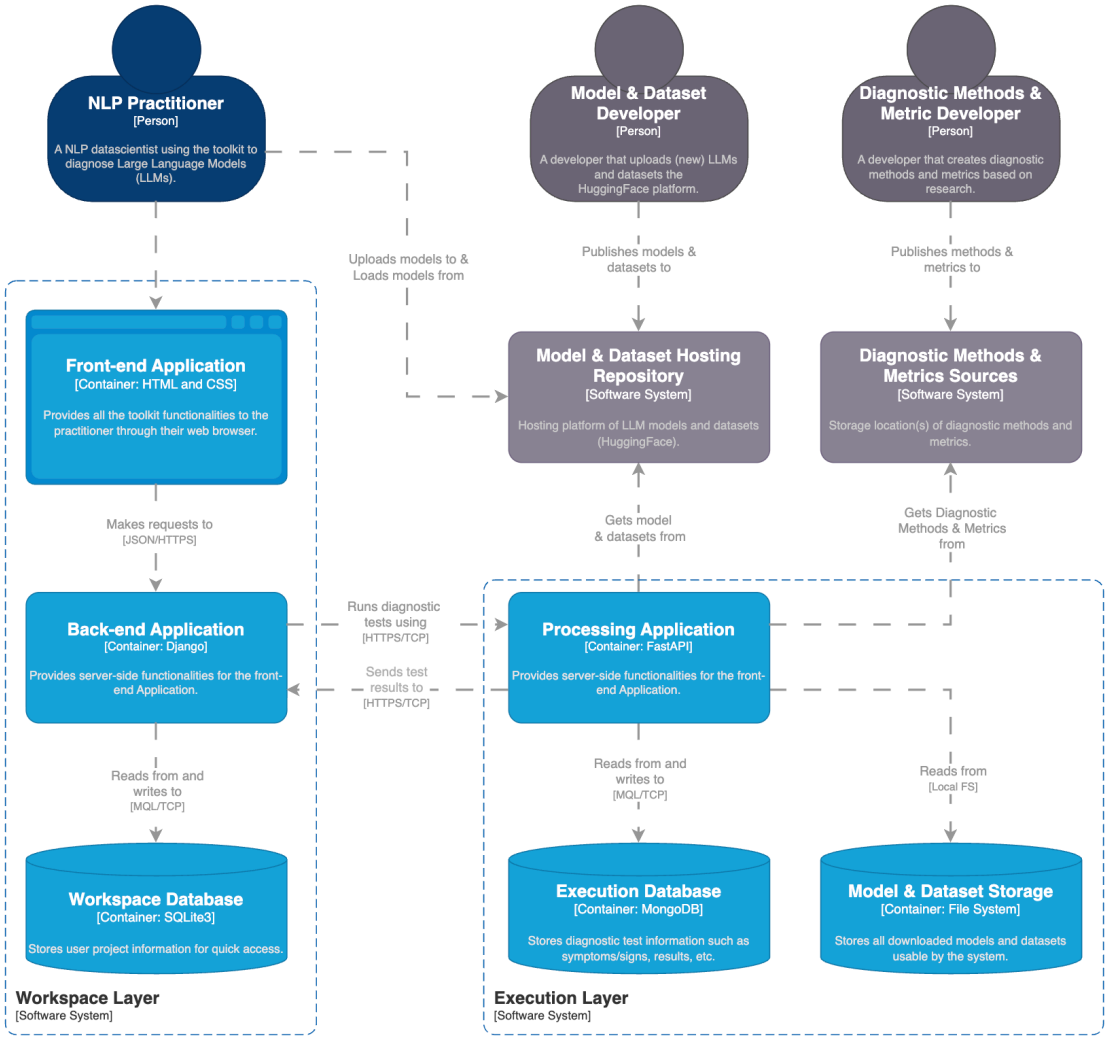

Figure 4.1: C4 container diagram of the diagnostic toolkit. The workspace layer manages projects and diagnostic artifacts, while the execution layer executes inference and diagnostic methods within resource constraints.

Execution layer (FastAPI + MongoDB). A separate FastAPI service provides an execution API that runs inference jobs and diagnostic methods. Execution bookkeeping (e.g., job queue state, run status, logs) is maintained in MongoDB to support more flexible document storage for job configuration outputs. The execution layer is containerized separately to allow independent scaling and resource allocation.

Execution constraints. The system executes one job at a time. While this limits throughput, it provides predictable resource usage and stable latency during evaluation sessions. Future work could explore concurrent execution with more sophisticated resource management.

UI prototyping. Before implementation, a front-end mock-up was created in Figma[1] to operationalize the task-level user stories into concrete interactive flows (e.g., project setup, test configuration, and result inspection). The mock-up served as a design artifact to inform navigation structure and information layout, with the aim of reducing implementation churn by identifying issues with key screens and transitions before development began. The final implementation follows the mock-up's overall navigation structure and interaction model, with deliberate deviations in specific screens where technical constraints or evaluation requirements necessitated alternative approaches. The Figma prototype is included in Appendix D.

[1] Figma is a collaborative design tool for creating user interfaces and prototypes. https://www.figma.com/

4.3. Toolkit Functionality and Workflow Support

Each feature in this section directly addresses one or more of the design goals established in Section 4.1. Project-based organization (§4.3.1) and job-based execution with traceability (§4.3.2) together operationalize the *reduce cognitive overhead* and *support non-linear workflows* goals: by preserving diagnostic state (model configurations, run histories, and outputs) across sessions, practitioners can branch into new hypotheses, revisit prior runs, and resume investigations without reconstructing context. Comparison views (§4.3.3) operationalize the *contextual interpretation* goal by enabling side-by-side inspection rather than isolated per-run analysis. The annotation and tagging system (§4.3.4) operationalizes the *support structured diagnostic reasoning* goal by providing in-tool representations for symptoms, signs, and failure patterns that accumulate across sessions rather than being discarded after each run. The reporting function in the same subsection further serves the *contextual interpretation* goal by synthesizing selected outputs, diagnostics, and annotations into a structured summary, enabling reflection across runs rather than in isolation, and supports sharing findings with others.

4.3.1. Project Workspace

Diagnostic work is organized into projects. A project encapsulates the selected model, associated datasets, metric configurations, and the set of diagnostic methods available in that context. This structure supports repeated investigation under consistent assumptions and enables swapping models or datasets without losing diagnostic history (see Figure 4.2).

4.3.2. Job Execution and Traceability

The primary unit of work in the toolkit is an *execution job*. Jobs run either (i) evaluation-style tasks (e.g., dataset-based inference) or (ii) diagnostic methods (e.g., attribution). Each job records its configuration (model, dataset, parameters), outputs, and metadata, enabling reproducibility and comparison. This job-based representation operationalizes the iterative nature of diagnosis: hypotheses are tested by running targeted jobs, comparing outcomes, and revising assumptions. Figure 4.2 illustrates an example project view with multiple jobs run under different parameter configurations for a single diagnostic method.

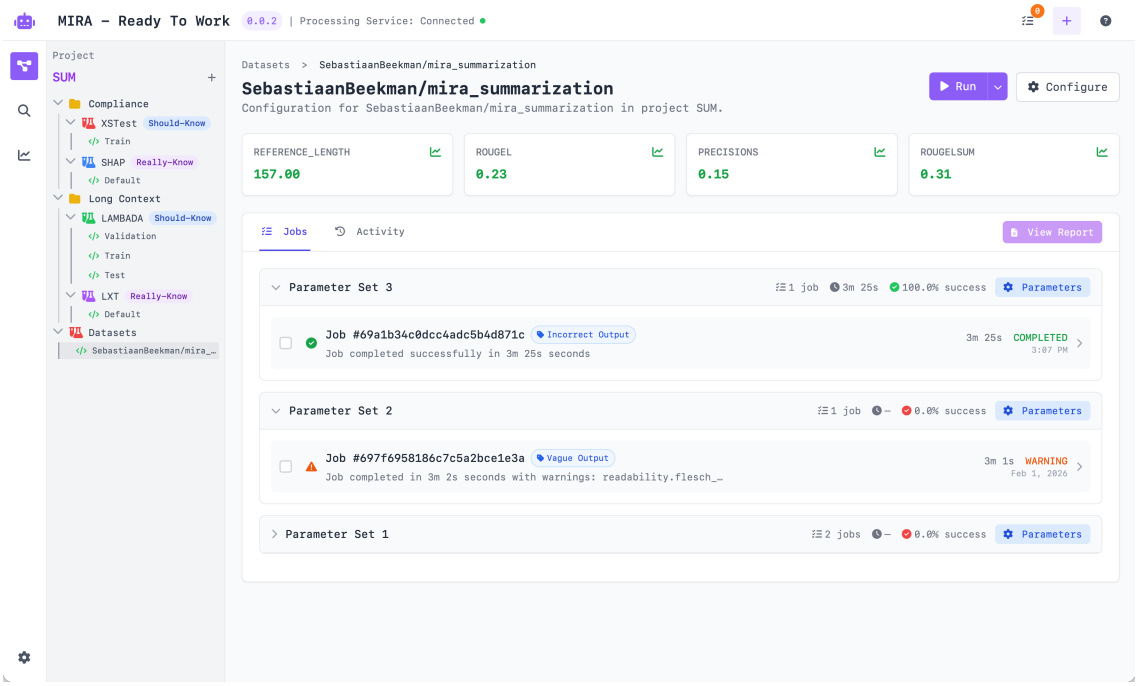

Figure 4.2: Project workspace view showing a diagnostic method with multiple jobs executed under varying parameter configurations. A symptom tag is visible alongside a job name, illustrating the annotation scheme described in Section 4.3. The view demonstrates project-based organization (§ 4.3.1), job traceability (§ 4.3.2), and inline symptom tagging (§ 4.3.4).

4.3.3. Result Exploration and Comparison

The result explorer provides access to outputs, scores, and diagnostic signals generated by jobs. Comparison views support side-by-side inspection across jobs (e.g., parameter variants, or different prompts), enabling practitioners to detect instability, recurring symptoms, or differences that support or weaken a failure pattern hypothesis. Figure 4.3 shows an example comparison, with the output view and metric view displayed side by side, as well as an example token attribution output from the AttnLRP method (RK).

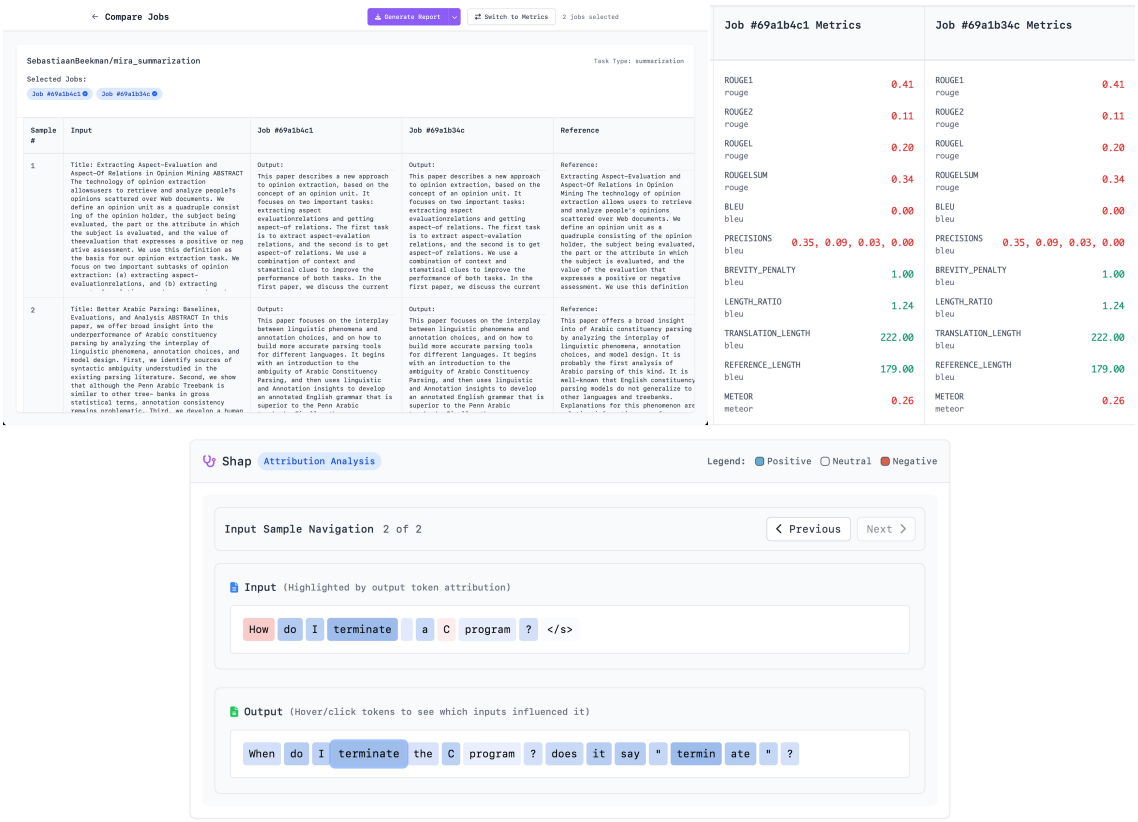

Figure 4.3: Comparison view showing two jobs side by side. The top-left panel displays the output view (model responses per input); the top-right panel displays the metric view (scores across evaluation dimensions); the bottom panel shows an example token attribution output from the AttnLRP method (RK), illustrating how attribution signals are surfaced alongside other diagnostic outputs. This collage illustrates how the toolkit supports detecting instability and differences across runs, as described in Section 4.3.

4.3.4. Annotation, Tagging, and Reporting

To support evidence accumulation, the toolkit allows practitioners to tag jobs with symptoms, signs, and failure patterns and to attach free-form notes. Tags support structured aggregation across sessions, while notes capture contextual judgments that cannot be formalized (an example tag is visible in Figure 4.2). This tagging scheme operates without enforcing deterministic mappings: practitioners may begin by tagging symptoms, add signs as instrumented evidence becomes available, and eventually attach candidate failure patterns as hypotheses stabilize. A lightweight reporting function synthesizes selected outputs and annotations into a structured diagnostic summary, supporting later reflection and analysis.

4.4. Should-Know and Really-Know Method Labeling

The toolkit operationalizes Should-Know and Really-Know as a method-level labeling scheme. **SK-based methods** support expectation articulation and validation, typically through datasets, benchmarks, or behavioral tests expressing what the model should do in a given context. **RK-based methods** support empirical probing, producing diagnostic signals about what the model relies on or how

its behavior changes under intervention. The SK/RK label should be interpreted as a method's *dominant diagnostic role*: some methods may indirectly contribute to both lenses, but labeling clarifies their primary use and supports structured comparison between expected and observed behavior.

4.5. Diagnostic Methods and Metrics

Thirteen methods were selected for implementation, chosen to provide coverage across both SK-based and RK-based diagnostic roles. The selection prioritizes behavioral benchmarks and capability tests on the SK side, and robustness tests alongside token-level attribution methods on the RK side, collectively covering a range of failure pattern hypotheses from shortcut reliance and context neglect to demographic bias and exaggerated safety behaviors.

4.5.1. Implemented Methods

The toolkit includes the following implemented methods:

LAMBADA [72]. A long-context word prediction benchmark used to probe whether models can maintain and use information across long spans. This supports SK-based expectation checks related to context utilization and can help diagnose symptoms of poor long-context performance or input neglect.

LM diagnostics (Psycholinguistic tests) [31]. A curated suite of tests targeting commonsense and pragmatic inference (CPRAG-102), event knowledge and semantic role sensitivity (ROLE-88), and negation handling (NEG-136). These tests support SK-based capability auditing and can surface systematic weaknesses aligned with reasoning-related symptoms and failure pattern hypotheses.

HANS [60]. A benchmark designed to expose reliance on shallow heuristics in Natural Language Inference (NLI) models. It supports SK-based evaluation and is particularly relevant for investigating shortcut-reliance hypotheses when poor NLI performance is observed.

SentEval [23]. A library for evaluating sentence embedding quality across probing and downstream tasks. Ten probing tasks are implemented to capture linguistic properties (e.g., sentence length, word content, tree depth) and more complex syntactic/semantic features (e.g., tense, subject/object number, semantic oddity) alongside downstream tasks (e.g., sentiment analysis, paraphrase detection). This supports SK-based evaluation of representational quality and can help diagnose symptoms of representational mismatch or insufficient linguistic encoding.

XSTest [84]. A benchmark focused on exaggerated safety behaviors, in which safe prompts are refused because they superficially resemble unsafe ones. This directly supports the exaggerated safety symptom and can contribute evidence toward instability or shortcut reliance hypotheses. SK-based evaluation on this benchmark can reveal whether the model's safety behavior is appropriately calibrated or if it exhibits signs of overgeneralization in safety-related contexts.

TREC-COVID (BeIR) [97, 101]. A dataset for biomedical information retrieval, focusing on the retrieval of relevant documents for COVID-19-related queries. This benchmark supports SK-based evaluation in the retrieval context and can help diagnose symptoms related to retrieval effectiveness in specialized domains.

SciFacts (BeIR) [108, 101]. A dataset for scientific claim verification, focusing on retrieving evidence documents to support or refute claims. This benchmark supports SK-based evaluation in the retrieval context and can help diagnose symptoms of evidence retrieval effectiveness.

SCIDOCS (BeIR) [22, 101]. A dataset for scientific document retrieval, focusing on retrieving relevant documents based on scientific queries. This benchmark supports SK-based evaluation in the retrieval context and can help diagnose symptoms related to retrieval effectiveness in specialized domains.

HolisticBias [94]. A collection of sentence templates and instances containing demographic identity language used to probe differential model behavior. Sentences like ”Hi! I am a Catholic grandmother.” are generated by varying identity terms across axes such as ability, nationality, and religion. This supports the diagnosis of harmful output symptoms and stereotyping failure pattern hypotheses through slice-based comparisons. Unlike benchmark-style SK methods, HolisticBias does not test against a fixed expected output. Rather, it surfaces differential reliance on demographic surface features across slices, making it RK-based.

Stress Test NLI [67]. Competence and distraction tests that probe robustness under controlled variations in NLI input. These tests are automatically constructed to introduce specific challenges (e.g., negation, antonymy, and numerical reasoning) and measure performance degradation. These support RK-based evaluation and can reveal signs of fragility through behavioral instability.

AttnLRP (LXT) [2]. A relevance propagation method for transformer models that provides an RK-based signal indicating how much each input token and each individual neuron contributes to the final prediction logit. AttnLRP combines attention weights with layer-wise relevance propagation to produce attribution scores through a single backward pass. This is used as a mechanistic probe to support hypotheses such as ignoring input or instruction negligence, while treating attribution outputs as diagnostic clues rather than causal proof.

SHAP [57]. A method for explaining model outputs by quantifying feature contributions. Attribution scores are derived by approximating Shapley values from cooperative game theory, which quantify each input feature’s contribution to the model’s output. For language models, features often correspond to input tokens or higher-level constructs. This supports the RK-based mechanistic interpretation and can help identify features that drive model decisions.

Integrated Gradients [98]. A gradient-based attribution method that quantifies the contribution of each input feature to the model’s output. By integrating gradients along a path from a baseline input (sensitivity axiom) to the actual input, it produces attribution scores that reflect the importance of individual features. This provides RK-based insights into model reliance on specific input components.

For all three attribution methods (AttnLRP, SHAP, and Integrated Gradients), outputs should be interpreted as diagnostic signals rather than causal explanations of model behavior. This is because attribution methods have known limitations (see Chapter 2) that can produce misleading signals if taken at face value. Instead, these methods are best used as part of a broader diagnostic process that triangulates evidence across multiple methods and considers the broader context of model behavior.

Table 4.1 summarizes the implemented methods and their primary role.

Method	Role	Typical diagnostic use (examples)
LAMBADA	SK	Probe long-context utilization; diagnose input neglect symptoms
LM Diagnostics	SK	Audit commonsense/pragmatic reasoning, event knowledge, negation handling; diagnose reasoning-related symptoms
HANS	SK	Expose heuristic reliance; diagnose shortcut-related symptoms
SentEval	SK	Evaluate sentence representation quality; diagnose representational mismatch symptoms
XSTest	SK	Probe exaggerated safety behaviors; diagnose safety-related symptoms
TREC-COVID	SK	Evaluate biomedical retrieval; diagnose retrieval effectiveness symptoms
SciFacts	SK	Evaluate scientific claim verification retrieval; diagnose evidence retrieval symptoms
SCIDOCS	SK	Evaluate scientific document retrieval; diagnose retrieval effectiveness symptoms
HolisticBias	RK	Probe demographic bias; diagnose harmful output symptoms
Stress Test NLI	RK	Measure robustness under input variations; diagnose fragility signs
AttnLRP (LXT)	RK	Attribute outputs to input tokens; diagnose input neglect hypotheses
SHAP	RK	Quantify feature contributions; diagnose input feature reliance
Integrated Gradients	RK	Attribute outputs to input features; diagnose input component reliance

Table 4.1: Implemented methods, their primary diagnostic role, and typical use within the framework.

4.5.2. Metrics

The toolkit supports 17 metrics for evaluating model performance and diagnostic signals. These include standard classification metrics (e.g., accuracy, precision, recall, F1), ranking metrics (e.g., AP, MAP, nDCG, MRR@K), and generation evaluation metrics (e.g., BLEU, ROUGE, METEOR, BERTScore). Additionally, specialized metrics such as SARI for simplification and readability metrics are included to support specific diagnostic contexts. The selection of metrics was guided by their relevance to the implemented methods and the diagnostic contexts they target. For example, retrieval tasks primarily use ranking metrics, while summarization and simplification tasks use generation evaluation metrics. Table 4.2 groups the supported metrics by task context to clarify their applicability.

Task context	Metrics (examples)
Retrieval	Accuracy, Precision, Recall, F1, AP, MAP, nDCG, MRR@K
Summarization	BLEU [73], ROUGE [56], METEOR [11], BERTScore [121], HaRiM+ [95], WeCheck [117], Readability metrics [29]
Simplification	BLEU [73], SARI [118], WeCheck [117], Readability Metrics [29], Perplexity,

Table 4.2: Metrics grouped by task context.

4.5.3. Considered, Omitted, and Failed Implementations

Not all candidate methods were implemented. Several were excluded by scope: judged valuable but not essential for the evaluation contexts targeted, or requiring infrastructure beyond the thesis constraints. A small number of attempts were made, but could not be successfully integrated due to techni-

cal constraints, primarily library incompatibilities with the containerized environment. These unsuccessful methods are listed in Appendix E with brief descriptions and reasons for omission for transparency and to inform future work.

4.6. Limitations as a Research Instrument

The toolkit operationalizes the diagnostic framework but is constrained as a research prototype. Several limitations affect the interpretation of evaluation results.

- **Not production-grade.** The toolkit prioritizes diagnostic workflow exploration over production readiness. Several features common in production systems are intentionally absent. For example, there is no support for user accounts, multi-device access control, advanced error handling, and extensive scalability features.
- **Single-user, non-collaborative.** The system is designed for local, single-user interaction rather than collaborative investigation, versioned team workflows, or shared dashboards. This reflects resource constraints and the focus on individual practitioner workflows.
- **Serial execution.** Only one job executes at a time, which limits throughput but improves stability and interpretability during evaluation sessions. Concurrent execution and resource management are left for future work.
- **Partial method coverage.** Only a subset of diagnostic methods is implemented; others were omitted due to scope or infeasibility under technical constraints. This limits the range of diagnostic signals available, but allows for a focused exploration of key methods aligned with evaluation contexts.
- **No automated diagnosis.** Failure patterns are not inferred automatically. The toolkit supports hypothesis formation and evidence organization, but does not produce definitive diagnoses. This reflects the complexity of diagnostic reasoning and the current limitations of automated methods.

Taken together, these limitations bind the scope of conclusions that can be drawn in relation to the research questions. With respect to SQ2 (what kinds of diagnostic support help practitioners identify and understand failure patterns), findings are bound to the methods and workflow features actually implemented, since a broader or different method selection may surface additional practitioner needs or expose workflow gaps not encountered here. With respect to SQ3 (how the toolkit complements or improves upon current diagnostic practices), the single-user, local context means conclusions speak to individual, exploratory diagnostic workflows rather than collaborative or production-scale settings, and the absence of automated diagnosis means findings reflect a human-in-the-loop model of diagnosis throughout. These limitations are acknowledged as part of the research design, with the aim of providing a focused evaluation of how structured support for diagnosis can enhance practitioner workflows within the constraints of a research prototype.

4.7. Summary

This chapter presented the diagnostic toolkit as a research instrument that operationalizes the framework from Chapter 3. Design goals were translated into a structured requirement hierarchy, and the system was designed using the C4 methodology to clarify architectural boundaries. The toolkit supports diagnosis through project-based organization, job execution with traceability, result comparison, and persistent annotation of symptoms, signs, and failure pattern hypotheses. The next chapter describes the practitioner evaluation used to assess how these tools support diagnostic workflows in practice.

5 Evaluation

This chapter describes the evaluation methodology used to study how practitioners diagnose undesirable behavior in large language models when supported by the proposed diagnostic toolkit. The evaluation is conducted as an observational, qualitative user study, complemented by standardized quantitative questionnaires capturing perceived workload and user experience. The focus of this chapter is on study objectives, participants, procedure, data collection, and analysis approach. Empirical findings are presented in Chapter 6.

5.1. Evaluation Objectives

The evaluation investigates the toolkit as a research artifact intended to support hypothesis-driven diagnosis. Concretely, the study aims to:

- Examine how practitioners diagnose undesirable LLM behavior using the toolkit in a believable diagnostic scenario,
- explore how the diagnostic framework (symptoms, signs, failure patterns; Should-Know/Really-Know lens) supports or impedes practitioner reasoning during diagnosis,
- and compare toolkit-supported diagnostic work to practitioners' existing diagnostic practices (as elicited through remarks and discussion with participants).

These objectives primarily inform the main research question and sub-questions SQ2 and SQ3. Challenges in diagnosis (SQ1) are treated as contextual background, surfaced organically during sessions.

5.2. Study Design

The following sections describe the study design, including methodological approach, participants, tasks, setting, and procedure.

5.2.1. Methodological Approach

The study follows a mixed-methods evaluation design. The core data originates from interactive, think-aloud, and semi-structured interviews, providing contextual insight into diagnostic workflows and reasoning. Quantitative questionnaires (NASA-TLX [40] and UEQ [55]) provide supplementary workload and user experience metrics, interpreted in light of the qualitative findings.

In terms of design science evaluation logic, the study evaluates the artifact through *observational use* by practitioners in a diagnostic scenario, as well as through reflective discussion on how the toolkit aligns with or differs from existing diagnostic practice.

5.2.2. Participants

Eight practitioners participated in the study (N=8). Sessions were conducted between October 2025 and February 2026. Participants were assigned alternatively to two diagnostic contexts (summariza-

tion and simplification), with assignment rotating between tasks (P1: summarization, P2: simplification, P3: summarization, etc.). Participants were recruited through professional contacts within the supervising team, which enabled access to practitioners with relevant expertise but limited the diversity of participants and the generalizability of the findings. Inclusion criteria required participants to have active professional or research involvement in machine learning and/or LLM development (including both industry and academic roles), with hands-on experience in model evaluation, testing, or debugging workflows.

Participant backgrounds are summarized in Table 5.1. All participants provided informed consent prior to participation.

ID	Task	Primary Role	Years Exp.	Evaluation/Debug Exp.
P1	Summarization	ML Engineer	3	Moderate
P2	Simplification	ML Researcher	7	Extensive
P3	Summarization	Data Scientist	4	Limited
P4	Simplification	ML Engineer	2	Moderate
P5	Summarization	PhD Candidate	4	Extensive
P6	Simplification	PhD Candidate	2	Limited
P7	Summarization	Applied Scientist	1	Extensive
P8	Simplification	ML Engineer	5	Moderate

Table 5.1: Interview participants' details and background. Years of Experience refers to the years they have spent in that role or have had that title. Evaluation/Debug Experience is based on self-reported background descriptions during the session. Limited experience indicates some exposure but no regular hands-on work; Moderate indicates regular hands-on work without deep specialization; and Extensive indicates deep specialization and regular hands-on work.

5.2.3. Tasks and Scenario

To ground the diagnosis in a concrete and consistent context, participants were presented with a fictional yet believable design brief describing the development of an academic search system for university students. The toolkit was designed to support three component tasks (retrieval, summarization, and simplification), but due to thesis timeline constraints, the retrieval diagnostic context was not completed in time for inclusion in the evaluation. The evaluation, therefore, focused on two diagnostic contexts. Each participant was assigned one of the following tasks, with four participants assigned to summarization and four to simplification:

- **Summarization:** diagnosing an encoder–decoder summarizer (`LongT5`[1]);
- **Simplification:** diagnosing a decoder-only instruction-tuned model (`Qwen2-1.5B-Instruct`[2]).

These models were selected based on computational feasibility (both models ran efficiently on available hardware) and diagnostic suitability (both produced reasonable but imperfect outputs, providing meaningful diagnostic opportunities without requiring best-in-class performance).

The intent of the tasks was not to "solve" a predefined bug, but to prompt participants to (i) explore model behavior, (ii) surface and externalize their diagnostic workflow, and (iii) reflect on how the toolkit supported or constrained their reasoning. Each context included a curated baseline dataset aligned with the design brief (based on scientific papers curated from the ACL ontology) to enable immediate evaluation without requiring participants to first select models and data sources. The full design briefs are provided in Appendix F.

5.2.4. Study Setting and Materials

Participants interacted with the diagnostic toolkit running on a dedicated virtual machine via a web interface. Technical bug fixes addressing implementation errors (e.g., metric calculation failures, visualization rendering issues) surfaced by earlier participants were applied in between sessions, affecting all participants approximately equally. As a research prototype rather than production software, these technical constraints were documented but did not substantially interfere with diagnostic exploration.

[1] https://huggingface.co/pszemraj/ long-t5-tglobal-base-16384-book-summary
[2] https://huggingface.co/Qwen/Instruct

No fundamental changes to diagnostic features or framework concepts were made during the study period. Sessions were conducted either in person or remotely via Microsoft Teams, depending on the participant's availability. A short demonstration video was shared with participants in advance to increase familiarity and engagement; however, watching it was optional.

No structured tutorial or training was provided prior to the session. This design choice aimed to approximate first-contact usage and to reveal how discoverable and self-exploratory the interface and workflow were for practitioners. However, this choice necessarily means the study captures both toolkit usability (learnability, discoverability) and diagnostic effectiveness, rather than isolating diagnostic workflow patterns alone.

5.2.5. Procedure

Each session lasted approximately 60 minutes and followed a semi-structured protocol:

- **Introduction and consent (approx. 3 min):** the study purpose and session structure were explained, consent was confirmed, and recording was initiated.
- **Participant background (approx. 2 min):** participants briefly described their role, experience, and current model testing/diagnosis practices.
- **Hands-on think-aloud session (approx. 30 min):** participants received the design brief and were asked to explore how they would validate and diagnose the model using the toolkit. The think-aloud protocol was selected to capture participants' real-time diagnostic reasoning and decision-making processes as they interacted with the toolkit's features, enabling the observation of both explicit verbalization and implicit workflow patterns. The facilitator prompted participants to verbalize their reasoning when silent (e.g., "What made you look at this?"), without directing them toward specific diagnostic conclusions. The 30-minute duration was chosen to keep the total session length manageable, though, as discussed in the limitations, this constraint likely biased participants toward exploratory, surface-level analysis rather than deeper diagnostic iteration.
- **Questionnaires (approx. 10 min):** participants completed NASA-TLX and the User Experience Questionnaire (UEQ).
- **Semi-structured interview and reflection (approx. 15 min):** participants reflected on helpful and unhelpful aspects of the toolkit, how the toolkit compared to their existing workflows, and what they would change or add.

The full procedure is available in Appendix G.

5.3. Data Collection

This section describes the data collected during the study, including qualitative and quantitative sources.

5.3.1. Qualitative Data

Multiple qualitative data sources were collected to capture both behavior and reasoning:

- **Audio recordings** of sessions;
- **Screen recordings** of the participant's interaction with the toolkit;
- **Automatic transcripts** (Microsoft Teams), cleaned manually using audio, screen recordings, and observational notes;
- **Observational notes** taken during the hands-on portion, focusing on moments of confusion, breakdowns, workarounds, and salient reasoning shifts.

5.3.2. Quantitative Data

Two standardized questionnaires were collected digitally:

- **NASA Task Load Index (NASA-TLX)** to capture perceived workload across six dimensions (Mental Demand, Physical Demand, Temporal Demand, Performance, Effort, Frustration).

- **User Experience Questionnaire (UEQ)** to capture perceived user experience across 26 semantic differential items, later aggregated into the standard UEQ scales (e.g., Perspicuity, Efficiency, Dependability) and benchmark.

These questionnaires were included to provide standard usability and workload metrics for the toolkit prototype, to be interpreted in light of qualitative findings (e.g., whether elevated workload scores correspond to high-friction moments observed during sessions), and to enable comparison with UEQ benchmark data. With eight participants, these data are treated as exploratory and descriptive rather than providing statistically generalizable claims.

5.4. Data Analysis

The following sections describe the analysis approach for qualitative and quantitative data.

5.4.1. Qualitative Analysis: Codebook Thematic Analysis

Interview and think-aloud data were analyzed using **codebook thematic analysis** [51, 46, 34], a systematic approach that balances structured coding with interpretative flexibility. This approach sits between reflexive thematic analysis [17, 16] and coding reliability methods [39, 15]. Codebook thematic analysis was selected because the study's research questions (SQ2: diagnostic workflow; SQ3: toolkit affordances) provide a focused analytic frame that benefits from structured analysis. Specifically, the existence of pre-defined research questions and diagnostic framework concepts introduced in earlier chapters made a hybrid approach appropriate: code families could be defined a priori based on research question alignment, while individual codes emerged iteratively through familiarization and systematic coding of transcripts. A purely inductive approach would have required setting aside these theoretically relevant constructs, risking the rediscovery of concepts already formalized in the framework, thereby reducing analytical efficiency without methodological benefit. Conversely, purely deductive coding would have constrained the identification of unanticipated patterns in practitioner behavior.

Epistemological and Ontological Positioning. The analysis is positioned within a *contextualist epistemology* [58, 41] and *critical realist ontology* [36]. From an ontological perspective, we assume that real, underlying patterns exist in how practitioners diagnose LLM failures, but that these patterns are not immediately transparent in participants' accounts. That is, practitioners may not explicitly articulate the full structure of their diagnostic workflow, the affordance dependencies they rely on, or the gating mechanisms that shape their reasoning. Instead, they require interpretation and inference to uncover these latent structures. For example, a participant may describe selecting a method based on "intuition" or "experience," but through analysis, we may infer that this selection is actually based on an implicit affordance dependency (e.g., "I know this method gives me quick feedback on output quality, which helps me decide whether to dig deeper") or a gating mechanism (e.g., "I only use this method because it doesn't require me to set up complex infrastructure, which I don't have time for").

From an epistemological perspective, we treat participants' accounts (both verbal reflections and observed behaviors) as valid knowledge within the specific context of LLM diagnosis. We do not assume that these accounts represent a complete or fully accurate depiction of their diagnostic reasoning, but we do treat them as the primary source of evidence for understanding how practitioners approach diagnosis in this domain. That is, when practitioners describe how they approach diagnosis "in a real setting" or justify their method selections, these accounts constitute valid evidence of diagnostic reasoning practices within this domain, though they may not generalize beyond the study's task contexts or participant expertise levels.

The analysis, therefore, focuses on interpreting these accounts to identify patterns and structures in diagnostic workflows and toolkit affordance dependencies, rather than treating them as direct windows into an objective diagnostic process.

Phase 1: Codebook Development and Initial Coding. The first phase of analysis focused on developing a codebook that captures patterns relevant to the research questions, with coding beginning as transcripts became available during the study period. The process involved:

Familiarization. Transcripts were read and re-read, accompanied by review of selected screen recording segments to contextualize verbal reasoning with observed interaction patterns.

Codebook Construction and Initial Coding. Three code families were defined a priori based on alignment with the research questions (SQ2 and SQ3). The author then conducted iterative coding on transcripts to develop individual codes within these families, with periodic review meetings with a supervisor to ensure coding logic was defensible and conceptually coherent. Initial codebook construction yielded 22 codes organized within the following families:s

1. **Diagnostic workflow and reasoning** (aligned with SQ2): codes capturing how practitioners establish expectations, select and interpret methods (Should-Know/Really-Know), form hypotheses, and triangulate evidence.
2. **Toolkit affordances and friction points** (aligned with SQ2 and SQ3): codes capturing discoverability, comparison use, state externalization, parameter adjustment, and constraints encountered.
3. **Relationship to current practice** (aligned with SQ3): codes capturing references to existing workflows, adoption drivers/barriers, and perceived complementarity.

Each code included a definition, inclusion/exclusion criteria, and example quotes to ensure consistent application. For instance, the code ”Inspect samples manually” was defined as ”Participant qualitatively reviews individual examples/outputs to assess quality or identify issues,” with inclusion criteria requiring explicit mentions of manual inspection (e.g., ”Let me look at this sample,” ”Input/output seems fine”) and exclusion criteria ruling out general statements about method selection without reference to manual review or proposing inspection without evidence of actual review (hypothetical statements like ”I would look at samples” were not coded as ”Inspect samples manually” unless accompanied by evidence of actual inspection).

Systematic Application. The finalized codebook was systematically applied to all eight participant transcripts, yielding 422 coded excerpts in total. Excerpts could receive up to three codes plus analytical memos.

Phase 2: Clustering and Theme Development. After coding all transcripts, the coded excerpts were analyzed to identify patterns and relationships. This involved:

Clustering and Synthesis. Coded excerpts were iteratively clustered into 28 sub-clusters representing fine-grained behavioral and reasoning patterns (e.g., baseline-first sequencing, comparison as sensemaking, method output uncertainty, tagging for longitudinal tracking). These sub-clusters were organized within broader thematic groupings aligned with the three code families, though clustering initially proceeded bottom-up from coded excerpts rather than top-down from predefined categories.

Insight and Theme Construction. Sub-clusters were reviewed for contextual overlaps and connections, leading to consolidation into higher-level insights. These insights were further abstracted into three overarching themes, refined iteratively for internal coherence (*do excerpts within a theme form a coherent pattern?*) and external distinctness (*are themes conceptually separable?*). The derivation of specific themes and insights is reported in Chapter 6 alongside supporting evidence.

Codebook Stabilization. By completion of coding and clustering, the codebook had stabilized at 34 codes across all eight participants.

Researcher Positionality. The author (a Dutch male MSc student in computer science at TU Delft) conducted this study in dual roles as toolkit designer and primary analyst. This background shaped the analysis in specific ways: training in software engineering and machine learning oriented the analysis toward technical and systems-level explanations of participant behavior, and prior investment in the diagnostic framework introduced a risk of interpreting participant actions as confirming intended use patterns. We acknowledge that this positionality may have influenced how codes were constructed, which patterns were foregrounded, and how ambiguous excerpts were resolved. In conducting the analysis, the author committed to treating participants' situated accounts as the primary source of evidence, prioritizing what practitioners actually said and did over what the toolkit was designed to support, and actively seeking instances where participant behavior contradicted or departed from the framework's

intended workflow. The reported themes represent one defensible interpretation of the data, informed by the author's disciplinary perspective and relationship to the artifact.

5.4.2. Quantitative Analysis

NASA-TLX responses were summarized descriptively (e.g., per-dimension scores and overall scale summaries). The UEQ was interpreted using its standard scale grouping, derived from the 26-item instrument, via the official analysis tool. These summaries provide descriptive reference points interpreted in light of qualitative findings (e.g., whether elevated workload or low perspicuity scores correspond to high-friction moments observed in the think-aloud data) rather than establishing independent confirmatory claims.

5.5. Ethical Considerations

The study was approved by the institutional ethics review board at TU Delft (approval code 5836). Participants provided informed consent prior to participation. Recordings captured audio and screen interaction with the toolkit. All data were anonymized, stored securely, and retained until the completion of the research project, in accordance with the consent agreement. No compensation was provided.

5.6. Summary

This chapter described the evaluation methodology used to study practitioner diagnostic workflows supported by the proposed toolkit. The study combined think-aloud protocols and semi-structured interviews, supported by workload and user experience questionnaires. The next chapter reports the empirical results of this evaluation and relates them to the research questions.

6

Results

This chapter presents the empirical findings from the evaluation study described in Chapter 5. Findings are organized into three themes derived through codebook thematic analysis (TA) alongside user experience and perceived workload results from standardized questionnaires. The chapter emphasizes descriptive reporting of patterns grounded in participant excerpts. Within each theme, recurring patterns are labeled as Insights (I1–I7), serving as precise reference points used throughout this chapter. Throughout this chapter, references to ”similar” behavior indicate conceptually equivalent statements or actions coded under the same theme during analysis, though specific phrasings and implementations varied across participants. Quote selection prioritized representative articulations that clearly illustrate each pattern, with participant coverage explicitly noted for transparency. Broader interpretation and implications are discussed in Chapter 7.

6.1. Overview of Findings

We report three themes that characterize how practitioners approach diagnosis with the toolkit, the forms of support they valued, and what gated deeper diagnostic work within the study setting. These findings are accompanied by the User Experience Questionnaire (UEQ) and NASA-TLX results that contextualize perceived usability and workload.

6.2. Theme 1: Establishing diagnostic expectations through baseline-driven calibration

All eight participants constructed operational expectations for what the model *should* do in the given task context before meaningfully assessing model behavior. This expectation was shaped by the design brief and participants' own diagnostic practice. Six participants (P1, P2, P4, P5, P7, P8) explicitly articulated acceptance criteria (e.g., faithfulness, completeness, non-hallucination) and, in some cases, considered worst-case risks or failure consequences. The other two participants (P3, P6) did not explicitly articulate acceptance criteria but demonstrated expectation-driven decision-making, for example, through their selective prioritization of metrics and diagnostic methods. P2 articulated the following criteria:

> *”I would expect several things for text simplification (of academic papers) […] I would need precision that all the facts introduced, all the things that are mentioned are actually there in the paper, preferably with some like citations or something. Second thing I would need is recall for (ensuring) that everything that should be mentioned is mentioned. So no crucial information is omitted from the original text.”*

All participants used initial evaluations as an expectation-calibration step: by running an initial 'baseline' and inspecting early model outputs, metric scores, and diagnostic outputs, they grounded (and sometimes revised) their expectations for the model in practice. Rather than committing to a single metric or diagnostic method, participants reduced uncertainty by triangulating across complementary

signals. Comparisons between runs further served as a recurring sense-making operation for interpreting whether observed changes were meaningful.

6.2.1. I1: Baseline-first as the default entry point

Seven participants (P1, P2, P3, P4, P5, P7, P8) treated a baseline run as the first empirical step after establishing task expectations, using it as a low-cost preliminary validation before investing effort into tuning parameters or applying deeper diagnostic methods. While one participant (P7) adjusted the system prompt during setup, they still followed the baseline-first principle by using these initial runs to calibrate expectations rather than committing to specific hypotheses. This baseline-first orientation helped verify that the overall setup behaved plausibly and surfaced symptoms worth investigating further. P1 explicitly stated:

> *"I suppose I don't need to change this [model parameters] for now […] I would probably reduce temperature […] but I'm going to leave that for now. I haven't worked on this particular model (before) […] I don't want to change anything for now. I first want to see if I can get some baseline results."*

Some participants (P1, P2, P5) combined this baseline-first approach with systematic data quality checks, inspecting for encoding issues, parsing artifacts, out-of-distribution tokens, and formatting problems. P5 explicitly described this data-first diagnostic entry:

> *"That [adversarial tokens] is something you would see from here [job comparison] now. You would look (to see) is there anything out of distribution [in the data] because you never know with online text if there is some weird stuff (in there) like emojis or something that might ruin everything. So that is the first thing I would check."*

Five participants (P1, P2, P4, P5, P8) also explicitly described progressively scaling the baseline evaluation by starting with a small subset to validate the pipeline, then expanding to the full dataset once outputs looked plausible. P2 articulated this staged workflow:

> *"I would first debug the way my setup is working. That is why we are running it on 10 samples, to check that everything is working the way I expect. For example, then I change the prompting (or) do some modifications on top of the dataset […] After that I run it on the entire test set to get the metrics."*

Variation within this pattern: P6 appeared uncertain about diagnostic direction through the session, exhibiting less systematic baseline-first reasoning compared to other participants. Their uncertainty suggests that the baseline-first pattern may be expertise-dependent or influenced by familiarity with the task domain.

6.2.2. I2: Diagnostic confidence is built through triangulation, not single observations

All eight participants avoided reaching diagnostic conclusions based on single observations. Participants rarely treated any single metric, sample, or visualization as sufficient evidence for judging model behavior. Instead, they built diagnostic confidence by triangulating across complementary signals: inspecting concrete outputs, checking whether metrics align with what they observe qualitatively, and selectively applying diagnostic methods when uncertainty remained. P3 explicitly verbalized this triangulation:

> *"SHAP, the explainability part, helped me get the confirmation that the model was not really working well. It is a matter of observing the data, observing how the model behaved when (it) performed the task […] those were the pieces that got me to the fact that the model is not working properly."*

While P1, P2, P3, P5, and P8 explicitly verbalized this triangulation pattern, all participants behaviorally demonstrated it through their actions: combining qualitative sample inspection, aggregate metrics, and selective diagnostic methods before drawing conclusions.

This triangulation behavior was evident when participants encountered ambiguous quality signals (e.g., acceptable aggregate scores but suspicious individual outputs), prompting them to cross-check sam-

ples and compare multiple runs before reaching a conclusion. For instance, P4 inspected a specific sample, noting: *"I was curious to why this score of factual consistency (is) so low for this [sample]"*. P2 and P5 similarly flagged encoding/parsing issues, and P3 flagged reference/output mismatches during manual inspection.

6.2.3. I3: Comparison is a central sense-making primitive

Comparison repeatedly served as a mechanism for turning isolated observations into diagnostic meaning. Six of eight participants (P1, P2, P5, P6, P7, P8) explicitly described seeking contrast between runs (e.g., prompt variants, parameter changes, or model versions) to determine whether changes materially affected outputs and to localize what might be driving (undesirable) behavior. P8 articulated this iterative pattern:

> *"A lot of times it is me changing something and running it again […] certain tweaks can change a lot. So I want to see how changing the system prompt would change the outputs. I would use comparison to see if I can create another job with a better prompt and see whether this job is better than the other [original] job […] to see the implications of those [changes]."*

Rather than treating results as static snapshots, participants used comparisons to establish directional judgments (e.g., better/worse, more/less faithful, more/less compliant), which supported sense-making even under time constraints. For example, P2 expressed enthusiasm for the comparison view: *"From what I can see in the report, that [config diff] is a perfectly valid thing to want because I will be able to see two things side by side. Should be amazing"*. While valuing existing comparison features, six participants also suggested enhancements to comparison capabilities (e.g., distribution overlays, prompt/config diffs) to better support this reasoning pattern.

6.3. Theme 2: Practitioners frame the toolkit as a scaffold for low-overhead execution, comparison, and state externalization

Seven of eight participants (P1, P3, P4, P5, P6, P7, P8) consistently framed the toolkit as a scaffold for diagnostic work rather than as a replacement for their existing workflows. Three forms of scaffolding were separately emphasized:

1. Low-overhead setup and execution that supports quick baselines and exploratory probing,
2. comparative sense-making via run- and sample-level comparison helps turn observations into diagnostic meaning, and
3. externalized diagnostic state (tags, notes, history, and exports) that preserves context across iterations and supports stakeholder communication.

Together, these forms of scaffolding positioned the toolkit as a scaffold (an organizing layer) around diagnostic work: participants could quickly execute baseline probes, compare outcomes across runs, and preserve intermediate conclusions for later reflection or communication. Rather than replacing existing practice, participants framed the toolkit as an organizational upgrade that reduced coordination overhead in their current notebook-based or spreadsheet-driven workflows while maintaining compatibility with external analysis and LLM-judging workflows. P8 framed this complementarity:

> *"In my current workflow I might end up creating a bunch of (Google) Colab notebooks […] it is a lot of mental effort on my end to keep everything organized and this [the toolkit] was good in organizing, keeping everything together."*

6.3.1. I4: Comparison as a sense-making scaffold in-tool

When comparing results was supported directly in the interface (e.g., output-, metric-, or configuration comparison), participants framed it as a practical scaffold for diagnosis: it reduced the overhead of manually tracking configurations and enabled quicker judgment of whether changes improve or degrade behavior. All eight participants explicitly commented on the value of in-tool comparison features, with six participants (P1, P2, P4, P5, P7, P8) providing explicit verbal confirmation of its value for sense-making,

and several suggesting enhancements (e.g., distribution overlays, prompt/config diffs) to support this reasoning pattern better. P5 articulated this value:

> *"This [comparison] is very nice. I can see the input and output very clearly. I see that there's also a performance scale […] That's very nice."*

6.3.2. I5: Externalizing diagnostic state enables continuity and communication

All eight participants valued features that externalized diagnostic state, such as run tagging, notes, parameter histories, and the reporting/exporting of results. These mechanisms preserve context across iterations and make intermediate conclusions communicable. Rather than keeping hypotheses and observations only *"in mind"* (P8), participants described wanting persistent artifacts that record what was tried, what was observed, and why a given run mattered. P1 articulated this explicitly:

> *"This [tagging] is useful. For this run, for example, if in this one I saw it was ignoring input, then I can add it and later look back […] track failures over time."*

All participants expressed enthusiasm for tagging as a memory mechanism, with six participants (P2, P3, P4, P5, P7, P8) providing explicit verbal confirmation. However, participants assumed sample-level tagging would be available and expressed surprise upon learning the current implementation was job-level only; they indicated that sample-level tagging with (open-form) annotations was desired for fine-grained failure tracking and pattern documentation.

Exportable reports and raw results (e.g., CSV/JSON) were framed as bridges to existing workflows and stakeholder communication by six participants (P1, P3, P4, P6, P7, P8), enabling practitioners to bring findings into meetings or downstream analysis. P1 described sharing reports with different audiences:

> *"Depends on how the project is shaped. Usually, I work together with other engineers. So we do share these things [reports] with each other. With a more general public, I distill it into core changes."*

P4 emphasized reporting for company stakeholders:

> *"The report is really nice. I think it would help a lot of people in companies because I am always asked to produce documents, reports, and graphs."*

P8 similarly framed reports as communication artifacts for both iteration and stakeholder alignment:

> *"I can see myself using this (reports) as a way to present to team members or a PI. Show them what metrics are changing and how. It is an easy way of doing that and a quick way to iterate."*

P2, P6, P7, and P8 similarly valued export and reporting functionality.

6.4. Theme 3: Gating factors for diagnostic depth and effectiveness

All eight participants encountered constraints that limited diagnostic depth and effectiveness during sessions. We interpret these as operating through two interconnected factors:

1. **Interpretation friction** with unfamiliar methods and metrics, and
2. **missing workflow glue**, such as data manipulation, richer comparisons, and test branching, needed for hypothesis refinement.

We interpret these factors as operating in combination rather than independently. For instance, navigating unfamiliar interface structures while simultaneously interpreting unfamiliar metric outputs compounded cognitive demands, substantially slowing diagnostic progress. These gating factors should be interpreted as study conditions that shaped what participants could do within the session timebox and prototype constraints, rather than as shortcomings of practitioner capability.

6.4.1. I6: Interpretation friction creates systematic barriers

All eight participants encountered at least one unfamiliar method or metric during their session, which is expected given the breadth of available options. What distinguished six participants (P1, P3, P4, P5, P6, P7) was that unfamiliarity required active external assistance (e.g., consulting LLMs, looking

up definitions, or asking questions) to continue. The remaining two participants resolved similar gaps independently without interrupting their diagnostic flow. The challenge for the six was not simply definitional: knowing what a method measures did not resolve the question of how to act on its output diagnostically. This created a dual cognitive load, with learning demands running in parallel to diagnostic work. P1 explicitly acknowledged this challenge:

> *"What I would do is put this into an LLM and then ask to interpret all of these metrics to some degree because I don't know all of these metrics […] I wouldn't know how to spot good improvements."*

P5 similarly expressed unfamiliarity during metric selection: *"I'm checking what half of them [metrics] are. I know BERTScore, ROUGE, Meteor, Bleu, and that's it. So, the question is, what are they, right?"*. This pattern of needing external resources to interpret method outputs was observed across six participants (P1, P3, P4, P5, P6, P7) who encountered unfamiliar metrics or interpretive visualizations.

6.4.2. I7: Diagnostic depth is constrained by workflow glue and mental-model friction

Within the study setting, diagnostic work frequently remained at the early-stage probing stage. Although the two constraining factors described below differ in nature, they are grouped here because they consistently co-occurred and compounded one another in practice, collectively limiting how far participants could progress in the diagnostic process. All eight participants exhibited constraints from at least one of these factors, with most encountering both simultaneously:

Missing workflow glue for efficient iteration (e.g., richer inspections, data slicing, or streamlined multi-run exploration) prevented deeper hypothesis refinement. P2 requested:

> *"I want to rerun it with some processed data. I can reupload the data somehow, but it is not immediately clear how to do it. […] I can go to some other place and create a new dataset/version and pass it (to the toolkit) as the updated version."*

P4 asked:

> *"I cannot select the sample that I want to run, right? I can just run everything again. If this sample […] was particularly bad, I change the prompt and I want to run the job only for this sample, can I do it?"*

Onboarding and mental-model friction consumed session time, creating barriers to deeper diagnostic work. P1 stated: *"It's not immediately (clear). I'm not familiar with the idea of should-know and really-know tests"*. P6 expressed strong confusion:

> *"I do not really know why I run this should-know really-know thing […] I am a bit confused. […] I don't know what they have to do with seeing whether the model is good for simplification. They [SK/RK selection] popped up at the (setup). I just clicked random stuff."*

All eight participants exhibited at least one form of mental-model friction during sessions, with representative examples from P1 and P6 shown above.

Importantly, this gating reflects conditions of the evaluation environment (timebox, prototype maturity, and available workflow affordances) rather than the participant's inability to perform diagnosis. P1 explicitly verbalized that deeper investigation was feasible but time-constrained:

> *"I like that you can change or select all of these metrics, even though some might not be immediately relevant. I am not that familiar with summarization, but if I had more time."*

While all other participants (P2–P8) did not progress to deeper diagnostic stages, they did not explicitly verbalize whether this was due to time constraints or other factors. The limited diagnostic depth observed across sessions reflects the compressed timebox and prototype constraints documented in Chapter 5, though individual participants may have encountered different specific barriers. Execution constraints stemming from prototype maturity (e.g., single-job execution, platform reliability) were also present across sessions. As these reflect deployment-context conditions rather than recurring diagnostic patterns, they are addressed in the limitations (Section 7.4).

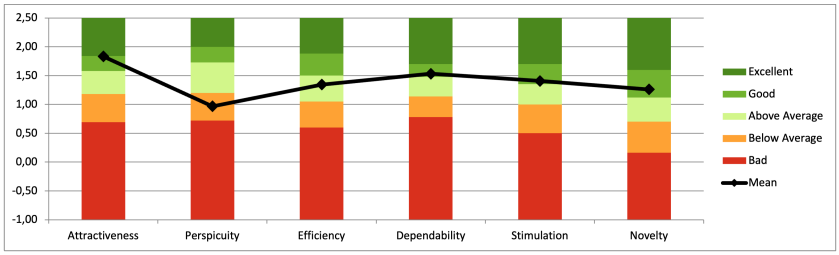

Figure 6.1: UEQ results for the diagnostic toolkit across six dimensions (N=8). Overall attractiveness was rated as Good according to the UEQ benchmark, with hedonic quality (Novelty and Stimulation) rated higher than pragmatic quality (Perspicuity and Efficiency). Perspicuity scored below average, suggesting potential issues with clarity and ease of learning, while Efficiency and Dependability were above average.

Variation within this pattern: The specific constraining factors varied by participant expertise and task context. More experienced participants (see Table 5.1) tended to encounter workflow glue limitations more frequently than mental-model friction, while participants with less LLM-specific experience encountered interpretation barriers more systematically.

6.5. User experience and perceived workload

The user experience and perceived workload of the diagnostic toolkit were assessed using the User Experience Questionnaire (UEQ) and the NASA Task Load Index (NASA-TLX). The UEQ captures multiple dimensions of user experience, while the NASA-TLX assesses perceived workload across six dimensions. Given the small sample size (N=8), these results are exploratory and indicative rather than confirmatory, providing descriptive reference points interpreted in light of the qualitative findings reported above. The following sections summarize the results from both instruments and connect them to observed qualitative patterns.

6.5.1. User experience questionnaire

The User Experience Questionnaire (UEQ) results, as shown in Figure 6.1, suggest a generally positive experience with the diagnostic toolkit (Attractiveness $M = 1.83$, $var = 0.54$ on the UEQ scale from -3 to $+3$). While the small sample size (N=8) precludes statistical testing of differences from benchmark populations, the UEQ benchmark tool provides a descriptive frame of reference for interpreting observed patterns. Using this frame, Attractiveness is positioned in the ”Good” benchmark category. Hedonic quality was rated slightly higher than pragmatic quality (Hedonic $M = 1.33$, Pragmatic $M = 1.28$), suggesting that participants experienced the toolkit as interesting and innovative. Dependability ($M = 1.53$, $SD = 0.58$), Stimulation ($M = 1.41$, $SD = 0.93$), and Novelty ($M = 1.26$, $SD = 0.25$) positioned in the ”Good” benchmark range, indicating that participants perceived the toolkit as reliable, engaging, and innovative. Efficiency ($M = 1.34$, $SD = 0.70$) positioned above the benchmark average, suggesting that participants found the interaction reasonably productive once they understood the system. However, Perspicuity scored below the benchmark average ($M = 0.97$, $SD = 2.06$), pointing to potential issues with clarity and ease of learning.

Integration with qualitative findings: The low Perspicuity scores align directly with Theme 3 (I6, I7), where all eight participants encountered mental-model friction and interpretation barriers. Participants' explicit statements about SK/RK confusion (e.g., P6: ”I do not really know why I run this should-know really-know thing”) provide qualitative grounding for the quantitatively observed clarity challenges. Conversely, the positive Stimulation and Novelty scores align with Theme 2, where seven participants framed the toolkit as providing novel organizational scaffolding compared to their existing notebook-based workflows.

As shown in Figure 6.2, there was notable variability in ratings for all dimensions except Attractiveness and Novelty, which had relatively small error bars indicating more consistent perceptions of overall appeal and innovation. Perspicuity and Dependability exhibited the largest error bars, suggesting differences among participants in their perceptions of ease of understanding and system reliability. This variability may reflect individual differences in prior experience with similar tools, diagnostic tasks, or the specific challenges encountered during the sessions.

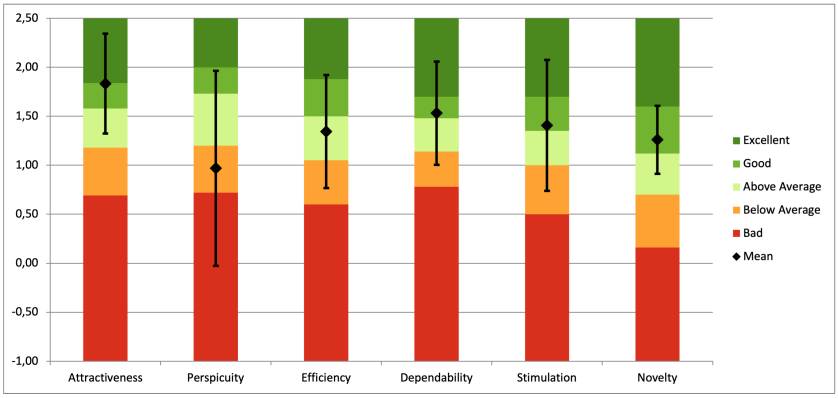

Figure 6.2: UEQ benchmark error bar comparison for the diagnostic toolkit across six dimensions (N=8). Notable variability was observed across all dimensions, except Attractiveness and Novelty, which had relatively small error bars. Perspicuity and Dependability showed the largest error bars, indicating differences among participants in their perceptions of ease of understanding and system reliability. This variability may reflect individual differences in prior experience, diagnostic tasks, or specific challenges encountered during sessions.

UEQ Dimension	Mean (M)	Cronbach's Alpha (α)
Attractiveness	1.83	.90
Perspicuity	0.97	.94
Efficiency	1.34	.64
Dependability	1.53	.62
Stimulation	1.41	0.72
Novelty	1.26	-0.34

Table 6.1: UEQ scale means and Cronbach's alpha for the diagnostic toolkit (N=8). Internal consistency was excellent for Attractiveness and Perspicuity, and acceptable for Stimulation. Efficiency and Dependability fell below the conventional .70 threshold, while Novelty showed negative alpha, suggesting inconsistent item behavior in this small sample. Results for scales with alpha below .70 (Efficiency, Dependability, and Novelty) are given less analytical weight and should be interpreted with additional caution.

Cronbach's alpha was used as an indicator of internal consistency for the UEQ scales. Given the modest sample size (N=8), these estimates should be interpreted cautiously and primarily as a preliminary check. Attractiveness (α=.90) and Perspicuity (α=.94) showed excellent internal consistency, while Stimulation showed acceptable consistency (α=.72). Efficiency (α=.64) and Dependability (α=.62) fell below the conventional .70 threshold, suggesting questionable reliability. Novelty showed negative internal consistency (α=-0.34), suggesting inconsistent item behavior (and potentially issues related to item polarity or interpretation of adjective pairs). Therefore, results for scales with alpha below .70 (Efficiency, Dependability, and Novelty) should be interpreted with additional caution and are given less analytical weight in the analysis below.

Inspection of the inter-item correlations indicated that Novelty items showed contradictory behavior, with negative inter-item correlations driving the negative alpha. This appeared to be driven by inconsistent response patterns across participants (e.g., diverging ratings on conceptually related novelty items such as 'creative/dull' and 'usual/leading edge'). While Stimulation achieved acceptable reliability (α=.72), Efficiency and Dependability approached conventional thresholds.

6.5.2. NASA task load index

Raw NASA Task Load Index (NASA-TLX) ratings suggest an overall low-to-moderate workload (aggregate score: $M = 29.9$, computed by averaging the six dimensions with Performance inverted). Workload was driven primarily by Mental Demand ($M = 43.75$), Temporal Demand ($M = 40.625$), and Perceived Performance ($M = 51.875$), consistent with the cognitive-intensive nature of exploring and applying diagnostic capabilities. Physical Demand was negligible ($M = 6.875$), Effort was low ($M = 11.875$), and Frustration remained relatively low ($M = 24.375$), suggesting that while the task

NASA-TLX Dimension	Mean (M)
Mental Demand	43.75
Physical Demand	6.875
Temporal Demand	40.625
Performance (inverted)	51.875
Effort	11.875
Frustration	24.375
Overall Workload	29.896

Table 6.2: NASA-TLX dimension means for the diagnostic toolkit (N=8). Overall workload was low-to-moderate, driven primarily by Mental and Temporal Demand, consistent with the cognitive-intensive nature of exploring and applying diagnostic capabilities. Physical Demand was negligible, and Frustration remained relatively low, suggesting the task was intellectually demanding but not emotionally taxing.

was intellectually demanding and time-pressured, it was not experienced as physically effortful or emotionally taxing.

Integration with qualitative findings: The elevated Mental Demand and Temporal demand scores align with Theme 3's identification of compounded cognitive load (I6, I7). Six participants explicitly described needing to learn metric interpretations while simultaneously navigating the interface, creating dual cognitive load (as described earlier in Theme 3). The 30-minute session constraint (Chapter 5) likely contributed to temporal Demand, but only P1 and P8 explicitly verbalized time pressure as a limiting factor for deeper investigation. The relatively low frustration scores ($M = 24.375$) despite high mental Demand suggest that participants perceived the cognitive challenges as inherent to diagnostic complexity rather than poor tool design, consistent with Theme 2's framing of the toolkit as proper organizational scaffolding.

The small sample size (N=8) and individual differences in experience (see Table 5.1) limit generalizability. These findings are best interpreted in light of the qualitative themes rather than as independent confirmatory evidence. Future iterations should consider cognitive load reduction strategies, particularly around method interpretation and onboarding, while preserving the toolkit's organizational and comparative affordances that participants valued. This is discussed further in Chapter 7.

6.6. Summary

This chapter reported three themes describing baseline-first diagnostic reasoning, toolkit scaffolding through comparison and externalized state, and gating factors that limited diagnostic depth in this study setting. User experience results indicated a positive overall experience with the toolkit, particularly in hedonic dimensions, while perceived workload was low-to-moderate and driven by mental and temporal demands. Chapter 7 discusses implications for the research questions, the diagnostic framework, and future tooling.

7

Discussion

The results in Chapter 6 were reported descriptively as themes grounded in participant excerpts. This chapter interprets those findings at a higher level of abstraction: situating them within the research questions and the diagnostic framework, and translating them into design implications for diagnostic tooling. Rather than treating themes as discrete findings, we use them as empirical indicators for broader claims about how practitioners establish diagnostic expectations, gather and interpret evidence, and what conditions gate deeper diagnosis (i.e., progression along the investigative cycle from baseline calibration toward failure pattern identification) in practice.

7.1. Synthesis and Interpretation

The three themes reported in Chapter 6 collectively answer the sub-questions as follows. SQ1 is answered by identifying that the primary challenges shaping LLM diagnosis are not method availability but interpretation barriers, workflow gaps, and execution constraints (Theme 3). SQ2 is answered by showing that the most effective diagnostic support scaffolds the investigative cycle (the iterative diagnostic sequence of expectation formation, baseline calibration, evidence triangulation, run comparison, and state externalization) rather than providing more analytically sophisticated individual methods (Theme 1). SQ3 is answered by showing that the toolkit complemented existing practice by reducing setup overhead and externalizing diagnostic state, while also revealing that integration alone is insufficient without interpretive guidance and workflow glue (Theme 2). The remainder of this section interprets what these answers jointly imply for diagnostic practice and tooling design.

7.1.1. Access to methods does not resolve the interpretation problem

Much of the XAI literature is motivated, at least implicitly, by the premise that providing practitioners with more accessible analytical methods improves their ability to investigate model behavior. This rationale is reflected in the continued development and publication of new interpretability and explainability techniques, and the present findings qualify it in a specific way. The toolkit provided embedded metrics and interpretability methods; participants used them but systematically encountered difficulty understanding what their outputs indicated and how to act on them within a specific diagnostic task. As described in Theme 3, six of eight participants required external resources, such as consulting language models, looking up definitions, or asking the facilitator to continue, after encountering unfamiliar method outputs (I6). Crucially, resolving definitional uncertainty did not, by itself, resolve the deeper question of how to act on an output in the context of a diagnostic task. Knowing what a metric measures does not indicate what an anomalous score implies about model behavior, or what the appropriate next step is. This is consistent with prior observations that XAI methods can be informative in isolation while remaining difficult to apply in context (see Chapter 2): attention-based explanations do not reliably indicate causal importance, and different XAI methods can produce divergent or contradictory insights.

The quantitative data reinforce this interpretation. UEQ Perspicuity scores below the benchmark average ($M = 0.97$) and elevated NASA-TLX Mental Demand ($M = 43.75$) are consistent with the dual cognitive load of learning a method while simultaneously applying it diagnostically. The implication

is that method access is a necessary but not sufficient condition for diagnostic progress. What mediates the gap between access and productive use is interpretive guidance that is contextual, scoped to the diagnostic question being addressed, and integrated at the point of decision rather than available elsewhere in the interface. This finding directly motivates the first design implication discussed in Section 7.3.

7.1.2. Diagnostic value is located in the investigative cycle, not individual methods

The findings from Theme 1 and Theme 2 converge on a point not straightforwardly predicted by method-centric accounts of XAI: the features participants found most practically valuable were not sophisticated interpretability methods but the workflow structures surrounding them. Baseline-first evaluation (I1), triangulation across complementary signal types (I2), run comparison as a sense-making operation (I3), and state externalization for continuity and communication (I5) constituted the practical core of diagnostic work across all eight sessions. These are properties of the investigative cycle rather than properties of any single method or explanation technique. This extends prior practitioner studies identifying XAI tools as underutilized in everyday diagnostic workflows [27, 9]. Those accounts locate the barrier primarily in adoption friction; the present findings suggest that even when methods are structurally accessible and embedded in the interface, the workflow glue that connects diagnostic steps is at least as consequential for progress as the methods themselves. The toolkit's framing as a complement rather than a replacement for existing practice (Theme 2) reflects this dynamic. Practitioners did not displace their existing workflows but embedded the toolkit as an organizational layer for activities (rapid baseline evaluation, structured run comparison, shareable reporting) that their existing tools supported only in fragmented or manually intensive ways. Crucially, this diagnostic reasoning could, under conditions of sufficient time and scaffolding, yield tentative failure pattern hypotheses. P3, for instance, combined LM Diagnostics results, SHAP attributions, and sample-level inspection to conclude that the model *"is not really good at understanding semantics […] seems like quite the random guesser."*; an early hypothesis about a systematic semantic weakness rather than a one-off output error. Taken together, these findings suggest that diagnostic tooling design should be oriented around supporting the full investigative cycle rather than expanding the analytical method inventory, a position developed in Section 7.3.

7.1.3. A structural gap separates diagnostic intent from operational execution

A cross-cutting pattern in the study is the consistent divergence between the diagnostic workflows participants described as their intended approach and what they were able to execute within the sessions. All eight participants articulated multi-step iterative workflows involving repeated probing, prompt revision, data slicing, and comparative evaluation across runs. None completed a full iteration loop within a session. This was not attributable to unfamiliarity with diagnostic reasoning, as participants' accounts of their intended workflows were coherent and aligned with the iterative, hypothesis-driven process characterized by the diagnostic framework. Instead, the constraint was operational as the workflow glue (the features and affordances that connect iterative diagnostic steps) required to enact those workflows was either absent from the prototype or disrupted by execution constraints (I7, Theme 3). This distinction matters for interpreting the findings. One might attribute the limited diagnostic depth observed to participants' unfamiliarity with the toolkit or its methods. However, participants clearly described what deeper diagnostic work would require and articulated coherent multi-step workflows. The constraint was not a lack of diagnostic understanding, but the absence of the workflow affordances needed to execute those steps. The present findings, therefore, suggest that method access is not the primary constraint in isolation. What determines whether iterative diagnosis is achievable in practice is a layer of workflow glue that connects diagnostic steps, distinct from both method availability and practitioner capability. This finding identifies the most consequential gap for future toolkit development, as discussed in Section 7.3.

7.1.4. RQ Synthesis: What kinds of tooling support practitioners in diagnosing undesirable behavior in LLMs?

The three interpretive claims above converge on a single answer to the main research question: effective diagnostic tooling is best understood as scaffolding for the investigative cycle rather than as a

collection of analytical methods. This entails supporting expectation formation and baseline calibration, enabling low-friction run comparison, facilitating triangulation across evidence types, and externalizing diagnostic state for continuity and communication while ensuring that methods are accompanied by contextual interpretive guidance and that workflow infrastructure for iterative hypothesis testing is in place. The gating factors identified in Theme 3 (interpretation barriers, missing workflow glue, and execution constraints) account for why method availability alone does not translate into diagnostic depth. Section 7.2 contextualizes the scope of this empirical validation with respect to the diagnostic framework, and Section 7.3 develops three concrete design implications for future tooling.

7.2. Empirical Validation and Scope of the Diagnostic Framework

This study constitutes a first-pass evaluation of the diagnostic framework rather than a systematic stress-test of its boundaries. Participants' behavior aligned with the framework's early-phase characterization: all eight used baseline evaluation as their entry point in a manner consistent with the Should-Know/Really-Know distinction, and all eight behaviorally combined qualitative sample inspection, aggregate metrics, and at least one diagnostic method before drawing conclusions, which is consistent with the framework's evidence triangulation pattern. These observations provide initial empirical support for the framework's early-phase constructs. However, the study was not designed to evaluate the full iterative cycle described by the framework, and the constraints documented in Section 7.4 prevented participants from progressing to later-stage hypothesis refinement. Stronger claims about the framework's later phases, therefore, require the longitudinal conditions described in Section 7.5.

The findings suggest that practitioners engage with diagnosis across a spectrum of investigative depth. Within this study, diagnosis ranged from early-stage validation and acceptance-criteria checking to failure pattern hypothesis formation (i.e., an unconfirmed suspicion about a recurring failure pattern), but rarely extended to the sustained, multi-cycle hypothesis refinement described in prior diagnostic frameworks such as Errudite [116]. The diagnostic framework proposed in this thesis can accommodate this spectrum, but failure pattern identification represents an endpoint of iterative investigation that may require multiple sessions, domain familiarity, and task-appropriate scaffolding to reach. The framework's primary contribution is therefore best understood as a conceptual model that structures the relationships among expectations, evidence, symptoms, and failure patterns at varying depths of diagnostic inquiry.

The evaluation also points to a concrete usability gap at the boundary between the framework's vocabulary and the toolkit's state externalization features. Participants expected to annotate individual samples with framework-relevant labels to associate observed symptoms or signs directly with specific outputs rather than at the job level. The current implementation supported only job-level tagging with categorical values, which participants found insufficient for fine-grained failure tracking: several indicated that open-form annotations alongside categorical tags would be needed to capture the nuances of individual output failures (I5). This suggests that the framework's concepts require richer operational support at the sample level, both in terms of granularity (sample- rather than job-level association) and expressiveness (open-form in addition to categorical labeling). This is discussed as a framework-facing future direction in Section 7.5.

7.3. Implications for Toolkit Design

The findings translate into three concrete design implications corresponding to the three gating factors:

(1) Embedded guidance for interpretation. Interpretability methods create systematic interpretation barriers. All eight participants encountered comprehension failures not only in locating features, but in understanding what a method's output means, when it applies, and how to avoid misreading it. While some participants also looked up standard metric definitions during sessions, the primary barrier concerned these interpretability methods (particularly the SK/RK diagnostic tests), where participants struggled to understand the diagnostic purpose and how to act on outputs. These interpretation barriers manifested quantitatively as well: UEQ Perspicuity scores positioned below the benchmark average ($M = 0.97$) indicate clarity and learnability challenges, while elevated NASA-TLX Mental Demand ($M = 43.75$) and Temporal Demand ($M = 40.625$) scores align with the compound cognitive load participants experienced when navigating unfamiliar methods under time pressure. This motivates the

inclusion of optional in-tool explanations, interpretation checklists, and guardrails (e.g., warnings about method limitations, reminders of assumptions, and suggested next actions). However, the presence of embedded guidance that was underutilized or insufficient suggests that guidance must not only be available but also contextually integrated, visually prominent, and aligned with the specific diagnostic question at hand; a design challenge that extends beyond simply providing documentation. All eight participants expressed confusion about Should-Know/Really-Know tests: from the tool alone, it was unclear what their diagnostic purpose was, when to apply them, and how they related to the primary evaluation task. In several sessions, the facilitator provided clarification to proceed.

(2) Workflow glue for iterative refinement. Prior work [9, 27, 116] characterizes diagnosis as requiring structure, guidance, and contextual reasoning to connect observations to potential underlying causes. All eight participants requested workflow features that enable iteration: six participants (P2, P3, P4, P5, P7, P8) specifically requested capabilities for branching runs, dataset slicing, richer comparisons, or sample-level issue tracking. P1 focused on LLM-based interpretation features, exporting results for LLM-based analysis and suggesting an integrated LLM interface for result interpretation, while P6 mostly emphasized the need for better runtime visibility through progress indicators similar to TQDM[1]. These requests indicate that diagnostic depth depends more on the workflow glue than on raw method coverage. Notably, P5 emphasized that mature, reliable methods are more valuable than cutting-edge techniques: *"We don't need the newest stuff [methods] because if you're going to use it, you'd have to be an expert in the field. […] Once the tool is established […] it should be in the toolbox"*, adding: *"If it's too new, it doesn't run and then what use is it in a toolkit?"*.

(3) Constraint-aware execution. Execution constraints (single job, platform limits) shape what diagnostic work is feasible, especially within time-bounded sessions. Tooling should make constraints visible, minimize disruption, and support partial progress (e.g., clear status, recoverable results, stable navigation to artifacts). Participants adapted their diagnostic workflows to address execution uncertainty. Several canceled long-running jobs early to preserve time for other diagnostic steps, checked queue states repeatedly without receiving consistent feedback, and in some cases were unable to complete planned runs due to compute constraints (I7). While some of these constraints reflect the prototype's deployment context (VM limitations, single-job execution), this behavioral adaptation, driven by opaque execution state and unpredictable runtimes rather than deliberate diagnostic decisions, indicates a genuine need for constraint-aware execution support in diagnostic tooling more broadly: transparent status, recoverable results, and graceful degradation under resource limits.

Together, these implications suggest that effective diagnostic tooling is achieved by combining method access with workflow scaffolding, interpretive guidance, and constraint-aware execution.

7.4. Limitations

As discussed in the methodology (Chapter 5), this study has several limitations that shape the interpretation of findings. First, unfamiliarity with the tool and the 30-minute hands-on session emphasized early diagnostic behaviors over deeper iteration toward stable failure pattern hypotheses, meaning our findings characterize early-phase diagnostic behavior rather than sustained diagnostic work. Second, the toolkit is a research prototype rather than production software, and platform constraints (VM limitations, single-job execution, occasional reliability issues) shaped what participants could practically accomplish. Third, the evaluation tasks were constrained to summarization and simplification within an academic search context. Transferability to other task types (e.g., question answering, code generation) or domains (e.g., clinical application, content moderation) remains to be established. Fourth, while participants represented a range of practitioner backgrounds (see Table 6.1), the study does not systematically characterize how expertise levels or prior experience with interpretability tools influenced diagnostic strategies, limiting our ability to make expertise-dependent claims. Taken together, these limitations shape the interpretation of a key finding: the absence of completed diagnostic iteration loops across all sessions reflects the intersection of time pressure, prototype constraints, and interpretation barriers rather than solely practitioner capability or framework limitations. The study successfully

[1] TQDM is a Python library that provides progress bars for loops and long-running processes, giving users real-time feedback on execution status. `https://tqdm.github.io/`

observed early-phase diagnostic patterns and systematic barriers to progression, but cannot make strong claims about later-stage hypothesis refinement or longitudinal evolution of the diagnostic workflow. Findings should therefore be interpreted as characterizing tool-supported early-phase diagnosis in a controlled setting rather than as a comprehensive account of diagnostic practice in the wild.

7.5. Future Work

Several empirical and conceptual questions raised by this study remain open and point to directions for future research.

Longitudinal evaluation of the full diagnostic cycle. This study observed only early diagnostic phases: all eight participants remained at the level of expectation formation, baseline calibration, and tentative hypothesis formation. None progressed to the sustained, multi-cycle hypothesis refinement that the framework characterizes as failure pattern identification. Whether the framework's later phases accurately describe how practitioners reason under conditions of sufficient time and scaffolding, and whether the toolkit's investigative cycle scaffolding translates into completed diagnostic loops, remain empirically open questions. Longitudinal evaluation, conducted by observing the same practitioners across multiple sessions on real-world tasks, would enable observation of complete diagnostic cycles and stronger evidence for or against the framework's characterization of later phases. Such designs would also represent a methodological advance for the field, as prior empirical studies of diagnostic and interpretability tool use have similarly relied on single-session snapshots.

Transferability of the framework across task types and domains. The evaluation was constrained to summarization and simplification tasks in an academic search context. The diagnostic framework's core constructs (the symptom/sign/failure-pattern framework, and the SK/RK lens) were developed and evaluated in this setting. Whether these constructs transfer to structurally different task types (e.g., question answering, code generation, dialogue) or higher-stakes domains (e.g., clinical NLP, content moderation) is an open empirical question. Task types that differ in output structure, ground-truth availability, or failure mode profile may require extensions or revisions to the framework's vocabulary and diagnostic sequence.

Practitioner variation and expertise effects. Chapter 6 documents variation in diagnostic behavior across participants: the baseline-first pattern was less systematic for less experienced practitioners (P6), and more experienced participants tended to encounter workflow glue limitations rather than interpretation barriers. However, the study was not designed to characterize how expertise, domain familiarity, or prior experience with interpretability tools shapes diagnostic strategy. A study designed to sample across these dimensions would enable more targeted claims about which framework components and toolkit features matter most for which practitioner profiles, and whether the investigative cycle scaffolding is expertise-dependent in its effectiveness.

Operationalizing failure pattern identification. The framework positions failure pattern identification as the endpoint of iterative diagnostic investigation, but this endpoint was not observed in the present study. It remains unclear what conditions are sufficient for a practitioner to move from a failure pattern hypothesis, as observed in P3's session, to a confirmed and documented failure pattern. Specifically, open questions concern how much evidence must be accumulated, how many diagnostic cycles must be completed, and what reasoning moves bridge hypothesis formation and pattern confirmation. Relatedly, participants expected to associate framework concepts directly with individual outputs rather than at the job level (I5), suggesting the framework's vocabulary may require finer-grained operationalization to support failure pattern documentation in practice. Future work should investigate both the cognitive and tooling conditions under which failure pattern identification is achievable, and how framework concepts can be operationalized at the sample level to support that process.

7.6. Summary

This chapter interpreted the results in relation to the research questions and contributions. Diagnostic support was found to depend less on adding more interpretability methods and more on scaffolding

expectation formation, baseline calibration, triangulation, comparison, and state externalization, while addressing gating factors related to interpretation, iteration infrastructure, and execution constraints. The findings provide empirical validation for prior characterizations of diagnosis as expectation-driven and evidence-based, while also revealing that participants largely remained in early diagnostic phases within the study's time-bounded setting. The toolkit successfully reduced diagnostic overhead and integrated evaluation methods into a cohesive workflow, but interpretation burden and gaps in workflow glue limited deeper hypothesis refinement. These results support a view of diagnostic tooling as scaffolding for hypothesis-driven reasoning rather than as a library of explanation techniques, with implications for tool design, the scope and future development of the diagnostic framework, and future empirical research into practitioner diagnostic workflows. The next chapter concludes the thesis and summarizes contributions.

8

Conclusion

This thesis investigated what kinds of tooling support practitioners in diagnosing undesirable behavior in large language models. It introduced (i) a diagnostic framework that structures diagnosis through symptoms, signs, and failure patterns and a complementary Should-Know/Really-Know lens, and (ii) a toolkit prototype that operationalizes these concepts in a practitioner-facing workflow. An evaluation study with eight practitioners examined how the framework and toolkit support diagnostic reasoning in practice, revealing both their utility and the conditions that gate deeper diagnostic work.

8.1. Summary of Contributions

This thesis makes three primary contributions:

A diagnostic framework for LLM failures. The framework conceptualizes diagnosis as a hypothesis-driven, iterative process that proceeds from symptoms (observed undesirable outputs) through signs (evidence from interpretability methods and metrics) towards failure patterns (recurring, explainable weaknesses). It adopts a complementary Should-Know/Really-Know lens that distinguishes between what a model is expected to know based on task requirements and what it actually knows, revealed through probing. This framing positions diagnosis as distinct from explainability: diagnosis answers contrastive *why* questions (”Why did the model fail here, as opposed to succeeding elsewhere?”), while explainability addresses descriptive questions about individual predictions.

A prototype diagnostic toolkit. The toolkit instantiates the diagnostic framework through integrated evaluation, run comparison, and state externalization features. It embeds metrics and interpretability methods (including the SK/RK lens) directly into a cohesive workflow, reducing setup friction and enabling baseline-driven calibration, evidence triangulation, and hypothesis testing via comparison. The tool complements rather than replaces existing practice by acting as an organization layer that supports quick baselines, persistent diagnostic state, and interoperability with external analysis tools.

Empirical insights into practitioner diagnostic behavior and design implications for diagnostic tooling. The evaluation study with eight practitioners characterizes how practitioners approach diagnosis with tool support: they adopt a baseline-first strategy, triangulate across samples metrics, and methods rather than relying on single signals, and use comparison as a central sense-making operation for hypothesis testing. These insights validate the framework’s early phases while identifying three gating factors that constrain diagnostic depth pointing to design implications that emphasize scaffolding the full investigative cycle rather than providing isolated analytical methods.

8.2. Answer to Research Question

The findings indicate that tooling supports for diagnosing undesirable LLM behavior are best understood as scaffolding for diagnostic reasoning rather than as a library of explanation techniques. Effec-

tive diagnostic tooling should provide five core capabilities: (i) helping practitioners specify and refine expectations about model behavior, (ii) enabling baseline-driven calibration to ground expectations in observed outputs, (iii) supporting triangulation across evidence types to manage uncertainty, (iv) making comparison a first-class operation for hypothesis testing, and (v) externalizing diagnostic state so that findings remain traceable and communicable across sessions. However, these capabilities alone are insufficient. Diagnostic depth is systematically impeded by three interacting factors: the availability of guidance for interpreting what methods reveal and how to act on findings; workflow infrastructure that supports iterative hypothesis refinement; and constraint-aware execution that makes limitations visible and sustains partial progress. Together, these findings position diagnostic tooling as infrastructure for sustained inquiry rather than as a collection of isolated analytical methods.

8.3. Closing Remarks

As large language models are deployed in increasingly consequential settings, the ability to systematically investigate their failures becomes essential. The distributed, emergent nature of LLM behavior means that undesirable outputs often cannot be traced to discrete bugs or localized faults. Instead, diagnosis requires sustained reasoning about what models should know, what they really know, and where systematic gaps or misalignment emerge. This thesis contributes both a conceptual framework for structuring such reasoning and a prototype tool that demonstrates how workflow-integrated scaffolding can support practitioners in moving from isolated observations toward coherent failure pattern hypotheses. The evaluation study reveals that while early diagnostic phases are well-supported by the framework and toolkit, sustained hypothesis-driven iteration is systematically impeded by interpretation barriers, navigation friction, and compound cognitive load. Addressing these barriers requires more than surface-level adjustments to interpretation guidance, workflow infrastructure, and constraint-aware execution. Future work should focus on these gating factors, expanding evaluation to longitudinal settings that enable observation of complete diagnostic cycles, and extending the framework to broader task domains and failure modes. Ultimately, effective diagnosis of complex LLM-based systems depends not on better explanations alone, but on better support for the investigative work that surrounds them.

9

Code and Data Availability

The code and curated datasets used in this thesis are associated with an outgoing publication and will be made publicly available upon its release. All code and data will be released under the Creative Commons (CC BY 4.0) license, allowing for reuse with proper attribution. For any questions regarding code or data access in the meantime, please contact the author.

10

AI Disclosure Statement

Throughout this thesis, I used AI-based tools (ChatGPT, Claude, and GitHub Copilot) for the following purposes: rephrasing and improving the clarity and flow of text I had written; surfacing potentially relevant academic literature for further review; and writing and debugging code. In all cases, the ideas, arguments, structure, and conclusions are my own. All AI-generated outputs were critically reviewed, verified, and edited by me. I take full responsibility for the accuracy, originality, and academic integrity of all content in this thesis, including any material developed with AI assistance.

Acknowledgement

Eight years is a long time. Looking back, I am struck less by the hours spent and more by how thoroughly this journey has changed me. From a practical HBO student finding his footing, to a bridging-programme survivor, to now, an MSc student three weeks from his final presentation, cautiously optimistic about what comes next. I look back on all of it, the highs, the lows, and the perfectly ordinary study days, with a smile. I hope to be writing something similar at the end of the next chapter.

This work would not exist without two supervisors who each, in their own way, helped me find my way forward. Lorenzo, you were available to me every single day, whether it was a small question on Teams or a larger conceptual puzzle I could not shake. You had a way of making it feel completely normal to be confused, while still gently pointing towards clarity. Jie, our weekly meetings were anchors. Your probing questions always reached the part of my reasoning I had not yet examined, and you helped me see what I was actually doing when I thought I was simply lost in it. Thank you both for your patience, your insight, and your generosity with your time. I could not have asked for better supervisors.

Nina, I am not sure I have the words for this. In a quiet way, the fact that we met at all still strikes me. You are my partner, my friend, my colleague, and my greatest source of support throughout this journey, and most importantly, you helped me fight through the bridging programme when I needed it most. Without you, I would not have made it through that door, let alone to the other side of it. I am deeply grateful to have you in my life.

To my parents, Caroline and Geert-Jan: you gave me something that is easy to underestimate and impossible to replace. Mom, thank you for your unconditional support and for always being there to listen and guide. Dad, thank you for your optimism and for never shying away from a critical discussion about my topic. Together, you gave me the freedom to always pursue what genuinely interested me, and I would not be the person I am today without that.

To the friends who made these years worth it:

To the Stieltjes Gaang; for the tea, the *tea*, and for letting me weasel my way into what was originally not my friend group at all. The conversations, the warmth, and the simple pleasure of an evening spent doing nothing in particular meant more than you might realize.

To my D&D friends; thank you for the reliable good times, for the worlds we built (and destroyed) together, the stories we told, and for always being up for whatever the evening called for. You were exactly the kind of escape that I needed.

To my Pokemon GO friends, who periodically forced me to touch grass (a more medically necessary intervention than it sounds); you always managed to lift my spirits, and I am genuinely glad we keep finding excuses to meet up, walk for hours, and talk about everything and nothing.

And to my (study) friends, from HBO through the bridging programme to the MSc and the long-lost ones in between; we covered a lot of ground together, in every sense of the phrase, and I look back on all of it fondly. Thank you for the camaraderie, the commiseration, and the shared triumphs. I hope we will continue to find ways to support each other in the next chapters of our lives.

Three weeks from now, I will give my final presentation. What comes after, I do not yet know with certainty, though I hope it involves a PhD at the TU Delft. Perhaps by the time I read this again, I will already know how that turned out. Whatever happens, I am grateful for the journey that has brought me here, and for all the people who have been part of it. *Thank you.*

J.S. Beekman
Delft, April 2026

References

[1] Stanford University Human-Centered Artificial Intelligence [HAI]. *Artificial Intelligence Index Report 2025*. Tech. rep. 2025. URL: https://hai.stanford.edu/ai-index/2025-ai-index-report.

[2] Reduan Achtibat et al. "AttnLRP: attention-aware layer-wise relevance propagation for transformers". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2024.

[3] J Alammar. "Ecco: An Open Source Library for the Explainability of Transformer Language Models". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Ed. by Heng Ji, Jong C. Park, and Rui Xia. Online: Association for Computational Linguistics, Aug. 2021, pp. 249–257. DOI: 10.18653/v1/2021.acl-demo.30. URL: https://aclanthology.org/2021.acl-demo.30/.

[4] David Alvarez-Melis and Tommi S. Jaakkola. "Towards robust interpretability with self-explaining neural networks". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 7786–7795.

[5] Saleema Amershi et al. "Software Engineering for Machine Learning: A Case Study". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 2019, pp. 291–300. DOI: 10.1109/ICSE-SEIP.2019.00042.

[6] Anthropic. *Home Anthropic*. URL: https://www.anthropic.com/.

[7] Katie Atkinson, Trevor Bench-Capon, and Danushka Bollegala. "Explanation in AI and law: Past, present and future". In: *Artificial Intelligence* 289 (Sept. 2020), p. 103387. DOI: 10.1016/j.artint.2020.103387. URL: https://doi.org/10.1016/j.artint.2020.103387.

[8] Giuseppe Attanasio et al. "ferret: a Framework for Benchmarking Explainers on Transformers". In: *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Ed. by Danilo Croce and Luca Soldaini. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 256–266. DOI: 10.18653/v1/2023.eacl-demo.29. URL: https://aclanthology.org/2023.eacl-demo.29/.

[9] Agathe Balayn et al. "Faulty or Ready? Handling Failures in Deep-Learning Computer Vision Models until Deployment: A Study of Practices, Challenges, and Needs". In: *CHI '23: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Apr. 2023), pp. 1–20. DOI: 10.1145/3544548.3581555. URL: https://doi.org/10.1145/3544548.3581555.

[10] Agathe Balayn et al. "How can Explainability Methods be Used to Support Bug Identification in Computer Vision Models?" In: *CHI Conference on Human Factors in Computing Systems* (Apr. 2022), pp. 1–16. DOI: 10.1145/3491102.3517474. URL: https://doi.org/10.1145/3491102.3517474.

[11] Satanjeev Banerjee and Alon Lavie. "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ed. by Jade Goldstein et al. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 65–72. URL: https://aclanthology.org/W05-0909/.

[12] Yonatan Belinkov and James Glass. "Analysis Methods in Neural Language Processing: A Survey". In: *Transactions of the Association for Computational Linguistics* 7 (2019). Ed. by Lillian Lee et al., pp. 49–72. DOI: 10.1162/tacl_a_00254. URL: https://aclanthology.org/Q19-1004/.

[13] Steven Bird and Edward Loper. “NLTK: The Natural Language Toolkit”. In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 214–217. URL: https://aclanthology.org/P04-3031/.

[14] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258 [cs.LG]. URL: https://arxiv.org/abs/2108.07258.

[15] Richard E. Boyatzis. *Transforming qualitative information: thematic analysis and code development*. Apr. 1998. URL: http://bvbr.bib-bvb.de:8991/F?func=service&doc_library=BVB01&local_base=BVB01&doc_number=008210667&sequence=000002&line_number=0001&func_code=DB_RECORDS&service_type=MEDIA.

[16] V. Braun and V. Clarke. *Thematic Analysis*. URL: https://www.thematicanalysis.net/.

[17] Virginia Braun and Victoria Clarke. “Using thematic analysis in psychology”. In: *Qualitative Research in Psychology* 3.2 (2006), pp. 77–101. DOI: 10.1191/1478088706qp063oa. eprint: https://doi.org/10.1191/1478088706qp063oa. URL: https://doi.org/10.1191/1478088706qp063oa.

[18] S. Brown. *The C4 model for visualising software architecture*. Feb. 2010. URL: https://c4model.com/.

[19] Simon Brown. “The C4 model for software architecture”. In: (June 2018). URL: https://www.infoq.com/articles/C4-architecture-model/.

[20] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[21] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. “Semantics derived automatically from language corpora contain human-like biases”. In: *Science* 356.6334 (2017), pp. 183–186. DOI: 10.1126/science.aal4230. eprint: https://www.science.org/doi/pdf/10.1126/science.aal4230. URL: https://www.science.org/doi/abs/10.1126/science.aal4230.

[22] Arman Cohan et al. “SPECTER: Document-level Representation Learning using Citation-informed Transformers”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 2270–2282. DOI: 10.18653/v1/2020.acl-main.207. URL: https://aclanthology.org/2020.acl-main.207/.

[23] Alexis Conneau et al. “What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 2126–2136. DOI: 10.18653/v1/P18-1198. URL: https://aclanthology.org/P18-1198/.

[24] Karina Cortiñas-Lorenzo, Wanling Cai, and Gavin Doherty. “Designing, Implementing, and Evaluating AI Explanations: A Scoping Review of Explainable AI Frameworks”. In: *ACM Trans. Comput.-Hum. Interact.* 32.6 (Dec. 2025). ISSN: 1073-0516. DOI: 10.1145/3769678. URL: https://doi.org/10.1145/3769678.

[25] DeepSeek. *DeepSeek*. URL: https://www.deepseek.com/en.

[26] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

[27] Mahdi Dhaini et al. *Adoption of Explainable Natural Language Processing: Perspectives from Industry and Academia on Practices and Challenges*. 2025. arXiv: 2508.09786 [cs.CL]. URL: https://arxiv.org/abs/2508.09786.

[28] Kaustubh Dhole et al. “NL-Augmenter: A Framework for Task-Sensitive Natural Language Augmentation”. In: *Northern European Journal of Language Technology* 9 (2023). Ed. by Leon Derczynski. DOI: 10.3384/nejlt.2000-1533.2023.4725. URL: https://aclanthology.org/2023.nejlt-1.5/.

[29] Carmine M DiMascio. *py-readability-metrics*. 2019. URL: https://py-readability-metrics.readthedocs.io/en/latest/.

[30] Finale Doshi-Velez and Been Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017. arXiv: 1702.08608 [stat.ML]. URL: https://arxiv.org/abs/1702.08608.

[31] Allyson Ettinger. “What BERT Is Not: Lessons from a New Suite of Psycholinguistic Diagnostics for Language Models”. In: *Transactions of the Association for Computational Linguistics* 8 (Jan. 2020), pp. 34–48. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00298. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00298/1923116/tacl_a_00298.pdf. URL: https://doi.org/10.1162/tacl_a_00298.

[32] Explosion. *SPACY · Industrial-strength Natural language processing in Python*. URL: https://spacy.io/.

[33] Nils Feldhus et al. “InterroLang: Exploring NLP Models and Datasets through Dialogue-based Explanations”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5399–5421. DOI: 10.18653/v1/2023.findings-emnlp.359. URL: https://aclanthology.org/2023.findings-emnlp.359/.

[34] Jennifer Fereday and Eimear Muir-Cochrane. “Demonstrating rigor using thematic analysis: a hybrid approach of inductive and deductive coding and theme development”. In: *International Journal of Qualitative Methods* 5.1 (Mar. 2006), pp. 80–92. DOI: 10.1177/160940690600500107. URL: https://doi.org/10.1177/160940690600500107.

[35] Javier Ferrando et al. “Automating Behavioral Testing in Machine Translation”. In: *Proceedings of the Eighth Conference on Machine Translation*. Ed. by Philipp Koehn et al. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1014–1030. DOI: 10.18653/v1/2023.wmt-1.97. URL: https://aclanthology.org/2023.wmt-1.97/.

[36] Christopher Frauenberger. “Critical Realist HCI”. In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’16. San Jose, California, USA: Association for Computing Machinery, 2016, pp. 341–351. ISBN: 9781450340823. DOI: 10.1145/2851581.2892569. URL: https://doi.org/10.1145/2851581.2892569.

[37] Mor Geva et al. “LM-Debugger: An Interactive Tool for Inspection and Intervention in Transformer-Based Language Models”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Wanxiang Che and Ekaterina Shutova. Abu Dhabi, UAE: Association for Computational Linguistics, Dec. 2022, pp. 12–21. DOI: 10.18653/v1/2022.emnlp-demos.2. URL: https://aclanthology.org/2022.emnlp-demos.2/.

[38] Karan Goel et al. “Robustness Gym: Unifying the NLP Evaluation Landscape”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*. Ed. by Avi Sil and Xi Victoria Lin. Online: Association for Computational Linguistics, June 2021, pp. 42–55. DOI: 10.18653/v1/2021.naacl-demos.6. URL: https://aclanthology.org/2021.naacl-demos.6/.

[39] Greg Guest, Kathleen MacQueen, and Emily Namey. *Applied Thematic Analysis*. Jan. 2012. DOI: 10.4135/9781483384436. URL: https://doi.org/10.4135/9781483384436.

[40] Sandra G. Hart and Lowell E. Staveland. “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research”. In: *Human Mental Workload*. Ed. by Peter A. Hancock and Najmedin Meshkati. Vol. 52. Advances in Psychology. North-Holland, 1988, pp. 139–183. DOI: https://doi.org/10.1016/S0166-4115(08)62386-9. URL: https://www.sciencedirect.com/science/article/pii/S0166411508623869.

[41] Karen Henwood and Nick Pidgeon. “Beyond the qualitative paradigm: A framework for introducing diversity within qualitative psychology”. In: *Journal of Community & Applied Social Psychology* 4.4 (1994), pp. 225–238. DOI: https://doi.org/10.1002/casp.2450040403. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/casp.2450040403. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/casp.2450040403.

[42] Alan R. Hevner et al. “Design science in information systems research”. In: *MIS Q.* 28.1 (Mar. 2004), pp. 75–105. ISSN: 0276-7783.

[43] Ester Hlavnova and Sebastian Ruder. “Empowering Cross-lingual Behavioral Testing of NLP Models with Typological Features”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 7181–7198. DOI: 10.18653/v1/2023.acl-long.396. URL: https://aclanthology.org/2023.acl-long.396/.

[44] Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. “exBERT: A Visual Analysis Tool to Explore Learned Representations in Transformer Models”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Ed. by Asli Celikyilmaz and Tsung-Hsien Wen. Online: Association for Computational Linguistics, July 2020, pp. 187–196. DOI: 10.18653/v1/2020.acl-demos.22. URL: https://aclanthology.org/2020.acl-demos.22/.

[45] Jen-tse Huang et al. “AEON: a method for automatic evaluation of NLP test cases”. In: *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. ISSTA 2022. Virtual, South Korea: Association for Computing Machinery, 2022, pp. 202–214. ISBN: 9781450393799. DOI: 10.1145/3533767.3534394. URL: https://doi.org/10.1145/3533767.3534394.

[46] A.M. Huberman and M.B. Miles. *Qualitative Data Analysis: an expanded Sourcebook, 2nd Edition*. 2nd ed. 4. SAGE Publications, Inc, Jan. 1994, pp. 336–337. URL: https://books.google.nl/books/about/Qualitative_data_analysis.html?id=U4lU_-wJ5QEC&redir_esc=y.

[47] Alon Jacovi and Yoav Goldberg. “Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness?” In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 4198–4205. DOI: 10.18653/v1/2020.acl-main.386. URL: https://aclanthology.org/2020.acl-main.386/.

[48] Sarthak Jain and Byron C. Wallace. “Attention is not Explanation”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 3543–3556. DOI: 10.18653/v1/N19-1357. URL: https://aclanthology.org/N19-1357/.

[49] Sami Kabir, Mohammad Shahadat Hossain, and Karl Andersson. “A Review of Explainable Artificial Intelligence from the Perspectives of Challenges and Opportunities”. In: *Algorithms* 18.9 (2025). ISSN: 1999-4893. DOI: 10.3390/a18090556. URL: https://www.mdpi.com/1999-4893/18/9/556.

[50] Harmanpreet Kaur et al. “Interpreting Interpretability: Understanding Data Scientists’ Use of Interpretability Tools for Machine Learning”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20. Honolulu, HI, USA: Association for Computing Machinery, 2020, pp. 1–14. ISBN: 9781450367080. DOI: 10.1145/3313831.3376219. URL: https://doi.org/10.1145/3313831.3376219.

[51] Nigel King. “Using Templates in the Thematic Analysis of Text”. In: Jan. 2004, pp. 257–270. ISBN: 9780761948889. DOI: 10.4135/9781446280119.n21.

[52] Thomas Kluyver et al. “Jupyter Notebooks—a publishing format for reproducible computational workflows”. In: *IOS Press*. 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

[53] Pang Wei Koh and Percy Liang. *Understanding Black-box Predictions via Influence Functions*. 2020. arXiv: 1703.04730 [stat.ML]. URL: https://arxiv.org/abs/1703.04730.

[54] Narine Kokhlikyan et al. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG]. URL: https://arxiv.org/abs/2009.07896.

[55] Bettina Laugwitz, Theo Held, and Martin Schrepp. "Construction and Evaluation of a User Experience Questionnaire". In: *HCI and Usability for Education and Work*. Ed. by Andreas Holzinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 63–76. ISBN: 978-3-540-89350-9.

[56] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[57] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. arXiv: 1705.07874 [cs.AI]. URL: https://arxiv.org/abs/1705.07874.

[58] Anna Madill, Abbie Jordan, and Caroline Shirley. "Objectivity and reliability in qualitative analysis: Realist, contextualist and radical constructionist epistemologies". In: *British Journal of Psychology* 91.1 (2000), pp. 1–20. DOI: https://doi.org/10.1348/000712600161646. eprint: https://bpspsychub.onlinelibrary.wiley.com/doi/pdf/10.1348/000712600161646. URL: https://bpspsychub.onlinelibrary.wiley.com/doi/abs/10.1348/000712600161646.

[59] Chandler May et al. "On Measuring Social Biases in Sentence Encoders". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 622–628. DOI: 10.18653/v1/N19-1063. URL: https://aclanthology.org/N19-1063/.

[60] R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. "Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3428–3448. DOI: 10.18653/v1/P19-1334. URL: https://aclanthology.org/P19-1334/.

[61] Melkamu Abay Mersha, Mesay Gemeda Yigezu, and Jugal Kalita. "Evaluating the effectiveness of XAI techniques for encoder-based language models". In: *Knowledge-Based Systems* 310 (Jan. 2025), p. 113042. DOI: 10.1016/j.knosys.2025.113042. URL: https://doi.org/10.1016/j.knosys.2025.113042.

[62] Tim Miller. "Explanation in artificial intelligence: Insights from the social sciences". In: *Artificial Intelligence* 267 (Oct. 2018), pp. 1–38. DOI: 10.1016/j.artint.2018.07.007. URL: https://doi.org/10.1016/j.artint.2018.07.007.

[63] Joao Luis Zeni Montenegro, Cristiano André da Costa, and Rodrigo da Rosa Righi. "Survey of conversational agents in health". In: *Expert Systems with Applications* 129 (2019), pp. 56–67. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.03.054. URL: https://www.sciencedirect.com/science/article/pii/S0957417419302283.

[64] Arghavan Moradi Dakhel et al. "GitHub Copilot AI pair programmer: Asset or Liability?" In: *Journal of Systems and Software* 203 (2023), p. 111734. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2023.111734. URL: https://www.sciencedirect.com/science/article/pii/S0164121223001292.

[65] John Morris et al. "TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Qun Liu and David Schlangen. Online: Association for Computational Linguistics, Oct. 2020, pp. 119–126. DOI: 10.18653/v1/2020.emnlp-demos.16. URL: https://aclanthology.org/2020.emnlp-demos.16/.

[66] Fuseini Mumuni and Alhassan Mumuni. *Explainable artificial intelligence (XAI): from inherent explainability to large language models*. 2025. arXiv: 2501.09967 [cs.LG]. URL: https://arxiv.org/abs/2501.09967.

[67] Aakanksha Naik et al. "Stress Test Evaluation for Natural Language Inference". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Ed. by Emily M. Bender, Leon Derczynski, and Pierre Isabelle. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 2340–2353. URL: https://aclanthology.org/C18-1198/.

[68] J.N. Nelen. *Developing a user-centered explainability tool to support the NLP Data Scientist in creating LLM-based solutions*. 2024. URL: https://resolver.tudelft.nl/uuid:700ce316-5047-4182-ae25-e47db8ca4054.

[69] Harsha Nori et al. *InterpretML: A Unified Framework for Machine Learning Interpretability*. 2019. arXiv: 1909.09223 [cs.LG]. URL: https://arxiv.org/abs/1909.09223.

[70] OpenAI. *OpenAI*. URL: https://openai.com/.

[71] Avash Palikhe et al. *Towards Transparent AI: A Survey on Explainable Language Models*. 2025. arXiv: 2509.21631 [cs.CL]. URL: https://arxiv.org/abs/2509.21631.

[72] Denis Paperno et al. "The LAMBADA dataset: Word prediction requiring a broad discourse context". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1525–1534. DOI: 10.18653/v1/P16-1144. URL: https://aclanthology.org/P16-1144/.

[73] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/.

[74] Letitia Parcalabescu and Anette Frank. "On Measuring Faithfulness or Self-consistency of Natural Language Explanations". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 6048–6089. DOI: 10.18653/v1/2024.acl-long.329. URL: https://aclanthology.org/2024.acl-long.329/.

[75] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[76] Jeffrey M. Perkel. "Why Jupyter is data scientists' computational notebook of choice". In: *Nature* 563.7729 (Oct. 2018), pp. 145–146. DOI: 10.1038/d41586-018-07196-1. URL: https://doi.org/10.1038/d41586-018-07196-1.

[77] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. Tech. rep. 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[78] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[79] Lucas Resck, Isabelle Augenstein, and Anna Korhonen. "Explainability and Interpretability of Multilingual Large Language Models: A Survey". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 20465–20497. ISBN: 979-8-89176-332-6. DOI: 10.18653/v1/2025.emnlp-main.1033. URL: https://aclanthology.org/2025.emnlp-main.1033/.

[80] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. arXiv: 1602.04938 [cs.LG]. URL: https://arxiv.org/abs/1602.04938.

[81] Marco Tulio Ribeiro et al. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, July 2020, pp. 4902–4912. DOI: 10.18653/v1/2020.acl-main.442. URL: https://aclanthology.org/2020.acl-main.442/.

[82] Samantha Robertson et al. “Angler: Helping Machine Translation Practitioners Prioritize Model Improvements”. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI ’23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394215. DOI: 10.1145/3544548.3580790. URL: https://doi.org/10.1145/3544548.3580790.

[83] Paul Röttger et al. “HateCheck: Functional Tests for Hate Speech Detection Models”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong et al. Online: Association for Computational Linguistics, Aug. 2021, pp. 41–58. DOI: 10.18653/v1/2021.acl-long.4. URL: https://aclanthology.org/2021.acl-long.4/.

[84] Paul Röttger et al. “XSTest: A Test Suite for Identifying Exaggerated Safety Behaviours in Large Language Models”. In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Ed. by Kevin Duh, Helena Gomez, and Steven Bethard. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 5377–5400. DOI: 10.18653/v1/2024.naacl-long.301. URL: https://aclanthology.org/2024.naacl-long.301/.

[85] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* 1.5 (May 2019), pp. 206–215. DOI: 10.1038/s42256-019-0048-x. URL: https://doi.org/10.1038/s42256-019-0048-x.

[86] Pranab Sahoo et al. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2025. arXiv: 2402.07927 [cs.AI]. URL: https://arxiv.org/abs/2402.07927.

[87] Gabriele Sarti et al. “Inseq: An Interpretability Toolkit for Sequence Generation Models”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Ed. by Danushka Bollegala, Ruihong Huang, and Alan Ritter. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 421–435. DOI: 10.18653/v1/2023.acl-demo.40. URL: https://aclanthology.org/2023.acl-demo.40/.

[88] D. Sculley et al. “Hidden technical debt in Machine learning systems”. In: *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 2503–2511.

[89] Arshdeep Sekhon et al. “White-box Testing of NLP models with Mask Neuron Coverage”. In: *Findings of the Association for Computational Linguistics: NAACL 2022*. Ed. by Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 1547–1558. DOI: 10.18653/v1/2022.findings-naacl.116. URL: https://aclanthology.org/2022.findings-naacl.116/.

[90] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[91] Sofia Serrano and Noah A. Smith. “Is Attention Interpretable?” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 2931–2951. DOI: 10.18653/v1/P19-1282. URL: https://aclanthology.org/P19-1282/.

[92] A Singla et al. *The state of AI in 2025: Agents, innovation, and transformation*. Nov. 2025. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai.

[93] Alisa Smirnova, Jie Yang, and Philippe Cudre-Mauroux. “XCrowd: Combining Explainability and Crowdsourcing to Diagnose Models in Relation Extraction”. In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. CIKM ’24. Boise, ID, USA: Association for Computing Machinery, 2024, pp. 2097–2107. ISBN: 9798400704369. DOI: 10.1145/3627673.3679777. URL: https://doi.org/10.1145/3627673.3679777.

[94] Eric Michael Smith et al. ""I'm sorry to hear that": Finding New Biases in Language Models with a Holistic Descriptor Dataset". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 9180–9211. DOI: 10.18653/v1/2022.emnlp-main.625. URL: https://aclanthology.org/2022.emnlp-main.625/.

[95] Seonil (Simon) Son et al. "HaRiM$^+$: Evaluating Summary Quality with Hallucination Risk". In: *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Yulan He et al. Online only: Association for Computational Linguistics, Nov. 2022, pp. 895–924. DOI: 10.18653/v1/2022.aacl-main.66. URL: https://aclanthology.org/2022.aacl-main.66/.

[96] GitHub Staff. *Octoverse: A new developer joins GitHub every second as AI leads TypeScript to #1*. Jan. 2026. URL: https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/.

[97] National Institute of Standards and Technology [NIST]. *TREC-COVID home*. 2020. URL: https://ir.nist.gov/covidSubmit/index.html.

[98] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG]. URL: https://arxiv.org/abs/1703.01365.

[99] Ian Tenney et al. "The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Qun Liu and David Schlangen. Online: Association for Computational Linguistics, Oct. 2020, pp. 107–118. DOI: 10.18653/v1/2020.emnlp-demos.15. URL: https://aclanthology.org/2020.emnlp-demos.15/.

[100] Ian Tenney et al. "The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Qun Liu and David Schlangen. Online: Association for Computational Linguistics, Oct. 2020, pp. 107–118. DOI: 10.18653/v1/2020.emnlp-demos.15. URL: https://aclanthology.org/2020.emnlp-demos.15/.

[101] Nandan Thakur et al. *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models*. 2021. arXiv: 2104.08663 [cs.IR]. URL: https://arxiv.org/abs/2104.08663.

[102] Haoye Tian et al. *A Taxonomy of Prompt Defects in LLM Systems*. 2025. arXiv: 2509.14404 [cs.SE]. URL: https://arxiv.org/abs/2509.14404.

[103] Betty Van Aken, Sebastian Herrmann, and Alexander Löser. "What Do You See in this Patient? Behavioral Testing of Clinical NLP Models". In: *Proceedings of the 4th Clinical Natural Language Processing Workshop*. Ed. by Tristan Naumann et al. Seattle, WA: Association for Computational Linguistics, July 2022, pp. 63–73. DOI: 10.18653/v1/2022.clinicalnlp-1.7. URL: https://aclanthology.org/2022.clinicalnlp-1.7/.

[104] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[105] Jesse Vig. *Visualizing Attention in Transformer-Based Language Representation Models*. 2019. arXiv: 1904.02679 [cs.HC]. URL: https://arxiv.org/abs/1904.02679.

[106] Vaishali Vinay. "A System-Level taxonomy of failure modes in large language model applications". In: *arXiv (Cornell University)* (Nov. 2025). DOI: 10.48550/arxiv.2511.19933. URL: https://doi.org/10.48550/arxiv.2511.19933.

[107] Sandra Wachter, Brent Mittelstadt, and Chris Russell. *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. 2018. arXiv: 1711.00399 [cs.AI]. URL: https://arxiv.org/abs/1711.00399.

[108] David Wadden et al. “Fact or Fiction: Verifying Scientific Claims”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 7534–7550. DOI: 10.18653/v1/2020.emnlp-main.609. URL: https://aclanthology.org/2020.emnlp-main.609/.

[109] Eric Wallace et al. “AllenNLP Interpret: A Framework for Explaining Predictions of NLP Models”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*. Ed. by Sebastian Padó and Ruihong Huang. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 7–12. DOI: 10.18653/v1/D19-3002. URL: https://aclanthology.org/D19-3002/.

[110] Boshi Wang et al. “Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 2717–2739. DOI: 10.18653/v1/2023.acl-long.153. URL: https://aclanthology.org/2023.acl-long.153/.

[111] Qianli Wang et al. “LLMCheckup: Conversational Examination of Large Language Models via Interpretability Tools and Self-Explanations”. In: *Proceedings of the Third Workshop on Bridging Human–Computer Interaction and Natural Language Processing*. Ed. by Su Lin Blodgett et al. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 89–104. DOI: 10.18653/v1/2024.hcinlp-1.9. URL: https://aclanthology.org/2024.hcinlp-1.9/.

[112] Xiao Wang et al. “TextFlint: Unified Multilingual Robustness Evaluation Toolkit for Natural Language Processing”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Ed. by Heng Ji, Jong C. Park, and Rui Xia. Online: Association for Computational Linguistics, Aug. 2021, pp. 347–355. DOI: 10.18653/v1/2021.acl-demo.41. URL: https://aclanthology.org/2021.acl-demo.41/.

[113] James Wexler et al. “The What-If Tool: Interactive Probing of Machine Learning Models”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 56–65. DOI: 10.1109/TVCG.2019.2934619.

[114] Sarah Wiegreffe and Yuval Pinter. “Attention is not not Explanation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 11–20. DOI: 10.18653/v1/D19-1002. URL: https://aclanthology.org/D19-1002/.

[115] Greg Wilson et al. “Good enough practices in scientific computing”. In: *PLOS Computational Biology* 13.6 (June 2017), pp. 1–20. DOI: 10.1371/journal.pcbi.1005510. URL: https://doi.org/10.1371/journal.pcbi.1005510.

[116] Tongshuang Wu et al. “Errudite: Scalable, Reproducible, and Testable Error Analysis”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 747–763. DOI: 10.18653/v1/P19-1073. URL: https://aclanthology.org/P19-1073/.

[117] Wenhao Wu et al. “WeCheck: Strong Factual Consistency Checker via Weakly Supervised Learning”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 307–321. DOI: 10.18653/v1/2023.acl-long.18. URL: https://aclanthology.org/2023.acl-long.18/.

[118] Wei Xu et al. “Optimizing Statistical Machine Translation for Text Simplification”. In: *Transactions of the Association for Computational Linguistics* 4 (2016). Ed. by Lillian Lee, Mark Johnson, and Kristina Toutanova, pp. 401–415. DOI: 10.1162/tacl_a_00107. URL: https://aclanthology.org/Q16-1029/.

[119] Xuesong Zhai et al. “A Review of Artificial Intelligence (AI) in Education from 2010 to 2020”. In: *Complexity* 2021.1 (Jan. 2021). DOI: 10.1155/2021/8812542. URL: https://doi.org/10.1155/2021/8812542.

[120] Jiawei Zhang et al. “Manifold: A Model-Agnostic Framework for Interpretation and Diagnosis of Machine Learning Models”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2019), pp. 364–373. DOI: 10.1109/TVCG.2018.2864499.

[121] Tianyi Zhang et al. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL]. URL: https://arxiv.org/abs/1904.09675.

[122] Haiyan Zhao et al. “Explainability for Large Language Models: A Survey”. In: *ACM Trans. Intell. Syst. Technol.* 15.2 (Feb. 2024). ISSN: 2157-6904. DOI: 10.1145/3639372. URL: https://doi.org/10.1145/3639372.

A

Overview of Existing Tooling for Model Diagnosis

Tool / System	Primary Purpose	Interaction	Diagnosis	Compare	Use Stage	Reference
Custom Scripts	Exploration	Scripted	–	No	Exploration	–
Jupyter Notebook	Exploration	Scripted	–	No	Exploration	[52]
scikit-learn	Evaluation	Scripted	No	Yes	Validation	[75]
NLTK	NLP Analysis	Scripted	Implicit	No	Exploration	[13]
spaCy	NLP Analysis	Scripted	Implicit	No	Exploration	[32]
Gensim	Embedding Analysis	Scripted	No	No	Exploration	[78]
Captum	Attribution	Scripted / API	No	Yes	Debugging	[54]
BertViz	Attention Visualization	Scripted / UI	No	No	Debugging	[105]
exBERT	Model Visualization	UI	No	No	Exploration	[44]
InterpretML	Interpretability	Scripted / UI	No	Yes	Debugging	[69]
Inseq	Sequence Attribution	Scripted	No	Yes	Debugging	[87]
TextAttack	Robustness Testing	Scripted	No	Yes	Validation	[65]
CheckList	Behavioral Testing	Scripted	No	Yes	Validation	[81]
Robustness Gym	Evaluation	Scripted	No	Yes	Validation	[38]
NL-Augmenter	Data Augmentation	Scripted	No	Partial	Exploration	[28]
TextFlint	Data Augmentation	Scripted	No	Partial	Exploration	[112]
LIT	Error Analysis	UI	No	Yes	Debugging	[100]
What-If Tool	Model Introspection	UI	No	Yes	Debugging	[113]
Ferret	Interpretability	Scripted	No	No	Exploration	[8]
Ecco	Model Introspection	UI	No	No	Exploration	[3]
AllenNLP Interpret	Interpretability	UI	No	Partial	Debugging	[109]
Errudite	Error Analysis	UI	Partial	Yes	Debugging	[116]
Manifold	Interpretability	UI	No	Yes	Debugging	[120]
LM-Debugger	Model Debugging	UI	No	Yes	Debugging	[37]
InterroLang	Interpretability	UI	No	No	Exploration	[33]
LLMCheckup	Interpretability	UI	No	No	Exploration	[111]

Table A.1: Consolidated overview of representative tools and interfaces used to diagnose (undesirable) model behavior, characterized by their diagnostic affordances rather than intended purpose.

B

Overview of Toolkit System Requirements

This appendix contains the complete set of epics, feature- and task-level user stories, and functional/non-functional requirements used to specify the diagnostic toolkit.

Table B.1: User story hierarchy and correspondence to C4 architectural levels.

Tier	Type	Corresponds to C4 level
Epic	Very broad goals	C1 (Context)
Feature-level story	Key capabilities	C2 (Container)
Task-level story	Specific UI/service interactions	C3 (Component)

B.1. User Stories

The user stories for the diagnostic toolkit are organized into three tiers, as summarized in Table B.1. Each tier corresponds to a level of the C4 model, starting from high-level context (epics) down to specific component interactions (task-level stories).

B.1.1. Epics (C1)

- **[E1]** As a practitioner, I want to **diagnose** LLMs, so that I can understand their behavior.
- **[E2]** As a practitioner, I want to **share** my findings, so that I can discuss them effectively during (stakeholder) meetings.
- **[E3]** As a practitioner, I want to **configure** and **manage** my diagnostic environment, so that I can tailor diagnostics to my needs and ensure consistent operation.

B.1.2. Feature-level Stories (C2)

[E1] Diagnose

- **[F1.1]** Run **diagnostic tests** on large language models to identify performance issues and failure modes.
- **[F1.2]** Access and analyze **test results in detail** to investigate potential causes of failure.
- **[F1.3] Compare** results across different **test runs** or model versions to detect regressions or improvements.
- **[F1.4]** View the **activity history** of a test to trace changes and reruns over time.
- **[F1.5] Configure test** parameters and settings to control how diagnostics are applied to each model.

- **[F1.6]** Duplicate or **reuse previous test configurations** to quickly run variations on similar models.

[E2] Share

- **[F2.1]** Generate and review **reports** to document and communicate model behavior.
- **[F2.2] Export test results and reports** in standard formats to share them or integrate them into other workflows.

[E3] Configure & Manage

- **[F3.1] Manage models** and diagnostic **tests** to tailor the toolkit to different use cases.
- **[F3.2]** Integrate **external benchmarks** and datasets to run diagnostics on standardized inputs.
- **[F3.3]** Access and apply standardized **metrics** to validate results consistently across models and benchmarks.
- **[F3.4] Organize** diagnostic work into projects to manage and revisit analyses over time.
- **[F3.5]** Interact through a clear and responsive **interface** to focus on interpreting results rather than managing the tool.
- **[F3.6] Search** and filter tests, reports, and models to quickly find relevant diagnostic information.

B.1.3. Task-level Stories (C3)

[F1.1] Run Diagnostic Tests

- See the current status of tests to estimate duration.
- Manually start a test to control when diagnostics are executed.
- Cancel a running test to stop misconfigured or unnecessary jobs.
- View a queue of pending or scheduled tests to understand upcoming workload.
- Run multiple tests at the same time to accelerate the diagnostic process.

[F1.2] Show Test Results

- Inspect individual outputs to trace specific failure behaviors.
- View aggregated metric scores to assess overall model performance.
- View metric scores of individual outputs to trace item-level performance.
- View automatically detected symptoms or signs per test to identify areas of concern.
- Annotate outputs with additional symptoms or signs to document missing or overlooked issues.

[F1.3] Compare Tests

- Select two or more test runs to perform a direct comparison.
- View metric differences between selected tests to spot differences.

[F1.4] View Test Activity

- View a change-log of test modifications to trace evolution.
- Access historical test runs to reproduce or verify past results.

[F1.5] Configure Tests

- Select diagnostic methods when creating a test to tailor it to the problem.
- Edit test parameters (e.g., temperature, thresholds) to fine-tune diagnostics.
- Receive warnings about invalid/incomplete configurations before running to avoid failed or misleading diagnostics.

[F1.6] Reuse/Duplicate Tests

- Duplicate an existing test to use it as a template.
- Apply the same test setup to a different model to compare behavior across models.
- Bulk clone multiple tests to reuse setups across projects.

[F2.1] Generate Reports

- Select which tests to include to control report content.
- Include test comparisons to highlight differences across diagnostic runs.
- Preview a report before finalizing to ensure clarity and relevance.

[F2.2] Export Reports

- Export a report as PDF or HTML to share it easily.
- Export test results in JSON or CSV for use in other tools.
- Include visualizations in exported reports to better communicate findings.

[F3.1] Model & Test Control

- Upload models and datasets to work with relevant assets.
- Manage model metadata (e.g., name, size) to track versions.
- Archive or delete unused tests to keep the workspace clean.

[F3.2] Benchmark Access

- Browse available benchmarks to choose relevant diagnostic inputs.
- Preview benchmark examples to understand datasets before testing.

[F3.3] Metric Validation

- Browse available metrics to choose relevant evaluators.
- View how each metric is computed to interpret scores correctly.
- Adjust metric parameters to tailor evaluations to the use case.

[F3.4] Project Organization

- Create, rename, or delete projects to manage workspaces.
- Tag or categorize projects and tests to find them easily.
- View an overview of all tests and reports in a project for efficient navigation.

[F3.5] Interface Interaction

- See the current status of services to know whether diagnostics can be run.
- Report bugs to contribute to improving the toolkit and experience.

[F3.6] Search

- Search/filter projects and tests (e.g., by name, date, tag) to find relevant diagnostics quickly.
- Search/filter benchmarks and metrics (e.g., by language, domain, difficulty) to test under specific conditions.

B.2. Requirements

This section details the functional and non-functional requirements derived from the user stories outlined above.

B.2.1. Functional Requirements (FRs)

- **FR1: Diagnostic Test Execution**
  - **FR1.1** The system shall enable users to initiate and terminate diagnostic tests through the user interface.
  - **FR1.2** The system shall support the concurrent execution of multiple diagnostic tests.
  - **FR1.3** The system shall display the execution status and progress of each test.
  - **FR1.4** The system shall display a queue of scheduled or pending tests.
- **FR2: Test Output and Result Display**
  - **FR2.1** The system shall display individual model outputs for each test item.
  - **FR2.2** The system shall display metric scores at both the aggregated and per-output level.
  - **FR2.3** The system shall detect and display model symptoms and signs based on diagnostics.
  - **FR2.4** The system shall allow users to annotate outputs with custom tags or symptom labels.
  - **FR2.5** The system shall allow users to filter and sort test outputs based on metrics, symptoms, or signs.
- **FR3: Test Comparison**
  - **FR3.1** The system shall allow users to select and compare multiple diagnostic tests.
  - **FR3.2** The system shall display metric differences between selected tests.
  - **FR3.3** The system shall display symptom and tag differences between selected tests.
  - **FR3.4** The system shall allow users to define and save a test run as a baseline for future comparisons.
- **FR4: Test Configuration and Management**
  - **FR4.1** The system shall allow users to configure a test by selecting diagnostic methods and parameters.
  - **FR4.2** The system shall validate test configurations before execution and display warnings if invalid.
  - **FR4.3** The system shall allow users to duplicate individual or multiple test configurations.
  - **FR4.4** The system shall allow test configurations to be applied to different models when applicable.
  - **FR4.5** The system shall allow users to create and manage reusable test templates across projects.
- **FR5: Test Activity Logging**
  - **FR5.1** The system shall maintain a change-log of test configuration edits.
  - **FR5.2** The system shall allow users to access historical versions of test runs and their results.
- **FR6: Report Generation and Export**
  - **FR6.1** The system shall allow users to generate reports from selected tests.
  - **FR6.2** The system shall provide a preview of the report before final export.
  - **FR6.3** The system shall allow exporting reports in PDF and HTML format.
  - **FR6.4** The system shall allow exporting raw test results in JSON and CSV formats.
  - **FR6.5** The system shall include visual elements (e.g., charts, tables) in exported reports.

  - **FR6.6** The system shall allow users to configure which visualizations are included in each report.
- **FR7: Model and Dataset Handling**
  - **FR7.1** The system shall allow users to upload and manage model checkpoints and datasets.
  - **FR7.2** The system shall allow users to edit metadata associated with models and datasets.
  - **FR7.3** The system shall allow users to archive or delete test runs and configurations.
  - **FR7.4** The system shall allow the deletion of models and datasets imported by the user.
  - **FR7.5** The system shall validate uploaded models and datasets for compatibility with available diagnostic methods.
- **FR8: Benchmarks and Metrics**
  - **FR8.1** The system shall provide a browsable list of benchmark datasets.
  - **FR8.2** The system shall display sample data from benchmarks for preview.
  - **FR8.3** The system shall provide a browsable list of available metrics with descriptions.
  - **FR8.4** The system shall allow users to adjust metric-specific parameters (e.g., thresholds, weights).
  - **FR8.5** The system shall provide descriptions and associated literature for benchmarks and metrics.
  - **FR8.6** The system shall allow users to upload custom benchmark datasets.
  - **FR8.7** The system shall allow users to register and configure custom evaluation metrics.
- **FR9: Project and Workspace Management**
  - **FR9.1** The system shall allow users to create, rename, and delete projects.
  - **FR9.2** The system shall allow tagging of projects and tests for categorization and filtering.
  - **FR9.3** The system shall display a dashboard outlining all tests and reports per project.
  - **FR9.4** The system shall support duplicating project structures for reuse across diagnostic efforts.
- **FR10: Interface Support and Service Visibility**
  - **FR10.1** The system shall display the operational status of services (e.g., test runner, DB, model loading).
  - **FR10.2** The system shall notify users when a required service becomes unavailable or fails during test execution.
- **FR11: Search and Filtering**
  - **FR11.1** The system shall support search and filtering of tests, reports, and projects by metadata (e.g., name, tag, date).
  - **FR11.2** The system shall support search and filtering of benchmarks and metrics by properties (e.g., domain, language, difficulty).

B.2.2. Non-functional Requirements (NFRs)

- **Usability**
  - **NFR-U1** The system shall offer a consistent and intuitive user interface across all views and workflows.
  - **NFR-U2** The system shall support a basic test/example workflow that first-time users can complete without reading external documentation.
  - **NFR-U3** The interface shall respond to user actions (e.g., clicking, navigation) within 200 milliseconds.

  - **NFR-U4** Forms and configuration menus shall display validation feedback within 500 milliseconds.

- **Performance**
  - **NFR-P1** The system shall return a test result summary within X seconds for tests under 1000 samples.
  - **NFR-P2** The system shall support loading and rendering project dashboards with $<$2s latency for projects with $\leq$20 tests.
  - **NFR-P3** The system shall support parallel execution of at least 3 diagnostic tests without performance degradation.
- **Scalability**
  - **NFR-S1** The processor service shall support running on a remote machine without reconfiguration of the navigation service.
  - **NFR-S2** The system shall scale to support projects with up to 100 diagnostic tests and 10,000 total test outputs.
  - **NFR-S3** The system shall allow multiple diagnostic workers (e.g., containerized jobs) to process tests concurrently.
- **Reliability**
  - **NFR-R1** The system shall retry failed test executions up to 3 times before marking them as failed.
  - **NFR-R2** If a service becomes unavailable, the system shall display a clear status and prevent dependent actions.
  - **NFR-R3** Partial test results shall be recoverable after unplanned shutdowns.
- **Security & Privacy**
  - **NFR-SP1** Uploaded models and datasets shall remain local to the user's environment unless explicitly shared.
  - **NFR-SP2** Diagnostic results and logs shall not be exposed over the network unless using HTTPS.
  - **NFR-SP3** The system shall store practitioner annotations and tags securely in the database.
  - **NFR-SP4** Sensitive environment configuration (e.g., API keys, model access tokens) shall be stored in `.env` files and not checked into version control.
- **Maintainability**
  - **NFR-M1** The system shall use modular service design to allow updating either the navigation or processor service independently.
  - **NFR-M2** All models, datasets, and metrics shall be implemented as plug-in components to allow extension.
  - **NFR-M3** The system shall log test activity, system errors, and service communications to separate log channels.
- **Interoperability**
  - **NFR-I1** The system shall support importing models from Hugging Face using standard model identifiers.
  - **NFR-I2** The system shall support benchmark datasets from Hugging Face.
  - **NFR-I3** The system shall export test results in JSON and CSV formats for integration with external tools.
  - **NFR-I4** The system shall export reports in HTML and PDF formats.
- **Portability**

  - **NFR-PT1** The entire toolkit shall be deployable via Docker Compose on Unix, macOS, and Windows-based systems.
  - **NFR-PT2** The system shall operate in both local-only and remote-service (split-host) configurations with a single parameter change.
- **Accessibility**
  - **NFR-A1** All visual elements (e.g., charts, tables) shall include alternative text or descriptions.

C

C4 Architectural Diagrams

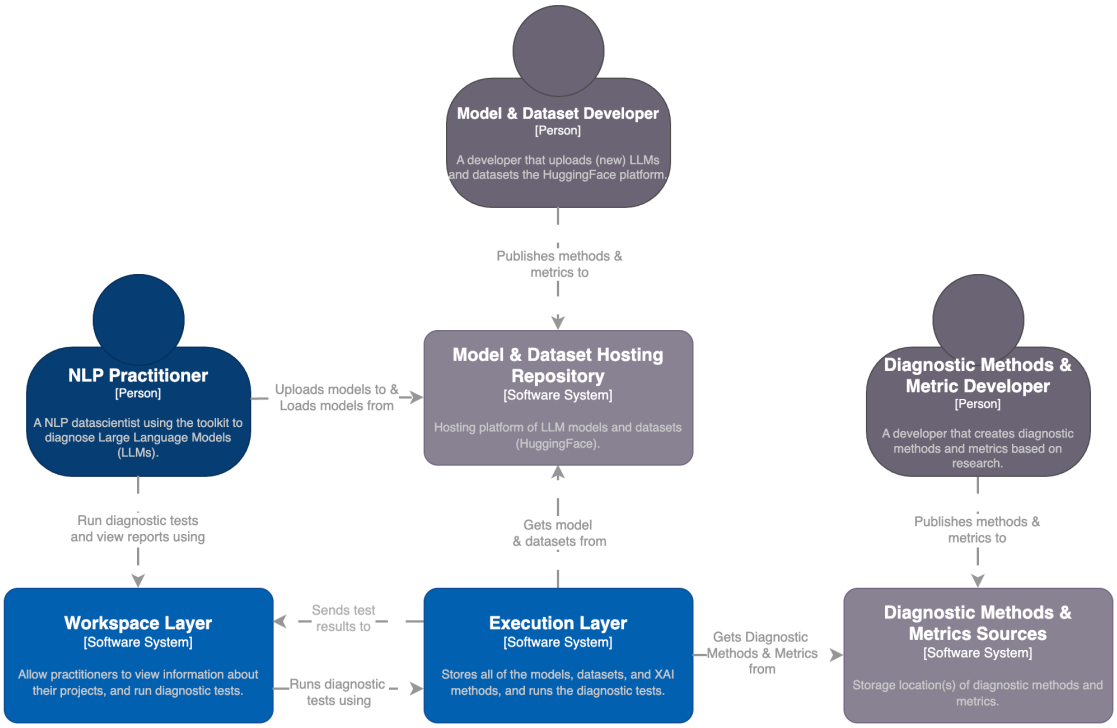

Figure C.1: C4 context diagram of the diagnostic toolkit. The NLP practitioner interacts with the toolkit to perform diagnosis and can manipulate models and datasets as needed through Hugging Face.

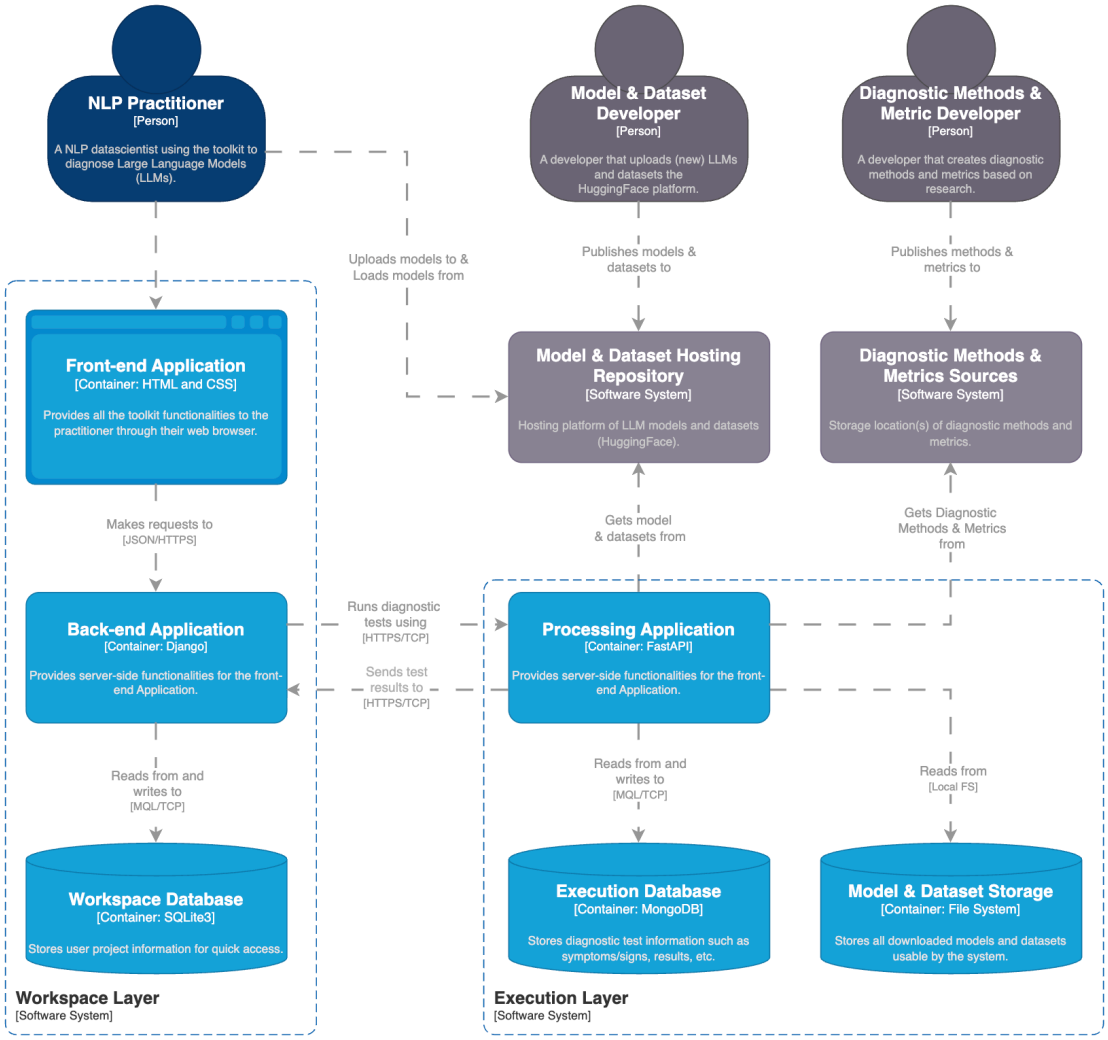

Figure C.2: C4 container diagram of the diagnostic toolkit. The navigation layer manages projects and diagnostic artifacts, while the processing layer executes inference and diagnostic methods within resource constraints.

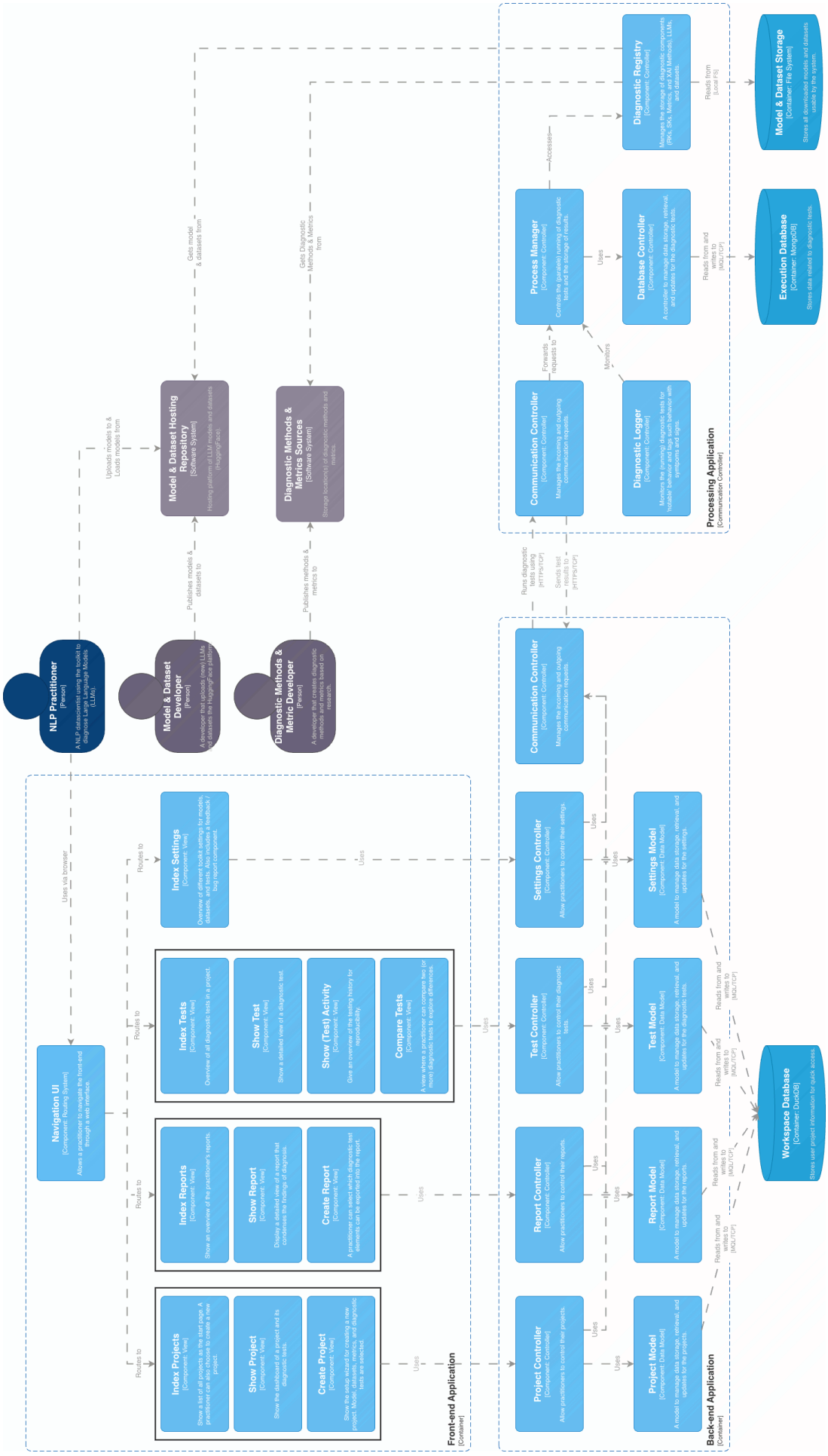

Figure C.3: C4 component diagram of the diagnostic toolkit. The navigation layer includes components for project management and report generation, while the processing layer contains components for model inference, diagnostic methods, and resource management.

D

Figma Front-end Designs

This appendix provides selected high-fidelity user interface mock-ups created in Figma prior to implementation. The mock-ups were used to translate task-level user stories into concrete interaction flows (e.g., project setup, test configuration, result inspection, and comparison), and to validate information layout and navigation before development (Chapter 4).

The full set of designs is available via the project SharePoint link (access restricted to authorized TU Delft accounts): `http://bit.ly/3LrLEJB`

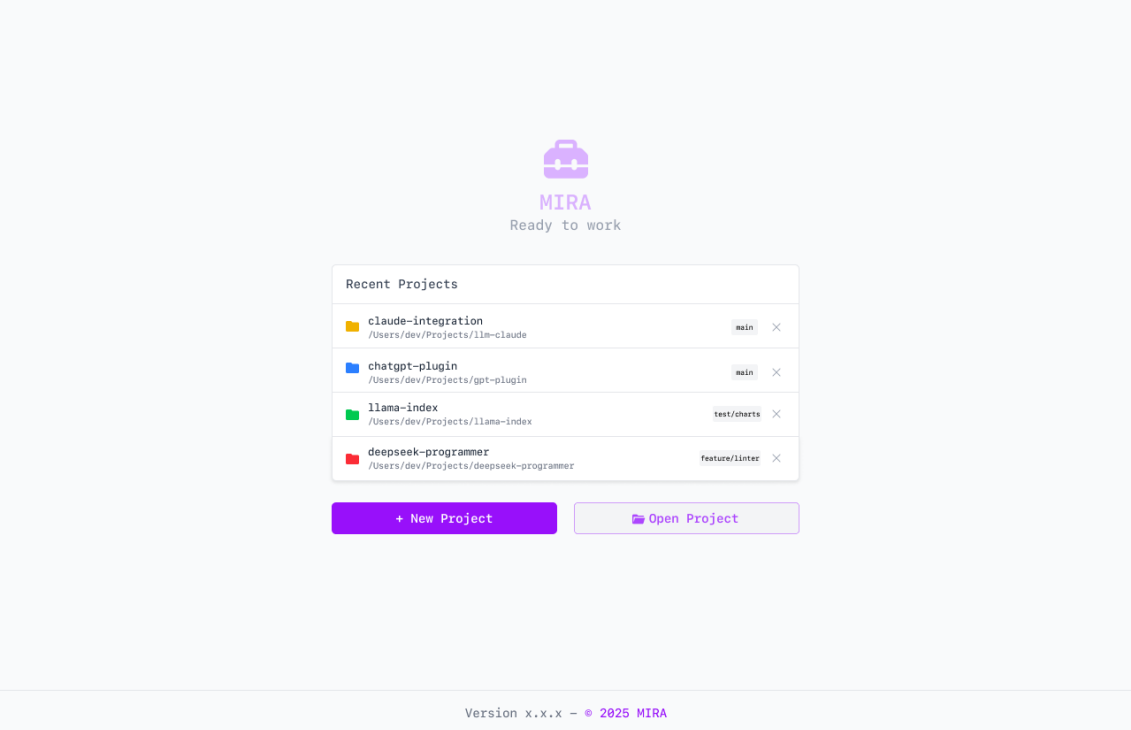

Figure D.1: Initial project selection screen. The view is populated with existing projects that can be opened, and provides an entry point for creating a new project.

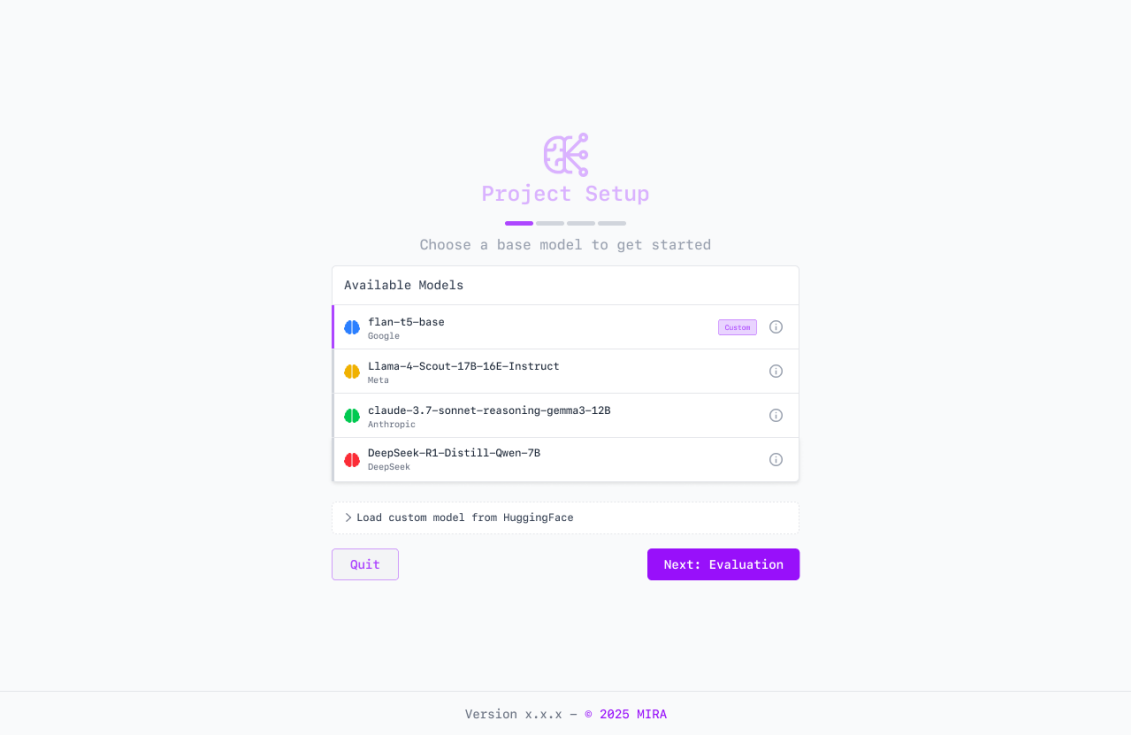

Figure D.2: Setup wizard (model selection). A custom Hugging Face model has been loaded and selected (shown here: `flan-t5-base`).

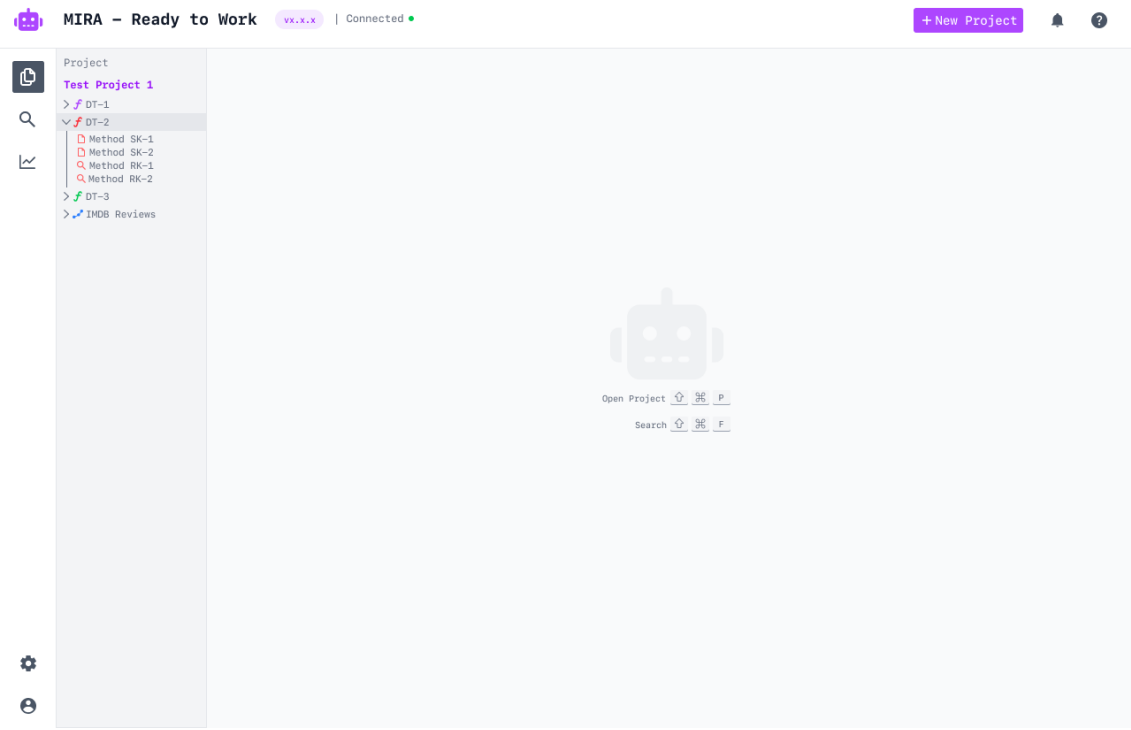

Figure D.3: Toolkit dashboard with the test navigation panel expanded, illustrating how users browse and manage tests within a project.

E

Overview of Considered Diagnostic Methods

The following methods were actively attempted but could not be integrated into the containerized toolkit due to technical incompatibilities. In both cases, the core problem was that the method's codebase was developed against library versions no longer maintained or incompatible with the dependency constraints shared by other components of the execution environment. Custom patches and local library forks were attempted, but did not resolve the conflicts.

Sent-Bias [59]. Sent-Bias implements the Sentence Encoder Association Test (SEAT), an extension of the Word Embedding Association Test (WEAT) [21] to sentence-level representations. It measures the degree to which sentence encoders associate social groups (e.g., gender, race, or occupation) with positive or negative attributes, using sets of target and attribute sentences as stimuli. Within the diagnostic framework, Sent-Bias would serve an RK-based role: it probes the associations the encoder has internalized by surfacing differential reliance on social-group surface features, making it a complement to HolisticBias for diagnosing stereotyping or harmful association failure patterns. *Reason for non-integration:* The repository targets Python 3.6 and PyTorch 0.4.1, with additional dependencies on early versions of `allennlp`, `tensorflow-hub`, and `pytorch-pretrained-bert`. These version constraints are incompatible with the Python 3.10+ and PyTorch 2.x environment required by other toolkit components. No maintained fork offering updated compatibility was identified. Attempts to patch the relevant modules locally were unsuccessful due to breaking API changes in the downstream libraries.

CC-SHAP [74]. CC-SHAP (Comparative Consistency SHAP) is a fine-grained self-consistency measure for natural language explanations in LLMs. It extends SHAP to compare a model's input attribution profile when predicting an answer against its input attribution profile when generating an explanation for that answer. Divergence between the two profiles is interpreted as evidence of self-inconsistency: the model relies on different input features when explaining than when predicting. This provides an RK-based signal about explanation faithfulness that goes beyond standard post-hoc attribution methods. It is particularly relevant to diagnosing over-reliance on spurious features and to evaluating whether model-generated explanations are grounded in the model's actual reasoning. *Reason for non-integration:* The CC-SHAP implementation bundles a modified fork of the `SHAP` library with custom extensions that must be compiled against a specific version of `SHAP` internals. The fork is not actively maintained and is incompatible with the latest `SHAP` versions. On top of this, the libraries are provided as a conda environment with specific versions that conflict with the toolkit's containerized environment. Attempts to compile the custom extensions against a parallel installation and isolate the CC-SHAP execution environment were unsuccessful.

F

Design Briefs

This appendix contains the design briefs presented to participants during the user study. Each brief was designed to provide a fictional yet believable diagnostic scenario, grounding the evaluation tasks in a concrete operational context without requiring participants to provide or configure their own models or datasets. Both briefs describe the same overarching product context (an academic search system for university students) and were differentiated only by the specific NLP component assigned to the participant. The briefs were provided to participants at the start of the hands-on think-aloud portion of the session (see Chapter 5) and served as the sole framing document for the diagnostic task.

Summarization Brief

The summarization brief was assigned to participants P1, P3, P5, and P7. It instructed participants to act as an ML engineer tasked with evaluating and diagnosing the summarization component of the academic search system. The component was described as using an encoder-decoder model (`LongT5`) to produce concise summaries of scientific papers for student display. The brief provided context on the intended users, the expected behavior of the component, and the evaluation dataset (scientific papers curated from the ACL ontology).

Design Brief

Context

You are part of a project tasked with creating a simple search system for academic publications that university students will use. The objective is to provide students with articles relevant to their query (e.g., a set of keywords) and produce simple summaries of papers to be displayed alongside other information, e.g., author list, venue, etc. For this project, different teams are allocated to create different components: a retriever, a text summariser, and a text simplifier.

Task: Summarisation

You are assigned to the team in charge of building the summarisation component. Given your experience, you settle on using a fine-tuned version of `LongT5` (https://huggingface.co/pszemraj/long-t5-tglobal-base-16384-book-summary), an encoder-decoder language model capable of summarising long texts effectively.

You are now about to carry out an in-depth analysis to ensure that this model is not only meeting the performance requirements but that it is behaving correctly (e.g., the summarisation is faithful to the original content and the query). What would be your process to do this? What techniques would you use, and why? Which datasets would you use, and why?

To conduct your analysis, you have with access to this model, some datasets, a testing suite, and the Huggingface repository to download additional resources you deem useful.

Figure F.1: Design brief for the summarization diagnostic task.

Simplification Brief

The simplification brief was assigned to participants P2, P4, P6, and P8. It instructed participants to act as an ML engineer tasked with evaluating and diagnosing the simplification component of the academic search system. The component was described as using a decoder-only instruction-tuned model (`Qwen2-1.5B-Instruct`) to rewrite paper abstracts into plain language accessible to non-expert students. As with the summarization brief, the context on intended users, expected behavior, and the evaluation dataset was provided.

Design Brief

Context

You are part of a project tasked with creating a simple search system for academic publications that university students will use. The objective is to provide students with articles relevant to their query (e.g., a set of keywords) and produce simple summaries of papers to be displayed alongside other information, e.g., author list, venue, etc. For this project, different teams are allocated to create different components: a retriever, a text summariser, and a text simplifier.

Task: Simplification

You are assigned to the team in charge of building the simplification component. Given your experience, you settle on using `Qwen2-1.5B-Instruct` (https://huggingface.co/Qwen/Qwen2-1.5B-Instruct), a well-known decoder-only LLM.

You are now about to carry out an in-depth analysis to ensure that this model is not only meeting the performance requirements but that it is behaving correctly (e.g., no non-factual information is introduced during the simplification process). What would be your process to do this? What techniques would you use, and why? Which datasets would you use, and why?

To conduct your analysis, you have with access to this model, some datasets, a testing suite, and the Huggingface repository to download additional resources you deem useful.

Figure F.2: Design brief for the simplification diagnostic task.

G

Interview Protocol

This appendix contains the full interview protocol used during the evaluation study described in Chapter 5. Each session lasted approximately 60 minutes and followed a semi-structured format combining a think-aloud hands-on task with a reflective discussion. The protocol was designed to elicit both observable diagnostic behavior during toolkit interaction and participants' reflective accounts of how the toolkit compared to their existing diagnostic practices. The protocol is reproduced across two pages below.

Interview Protocol

>> Note that (1) the duration of each individual part of the session and (2) the wording of the questions might be subject to minor changes after the initial 1-2 interviews. <<

Session Guide

A. Admin Points

3 MINUTES

1. Introduction to the project
2. Explain the structure of the session
3. Double-check informed consent
4. Start recording if everything is ok

B. Participant Background

2 MINUTES

1. Years of experience in the current role
2. Background in computer science/engineering or something else?
3. Do you have experience with LLM testing and/or diagnosis? If so, can you briefly explain what you do?

C. Hands-on with the Toolkit

30 MINUTES

1. Give and explain the design brief (next pages)
2. Video and audio record what is happening (screen, audio, not their face)
3. Note down moments of difficulty
4. Make their thought process emerge while they use the toolkit

D. Questionnaires

10 MINUTES

1. NASA Task Load Index
2. User Experience Questionnaire

E. Open Discussion and Reflection

15 MINUTES

1. Factors
    1. How helpful was the toolkit to diagnose LLMs?
    2. Would your process to test/diagnose LLMs potentially be different if you had access to something like our toolkit?
    3. What did you find the most helpful in the toolkit, and why?
    4. Instead, what did you find the least helpful, and why?
2. Problem Localisation Efficiency
    1. How challenging was it to diagnose the LLMs, and why?
    2. How quickly do you feel you were able to diagnose issues in LLMs, and why?
    3. Were the techniques included in the toolkit expressive enough to diagnose LLMs, and why?

Figure G.1: Interview protocol, page 1 of 2. Covers the session introduction, informed consent, participant background elicitation, instructions for the think-aloud hands-on task, and points one and two of the discussion and reflection phase.

3. Preferences
    1. Which method (or subset of methods) did you prefer using? Datasets, methods, or XAI?
    2. Is there anything missing?

F. Closing

1. Is there anything you would like to add or ask?
2. Is there anything that I should have asked you but did not?

Figure G.2: Interview protocol, page 2 of 2. Covers point three of the discussion and reflection phase, as well as the session close.