Chapter 5 : Evaluating Agents : from challenge to opportunities

Evaluation in AI has always lived inside a paradox. It is central to every system we build and silently shapes the entire research ecosystem, yet almost nobody wants to do it. Part of this comes from a mismatch of timelines. Research is supposed to operate far ahead, exploring which directions are worth following, while industry focuses on delivering value. In practice, this means demonstrating that something can be built at all (a Proof of Concept) before refining, optimizing and putting it into production.

Generative AI has collapsed this timeline. As Andrew Ng noted, we moved from a world where a data science team needed six months to collect data, train a model and ship the first version, to one where an LLM can produce a functional (albeit imperfect) draft of a feature within days. This dramatic acceleration creates a dangerous illusion: if an LLM produces something plausible, one might believe the product is already reliable. But an agent does not output truth; it outputs a *probable* solution. And probability is not engineering.

We are therefore facing a genuine paradigm shift: we no longer evaluate a model, but a system. A system with compounding stages, cascading decisions and multiple chances to fail. As Chip Huyen[1] points out, agents amplify two structural vulnerabilities of LLMs. Compound mistakes accumulate at every step: if a model is 95% accurate per step, ten steps bring you down to roughly 60%, and a hundred steps reduce reliability to microscopic levels. And the stakes are higher: an agent equipped with tools can take actions whose consequences propagate far beyond text generation.

So why do we care more than ever about evaluation? We might borrow from Socrates, who declared during his trial in *Apology* 38a: **“The unexamined life is not worth living for a human being.”** The line is often quoted vaguely, but its force lies in the idea that without deliberate reflection, we cannot claim to act responsibly or even understand what we are doing. « **an unexamined agent is not worth using** » In our context, it reminds us that an unexamined agent is not worth deploying. An AI system that acts without scrutiny, without introspection, without a clear understanding of its own behavior, is not merely incomplete: it is potentially harmful. The analogy works almost too well : agents, like humans, require examination not as a luxury but as a condition for trust and meaningful action. The risks are real. A bad reflection step might produce operational damage, think of an automated report that confidently misattributes financial responsibility or misclassifies a legal clause. And beyond immediate harm, there is another danger: erosion of trust. Trust in the software, trust in the company deploying it, and eventually trust in the entire technology sector. Agentic AI runs the risk of creating its own bubble if we rush forward without the discipline to examine how these systems behave.

As you know, every NLP task must be evaluated: to track performance during implementation, to tune parameters in the development phase, and to situate our approach within the broader community by comparing it to baselines and competing systems. At first glance, evaluating agents appears to be the logical continuation of evaluating NLP or LLM models. But as soon as we look more closely, we see that the analogy breaks down. We are no longer evaluating a model; we are evaluating a *system*.

---

1 https://huyenchip.com/2025/01/07/agents.html

This should feel familiar to anyone who has worked on classical NLP pipelines—information extraction, for instance, where entity detection feeds relation extraction, and each stage amplifies or mitigates the errors of the previous one. But agents introduce an additional layer of instability: the lack of determinism inherent to LLMs, the emergence of entirely new tasks that have no established metrics, the inadequacy of inexpensive automatic measures, and, more broadly, the absence of accepted metrics altogether.

With increased complexity comes increased opportunity for failure. A single faulty tool call can corrupt the entire trajectory. A poorly handled loop can send the agent into an infinite drift. Reflection failures can produce false task completion, where the system confidently asserts that the work is done when it is not. Even the instructions can drift, slowly bending the agent's behaviour away from the user's intent. These are not exotic edge cases; they are structural weaknesses of agentic systems. This raises an important question: what exactly is the goal of evaluating an agent? Is it to measure the general abilities of the underlying LLM, or to assess how a specific system behaves on a precise task under realistic constraints? When evaluating agents, it is not enough to judge the quality of the final output. We must also examine the *path* that produced it. The sequence of decisions, reflections, tool calls, and corrections is itself an object of evaluation. In many ways, the agent's reasoning becomes an optimization problem: not merely "did it succeed?" but "did it succeed efficiently, safely, and coherently?"

The difficulty of evaluating AI agents is therefore not the simple technical prolongation of evaluating models, nor is it merely an open research question awaiting a clever metric. Non-determinism challenges our previous quality assurance protocols at a fundamental level. The very way we build and engineer systems is being put under pressure. This is, quite literally, where it hurts: agents force us to confront the fact that we can no longer rely on fixed pipelines, stable intermediate representations, or reproducible execution traces. Everything moves. Everything interacts. And everything must be examined.

If we try to situate agent evaluation within the broader history of evaluation in computing, what emerges is a slow but unmistakable progression: at each technological step, evaluation becomes a little more ambiguous, a little more fragile, and a little more demanding. What follows is not merely a comparison, but a chronological deepening of difficulty.

Software is the simplest case. Failure is explicit, obvious, traceable, and reproducible. A bug manifests as a crash, an exception, a wrong output that can be deterministically triggered. One can bisect, log, debug, and guarantee that the same inputs produce the same failure every time. Evaluation here is a matter of engineering discipline.

Machine Learning complicates the picture. Models become black boxes. Evaluation is no longer trivial: each task requires a carefully chosen metric, and different metrics capture different aspects of performance. Yet despite that complexity, ML still lives in a world where evaluation is comparable and reproducible. A trained model on a fixed dataset yields a fixed score. Even if we do not understand the internals, we can measure the output reliably.

LLMs extend the difficulty further. They inherit all the weaknesses of ML—algorithmic biases, factual hallucinations, performance drift when data distribution changes, emergent behaviours where the model tries to cheat objectives or exploit loopholes. Evaluation cannot be reduced to

metrics anymore and increasingly requires human judgement. LLMs blur the border between correctness and plausibility, between capability and illusion.

Agents, finally, represent a qualitative jump. Their failures are insidious. The model still outputs a final answer, but it may be entirely wrong without any crash at runtime. The LLM is no longer just a text generator; it becomes a task planner, a decision-maker, a coordinator of tools. Non-determinism operates not only at each step but in the *path* taken: two runs of the same agent may follow different trajectories, call different tools, and take different decisions. This compounds unpredictability at every stage. Interactions with external systems add a dynamic layer that breaks reproducibility. Memory, context accumulation, and stateful reasoning ensure that each run is, in a sense, unique. What we see is an evolution where every new advance adds a layer of complexity: By the time we reach agents, evaluation becomes an optimization problem where we are not even sure what the objective function is, or how to measure it.

This is the true paradigm shift: we have moved from evaluating a model to evaluating a system. Where model evaluation relied primarily on qualitative and task-specific metrics, system evaluation must now integrate pragmatic engineering concerns :efficiency, robustness, safety, controllability. These concepts existed before, but they were not the bottleneck. Quality was. With agents, the bottleneck becomes everything else: the path, the interactions, the stability of decisions, the reliability of tools, the shape of the reasoning process itself. To better describe this paradigme shift, Google in a very recent white paper about Agent Quality[2] hilighted a trinity of principles everyone should follow making evaluation a key component of the agent design, and not a second hand add on :

* The Trajectory is the Truth
* Obervability is the Foundation
* Evaluation is a Continuous Loop, the Agent Quality Flywheel

In this chapter, we'll try to decode these esoteric formula and study how we can reliably observe and evaluate agents today, regardless of their complexity or architecture.

Toward a Highly Experimental Field, or Toward New Ways to Train and Optimize?

Agent evaluation therefore sits at an odd frontier. It is not only a matter of measuring performance but of *calibrating* behaviour, of steering systems whose internal dynamics matter as much as the output. In many ways, evaluation is becoming the new fine-tuning: instead of adjusting weights, we refine workflows and reasoning paths through iteration and observation. This raises a final question that will guide the rest of the chapter: are we entering a highly experimental field, where agents must be probed like scientific systems, or inventing a new way to train and optimize AI altogether? The real challenge will be learning how to balance these competing objectives without a single metric to unify them.

2 https://drive.google.com/file/d/1EnTSGztSrjooYMLaDe8EnoATfsSoe3xv/view

I) What are we evaluating exactly ?

The evaluation of agents is beginning to attract significant attention in the academic community, yet the resulting efforts often diverge widely: some focus on probing the agentic abilities of LLMs, others attempt to benchmark “general intelligence,” while industry practitioners urgently need evaluations that reflect concrete tools, real environments, and immediate deployment challenges. This fragmentation creates a risk of talking past one another. Before designing any benchmark or metric, one must first be explicit about *what* is being evaluated and tailor the protocol to that target. Evaluating agents begins with a deceptively simple question: *what exactly are we evaluating?* Are we measuring the underlying LLM’s agentic capabilities, its general intelligence, or the performance of a specific agent operating in a specific environment? In practice, these layers blur into one another. SAs soon as the system interacts with tools, environments or users, a third dimension appears: we are no longer evaluating isolated abilities but the behaviour of the agent as a whole. This shift forces us to rethink benchmarks, metrics, and even what “performance” means in a world where tasks are open-ended, trajectories matter as much as outcomes, and the very notion of reproducibility is up for negotiation.

a) Evaluating LLM’s agentic capabilities

TODO add sentence to see that we must ensure that the engines at the heart of agents are adapted to the architecture they are put in.

Evaluating the agentic abilities of LLMs is the most natural place to start, because it sits closest to standard LLM benchmarking while already escaping its boundaries. There is, of course, some overlap: many tasks still probe reasoning, factuality, or instruction following. But as soon as an LLM is placed inside an agentic frame, its applicability becomes much broader and requires new evaluation methodologies, new benchmarks, new environments, new metrics, and sometimes new philosophies of what “performance” even means. The goal here is not to evaluate the final agent as a system, nor to test anything resembling general intelligence. It is to measure how well an LLM can *become* an agent. In other words, we want to know how effectively a pretrained model can be integrated into an agentic architecture and whether its internal capabilities are sufficient to sustain multi-step behaviour.

Reasoning and planning

This begins with reasoning benchmarks, many inherited from the LLM era: HotpotQA, StrategyQA, ARC. But new benchmarks such as FlowBench or PlanBench explicitly target planning and multi-step decomposition. Some were even designed for the ReAct framework and assume the agent will alternate between reasoning and acting.

Yet classical reasoning is not enough. The emergence of agentic workflows has created new abilities that must be tested: the capacity to recover from errors, to plan when the environment is interactive and not static, to decompose tasks effectively given the available tools, to track state over time, and

even to engage in meta-planning—reflecting on which path is best before taking the first step. None of these existed in the pre-agentic era of LLM evaluation.

Correct tool use

Another crucial axis is tool use. Here, the challenge is deceptively subtle: it not only concerns selecting the right tool at the right moment, but also generating the correct arguments, understanding when *not* to use a tool, and producing a well-grounded answer from the tool's output. This ties agent evaluation to long-standing concerns in RAG: factual grounding, hallucination reduction, and extraction quality.

Existing benchmarks like ToolAlpaca, APIBench, ToolBench, or the Berkeley Function Calling Leaderboard usually evaluate this in a single-round setting. Newer versions introduce multi-turn scenarios that inevitably overlap with planning itself. ComplexFunBench goes further and makes tool use dynamic, forcing the model to adapt arguments and sequences to the evolving environment.

Self-reflection

Self-reflection, often associated with prompting strategies like Reflexion or chain-of-verification, has also become a target of evaluation. The methodology is straightforward: take an existing benchmark, add a feedback loop, and measure whether the model solves more items once it is allowed to reflect on its mistakes. The question is no longer "can the model answer correctly?" but "can it improve itself in the middle of the task?"

Evaluating individual components: memory as a case study

Beyond the LLM itself, certain components integral to agent behavior—especially memory—require dedicated evaluation. This is much harder than it seems, because memory architectures differ widely from one implementation to another. As a consequence, evaluating memory often combines two strategies: measuring extrinsic performance improvements on standard tasks such as HotpotQA, or using specialized, targeted benchmarks like LocoMO that explicitly probe the agent's ability to recall, update, or use past information.

Memory evaluation is a perfect illustration of the shift in paradigm: the goal is no longer to test the linguistic prowess of the model, but the functionality of a subsystem whose behavior only emerges during multi-step interactions.

TODO ccl

b) Evaluating domain specific agents

Once we leave the realm of pure LLM capabilities, the nature of evaluation changes entirely. We are no longer interested in how well the model reasons in the abstract, but in how the *agent behaves as a whole* within a concrete scenario. The goal is to measure performance in application-specific, real-life stakes settings, where tools, external knowledge bases, and interaction patterns matter as much as the underlying LLM.

In classical NLP, evaluating a model required only two ingredients: a dataset that defines what the system is supposed to achieve, and a metric—reference-based or reference-free—that scores its

output. This was sufficient as long as the task itself was static. For agents, however, a third ingredient becomes indispensable: *an environment.* The operating environment may be simulated or real, static or dynamic, simple or tool-rich. It may incorporate user simulators, APIs, databases, constraints, and policies the agent must respect. As soon as the environment enters the picture, evaluation becomes an order of magnitude harder.

This explains why most existing efforts in this direction target general-purpose agents for which a generic set of tasks and tools makes sense: web navigation (WebAgents), software engineering agents, scientific agents for literature retrieval or domain-specific instruction following, conversational agents evaluated on their ability to reach a target state and communicate the correct answer. In all these cases, the benchmark designer must invent *application scenarios*, each one an attempt to approximate a slice of real life.

Designing such benchmarks is an art of compromise. They cannot be too specific, or they end up evaluating a single tool integration; but they cannot be too general, or the task becomes trivially solvable or hopelessly abstract. The difficulties are well known: finding the right level of complexity, collecting realistic scenarios in sufficient quantity, selecting tasks that an agent can reasonably attempt but cannot trivially solve, and constructing an environment that is rich enough to matter without becoming so heavy that it collapses under its own weight. τ-Bench, for example, emulates dynamic conversations between an agent and an LLM-simulated user in customer service domains. Each domain comes with databases, APIs, and a policy that constrains behaviour—a miniature world carefully crafted for evaluation.

For software engineering, the shift is even clearer. It is no longer enough to evaluate algorithmic reasoning on toy problems. SWE-bench, built from real GitHub issues, provides full repositories, execution environments, and unit tests, bringing together every component of a “proper” benchmark: realistic tasks, a concrete environment, and a golden solution that can objectively validate success. SWE-Lancer pushes this further by evaluating agents on real freelance coding tasks, forcing them to operate closer to the messy conditions of actual production work.

At this point, one may wonder: how is this different from domain adaptation in classical NLP? After all, adapting a PLM to biomedical or legal text also involves domain-specific constraints. The difference is structural. Domain adaptation in NLP typically means adjusting the model—via fine-tuning or knowledge injection—so that it performs as well on specialized data as it does on generic data. The task stays the same; only the distribution changes. For agents, domain adaptation means adapting *the system*: new tools, new knowledge bases, new interaction protocols, and sometimes even new architectural components. The environment itself becomes part of the task definition, and the evaluation must reflect that.

Evaluating specialized agents therefore marks a decisive departure from model evaluation. It is not enough to ask whether the agent produces the right answer; we must test whether it can navigate an environment, manipulate tools, respect domain constraints, recover from failed attempts, and converge toward a goal in conditions that approximate real life. In other words, the benchmark is no longer a dataset—it is a world.

c) Benchmarking General Intelligence : towards new Turing tests ?

At the far end of the evaluation spectrum lie the attempts to benchmark *general* agents, benchmarks that aspire, implicitly or explicitly, to approximate a new kind of Turing test. These efforts are often presented as a natural extension of standard LLM evaluation: rather than measuring linguistic quality or factual recall, they seek to probe higher-order abilities such as multi-step reasoning, interactive problem-solving, and tool-mediated action. In other words, they claim to evaluate something approaching "intelligence," or at least a version of it that goes beyond the linguistic capabilities we have already mapped : has general intelligence emerged from the scaling law ?

Yet this ambition immediately runs into conceptual trouble. Evaluating general agents presupposes that we agree on what general intelligence is, a notion that remains contentious even in cognitive science. And there is a practical issue: no benchmark evaluates "intelligence" in the abstract. All of them operate within *systems*: tools, environments, APIs, simulated users, domain constraints. What we end up evaluating is not an open-ended intelligence but, more modestly, the performance of an artificial assistant operating at the level of an entry-level knowledge worker in a software-rich environment. The rhetoric gestures toward AGI, but the reality is closer to office automation under complex conditions. This tension is amplified by the challenges inherited from LLM evaluation: the incessant push to "go beyond" whatever capability was measured yesterday, and the chronic obsolescence of benchmarks. As the contextual.ai essay puts it, AI is bottlenecked by proper benchmarking; we cannot fix a limitation we have not clearly diagnosed. Benchmarks saturate quickly, then collapse, forcing the field to perpetually seek new ones. Agents only accelerate this cycle, because every extension of LLM abilities,planning, tool use, web navigation, creates new tasks and pushes the state of the art into uncharted territory.

GAIA: real-world questions and real-world action

GAIA (2023) represents one of the most ambitious attempts to address this. It proposes real-world questions requiring classical reasoning, multimodal perception, web browsing, and tool-use proficiency. Human respondents achieve around 92% accuracy; GPT-4 with plugins scored around 15%. Later systems dramatically improved—up to 87% for ZTE AICloud using Gemini Pro 2.5—but the gap between humans and models remains significant.

GAIA has several defining features. It is a short-answer benchmark, with questions that are unambiguous and verifiable. It embraces the fact that tasks that are hard for humans (writing essays, generating long text) are trivial for LLMs, and instead focuses on conceptually simple tasks requiring precise execution of complex sequences of actions in the real web environment. The emphasis is on the ability to chain actions: navigating websites, extracting information, combining modalities, and adapting to a dynamic environment. Annotators categorize questions into difficulty levels based on how many tools or actions were needed to arrive at the correct answer. This makes GAIA particularly relevant for evaluating the action-chaining behaviour of agents.

However, limitations remain. GAIA implicitly evaluates the reasoning engine and the browsing capability of the agent, but not a large or diverse toolset. Furthermore, the classic problem of data contamination looms: should the benchmark release both questions and answers (leading to trivial memorization), only questions (encouraging over-engineering), or nothing (turning it into a black

box)? Each option compromises either transparency, scientific validity, or fairness. Despite these difficulties, GAIA's designers propose its use as an early proxy for "t-AGI"—a bold but debatable claim.

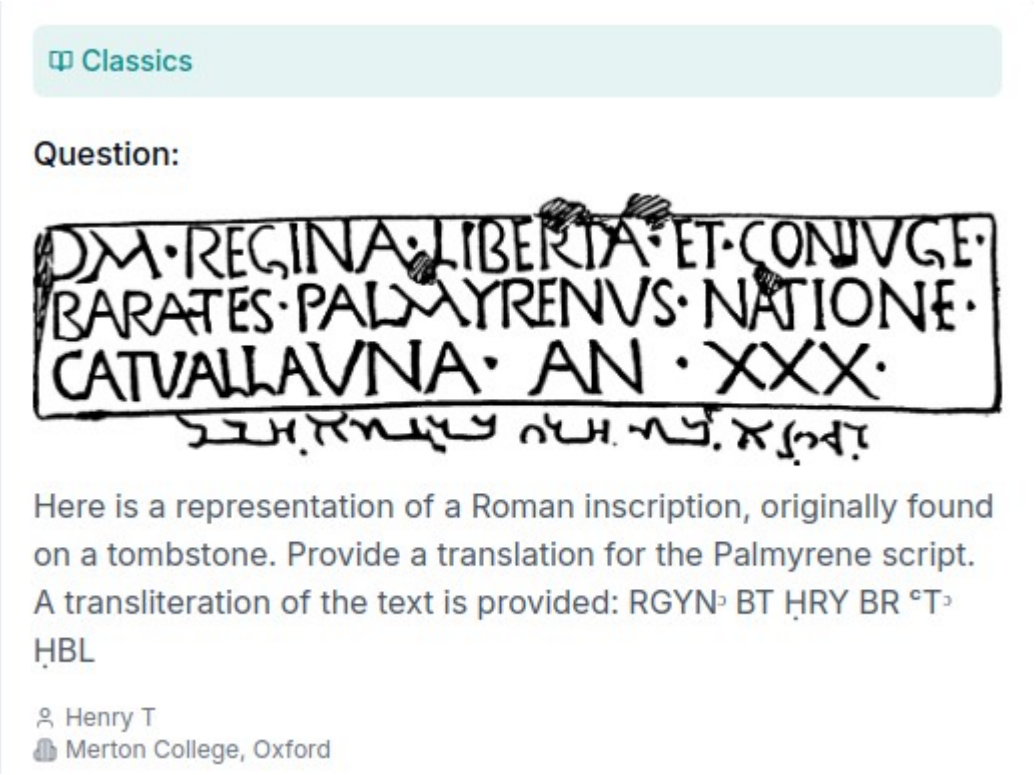
AgentBench, Galileo and the risks of closed environments

Other benchmarks, such as AgentBench or more recent "agentic evaluation" frameworks like Galileo, adopt closed environments: small, fully specified worlds with limited tools and predictable dynamics. This makes evaluation easier but introduces an obvious risk: models might learn the specific APIs or patterns of the benchmark rather than demonstrating genuine, transferable ability. The result is an evaluation of *environment-specialization*, not general agentic competence.

Humanity's Last Exam

Humanity's Last Exam (HLE) positions itself as the last great closed-ended academic benchmark. With over 2,500 multimodal questions at the frontier of human knowledge, it aims for extreme breadth and difficulty. Its creators anticipate that models may surpass 50% accuracy by the end of 2025. However, even such high scores would demonstrate only expert-level mastery of verifiable, closed tasks; they would not suggest autonomous research ability or anything resembling AGI.

HLE reveals another fundamental limitation: designing closed-ended, high-difficulty questions in a world where knowledge is open-ended and evolving is an annotation challenge of its own. In frontier domains, even determining the *correct* answer may require expert debate. This raises deep questions about the role of question design in evaluating agents: is there a type of question LLMs are inherently better suited to answer? And conversely, are there cognitive niches where humans retain structural advantage?



As benchmarks proliferate, they become harder to compare. The Holistic Agent Leaderboard (HAL) is a recent attempt to consolidate multiple evaluation settings and criteria under a single platform. This reflects an emerging recognition: evaluating general agents requires unifying perspectives across reasoning, planning, tool use, retrieval, environment navigation, and long-horizon trajectories. If LLM benchmarking taught us anything, it is that capabilities grow faster than benchmarks can capture them. We may well be witnessing the same phenomenon at the system level. Static benchmarks will saturate; dynamic, environment-rich evaluations will drift; trajectories will become as important as

outcomes. The field may move toward *live* benchmarks, continually updated environments where agents must adapt to shifting conditions. And as tasks grow more complex (long reports, multi-week coding projects, open-ended investigations), evaluation itself becomes the bottleneck. In a sense, benchmarking general intelligence may evolve from datasets to ecosystems, from static tasks to interactive worlds. Whether these efforts will ever converge toward a consensus definition of intelligence is unclear. But one thing is already certain: evaluating general agents is no longer simply a matter of scoring answers. It is an open challenge that forces us to rethink the very nature of what we are trying to measure.

II) Practical guide for Agents evaluation :

Despite the surge of academic benchmarks and the flurry of agent-centric surveys, a growing disconnect is emerging between research and industry needs. Most existing benchmarks ultimately serve one audience: model providers aiming to polish their next frontier model. They help characterize agentic capabilities in vitro, not evaluate agents in vivo. Meanwhile, the real challenge facing practitioners is far more concrete: **how do we evaluate a specific agent, built for a specific task, operating within a specific environment?** And even more importantly, how do we integrate this evaluation into the development cycle so that the agent actually improves?
The next part of the chapter provides a guide to evaluating agents in production contexts, where the goal is not to rank models but to make systems work.

a) Zooming in and out : multilayer evaluation

Evaluating AI agents means evaluating a *system*, not an isolated model. In classical machine learning, evaluation is tightly coupled to training: you measure the model's output, compute a score, adjust the parameters, and repeat. The final evaluation simply checks whether the refined model now performs well on the task. But agents break this clean logic. They are multi-component systems, often integrating several models, external tools, retrieval components, memory modules, and environment interfaces. Evaluating only the final output is necessary, but nowhere near sufficient to understand what should be modified to improve performance on the next run. Improving an agent requires understanding not just what it produces, but **how** it produces it.

This problem is not entirely new. It resembles the challenges faced when evaluating complex NLP pipelines such as RAG or Open Information Extraction. These systems have multiple sequential steps, and errors propagate through the pipeline. A small drift at each stage compounds dramatically: with five steps at 90% accuracy each, the global success rate drops to 59%. In such systems, one must distinguish the evaluation of the final answer from the evaluation of each stage in the process. This is component-wise evaluation—often overlooked, yet absolutely central to diagnosing system behaviour.

A component can be evaluated in two ways. *Extrinsic evaluation* modifies one component and observes how changes affect the system as a whole. In a RAG pipeline, for instance, increasing the

retriever’s top-k from 5 to 10 may improve recall or may pollute the context with irrelevant documents and degrade the final answer.

Intrinsic evaluation, by contrast, designs a specific test for the component itself, scoring only the transformation between its input and its output. Before integrating any component into an agent, especially external tools, it is crucial to evaluate it in both senses on data similar to the production scenario. Dynamic systems can behave unpredictably—a pipeline built from two individually “worse” components can sometimes outperform one assembled from “better” ones—but incorporating the least error-prone components remains an essential first filter to avoid aberrant behaviour.

Some components, however, are more central than others. Beyond the final-output evaluation and the component-wise analysis, one must also evaluate system-level behaviours—how components interact, coordinate, and cascade. This is where evaluating LLM agentic capabilities becomes indispensable. The model must plan efficiently, avoid prematurely producing an unfinished answer, select relevant tools, infer the correct intent behind a task, and fill parameters appropriately. These behaviours are not surface features; they determine whether the entire agent executes its mission competently.

This multilayered evaluation is long and costly, but it is also the only approach that produces actionable insight. If an agent satisfies only 0.6% of user expectations for final outputs, a global score does not reveal *why*. “If the agent fails to book a flight, was it because the LLM misunderstood the itinerary, the retrieval returned outdated data, the tool API failed, or the memory step reused an incorrect state?” A single sentence like this would immediately anchor the reader. Evaluating each component’s behaviour, step by step, on the failing examples reveals precisely where the drift occurred and where developers should focus their efforts for maximum improvement.

The goal of this evaluation process is straightforward: to transform complex multi-step behaviours into interpretable, actionable signals. Only then can we monitor agents effectively and understand, not just whether they failed,but *how* they failed, and how to make them succeed next time. The engineering challenge is therefore to design ways to evaluate each component and the final output effectively.

b) How to evaluate ; The Revenge of the Metrics

AI agents are now deployed on an extraordinary diversity of tasks: translating low-resource languages, filing customer-support tickets, debugging software, summarizing legal cases, computing Minecraft statistics, or drafting research papers. Some of these tasks have well-established evaluation protocols inherited from decades of NLP work; others belong to entirely new categories for which no prior metric exists. The expansion of what can be automated has therefore produced a corresponding explosion in task types—far beyond what our evaluation toolkit was originally designed to handle.

The most immediate and intuitive way to evaluate an agent is to compute its **success rate**: out of all the examples in the evaluation set, how many did the agent handle satisfactorily? This is simple, interpretable, and appealing, but even this basic metric reveals serious technical challenges.

First, **success rate lacks granularity**. If the agent is tasked with writing an NLP research paper, a binary metric,“accepted at a conference or not”is hardly informative. Human reviewers do not think in binaries: they give fine-grained scores (from -2“strong reject” to 2“strong accept”) and provide textual feedback that indicates what worked, what failed, and how the work could be improved. Agents deserve at least the same level of nuance. Second, for complex tasks there is often **no obvious way to decide whether a run is successful at all**. Determining success may require task-specific expertise, subjective judgement, or multi-dimensional criteria. We can of course rely on human evaluators, but this quickly becomes prohibitively expensive and slow, especially for agents designed to operate continuously or interactively.

Engineers must therefore develop principled ways to *score* agents—whether by evaluating the system as a whole or by isolating the performance of individual components. Metric design has always been a foundational topic in NLP, but in the agent era it becomes even more critical. The challenge is to create interpretable scores that capture the behaviour of complex, multi-step systems without oversimplifying or distorting what is actually happening. In the following pages, we explore the techniques available to design such metrics.

Metrics

Whenever agents (or, more simply, LLMs) are used on tasks historically tackled by specialized ML models, the existing task-specific metrics remain valuable. Machine Translation is the canonical example: if an agent performs translation, the same metrics that evaluated translation systems(BLEU, chrF, COMET…) remain valid tools to assess its output. Of course, metrics can saturate just as benchmarks do: once systems cluster at the top of the scale, small numerical differences become meaningless and no longer discriminate between models. Some metrics, such as BLEU, have also been heavily criticized for the way they compute similarity. But better alternatives eventually emerge, and the key is to choose metrics intentionally and interpret them critically.

Traditional metrics are also indispensable for evaluating *tools* independently. Any component involving information retrieval, for instance, can still be measured with top-k precision, recall, and other IR metrics, robust, well-understood, and unaffected by the agentic framing.

Beyond these classical measures, more general behavioural metrics are beginning to appear. They aim to characterize aspects of agent behaviour that did not previously exist: the success rate of tool selection, the correctness of argument filling, the coherence of multi-step planning, the ability to recover from errors, or the stability of the agent’s trajectory under minor perturbations. These emerging metrics do not replace task-specific ones; they complement them by providing a higher-level view of how the agent behaves across tasks. They are particularly valuable for monitoring the global health and reliability of an agentic system.

Ideally, a metric should quantify a clear evaluation criterion and reveal something informative about the component it measures, even if some tasks still require broader, more holistic metrics to capture system-level behaviour.

Objectives	Category	Metrics	Relevant Papers
Agent Behavior	Task Completion	Success Rate (SR), F1-score, Pass@k, Progress Rate, Execution Accuracy, Transfer Learning Success, Zero-Shot Generalization Accuracy	AgentBoard [8], WebShop [103], AgentBench [58], Tool Use Evaluation [53], InformativeBench, SQuAD [79][78] [56], ResearchArena [41], AgentBoard [8], AppWorld [91], TheAgentCompany [97], MAGIC [99], Mobile-Env [117], Re-ReST [19], XMC-AGENT [59], SWE-bench [40]
	Output Quality	Coherence, User Satisfaction, Usability, Likability, Overall Quality	PredictingIQ [80], EnDex [98], PsychoGAT [101]
	Latency & Cost	Latency, Token Usage, Cost	Cluster diagnosis [84], MobileBench [18], MobileAgentBench [92], LangSuitE [39], WebArena [126], Mobile-env [116], GUI Agents [94], GPTDroid [60], Spa-bench [10]
Agent Capability	Tool Use	Task Completion Rate, Tool Selection Accuracy	ToolEmu [81], MetaTool [38], AutoCodeRover [119]
	Planning & Reasoning	Reasoning Quality, Accuracy, Fine-Grained Progress Rate, Self Consistency, Plan Quality	AgentBoard [8] Massive Multitask Language Understanding [36], LLM-Augmented Autonomous Agents [55], Cluster diagnosis questions [84], SimuCourt [35], Magis [90]
	Memory & Context Retention	Factual Accuracy Recall, Consistency Scores	LongEval [43], SocialBench [9], LoCoMo [65], Optimus-1 [50]
	Multi-Agent Collaboration	Information Sharing Effectiveness, Adaptive Role Switching, Reasoning Rating	AgentSims [52], WebArena [126], MATSA [66], GAMEBENCH [15], BALROG [72], TheAgentCompany [97]
Reliability	Consistency	pass^k	τ-Bench [104]
	Robustness	Accuracy, Task Success Rate Under Perturbation	HELM [51], WebLinX [63]
Safety	Fairness	Awareness Coverage, Violation Rate, Transparency, Ethics, Morality	CASA [77], R-Judge [112], SimuCourt [35], MATSA [66], FinCon [111], AutoGuide [24]
	Harm	Adversarial Robustness, Prompt Injection Resistance, Harmfulness, Bias Detection	Agent Security Bench(ASB) [118], AgentPoison [13], AgentDojo [17], Backdoor Attacks [102], SafeAgentBench [108], Agent-Safety Bench [122], AgentHarm [5], Adaptive Attacks [113], RealToxicityPrompts [27]
	Compliance & Privacy	Risk Awareness, Task Completion Under Constraints	R-Judge [112], Cybench [114], TheAgentCompany [97]

Exact Match

When reference (or *golden*) data is available—typically for tasks requiring the extraction of specific information, such as answering an HLE question or retrieving the price of an item during a customer-service interaction, the agent's success can often be reframed as its ability to identify the correct data. In such cases, the evaluation becomes refreshingly straightforward: extract the key information from the agent's output (using structured outputs or a simple REGEX), compare it with the expected answer, and compute a standard score. Accuracy, recall, and F1, metrics we all know by heart, remain the most efficient and transparent way to determine whether the tool or agent succeeded.

Sometimes, even in the absence of full reference data, **explicit criteria** can still be checked automatically. If a customer-support agent is expected to greet the user by name, we can verify whether the name appears in the generated message. If an agent is supposed to book a train ticket but no booking is made, the run is unambiguously a failure. These targeted checks allow us to monitor fine-grained aspects of the agent's behaviour, much like unit tests on a codebase: small, precise, high-signal diagnostics that catch subtle errors before they compound into systemic failures.

LLM as a Judge

Exact-match techniques can only assess quantitative elements , whether a piece of information appears in the output, how many items were retrieved, what percentage of subwords overlap with a reference, and so on. But many agentic tasks depend on *qualitative* criteria: is the translation fluent? Is the answer factually correct and grounded in the provided context? Does the summary strike the

right balance between completeness and conciseness? These forms of quality are much harder to evaluate automatically.

This difficulty forces a shift in how metrics are designed. Traditional metrics emerged from what was objectively observable in the raw output. Qualitative criteria, by contrast, require us to bring human preferences back into the center of the evaluation process. Instead of measuring only what we can count, we first articulate what we *want*: the criteria that define a good answer for a given task. Then we design metrics that try to capture those criteria. Finally, we must demonstrate—through annotated datasets—that the metric correlates well with human judgement. This long, tedious, and often noisy procedure is the well-known process of **meta-evaluation**: evaluating the evaluator.

However, meta-evaluation requires substantial data collection and only works for *one criterion at a time*. Given the speed at which Agent tasks diversify, we cannot realistically engineer a separate trained metric for every new qualitative dimension of agent behaviour.

This is where a new idea emerged: if qualitative judgement requires linguistic understanding, why not use a model that already excels at linguistic understanding? Thus appeared what is now broadly known as **LLM-as-a-Judge**. It has gained enormous traction in industry because it is straightforward to implement and far more flexible than hand-crafted metrics.

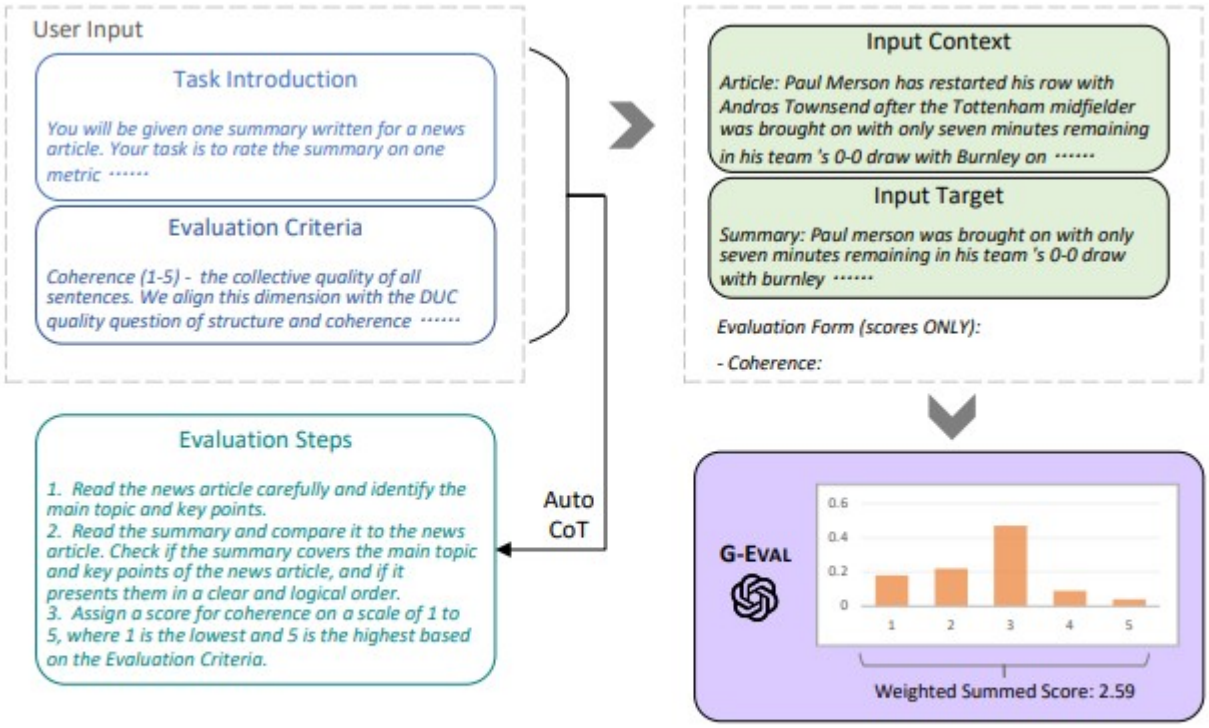


Figure 1: The overall framework of G-EVAL. We first input Task Introduction and Evaluation Criteria to the LLM, and ask it to generate a CoT of detailed Evaluation Steps. Then we use the prompt along with the generated CoT to evaluate the NLG outputs in a form-filling paradigm. Finally, we use the probability-weighted summation of the output scores as the final score.

• Pairwise comparison.
The LLM is given the definition of one or more criteria, the task input, and two candidate outputs. It must decide which one is better. GPT-4, for example, has been shown to correlate strongly with

human judgement in pairwise settings[3]. This approach is useful when comparing systems—during development or against a competitor—but it does not provide an absolute measure of how well a single system performs.

• Likert-scale evaluation.
The LLM receives the definition of a criterion in its system prompt, along with the task input, the agent's output, and any available reference. It then assigns a qualitative score—typically on a 1-to-5 or 1-to-7 scale. This configuration offers a direct, interpretable assessment of a system's performance and has become a de facto standard for evaluating complex, high-level agentic tasks.

However, even if this technique is now widely adopted, the use of LLMs as evaluators raises (though not in the mind of everyone, unfortunately) a number of serious questions:

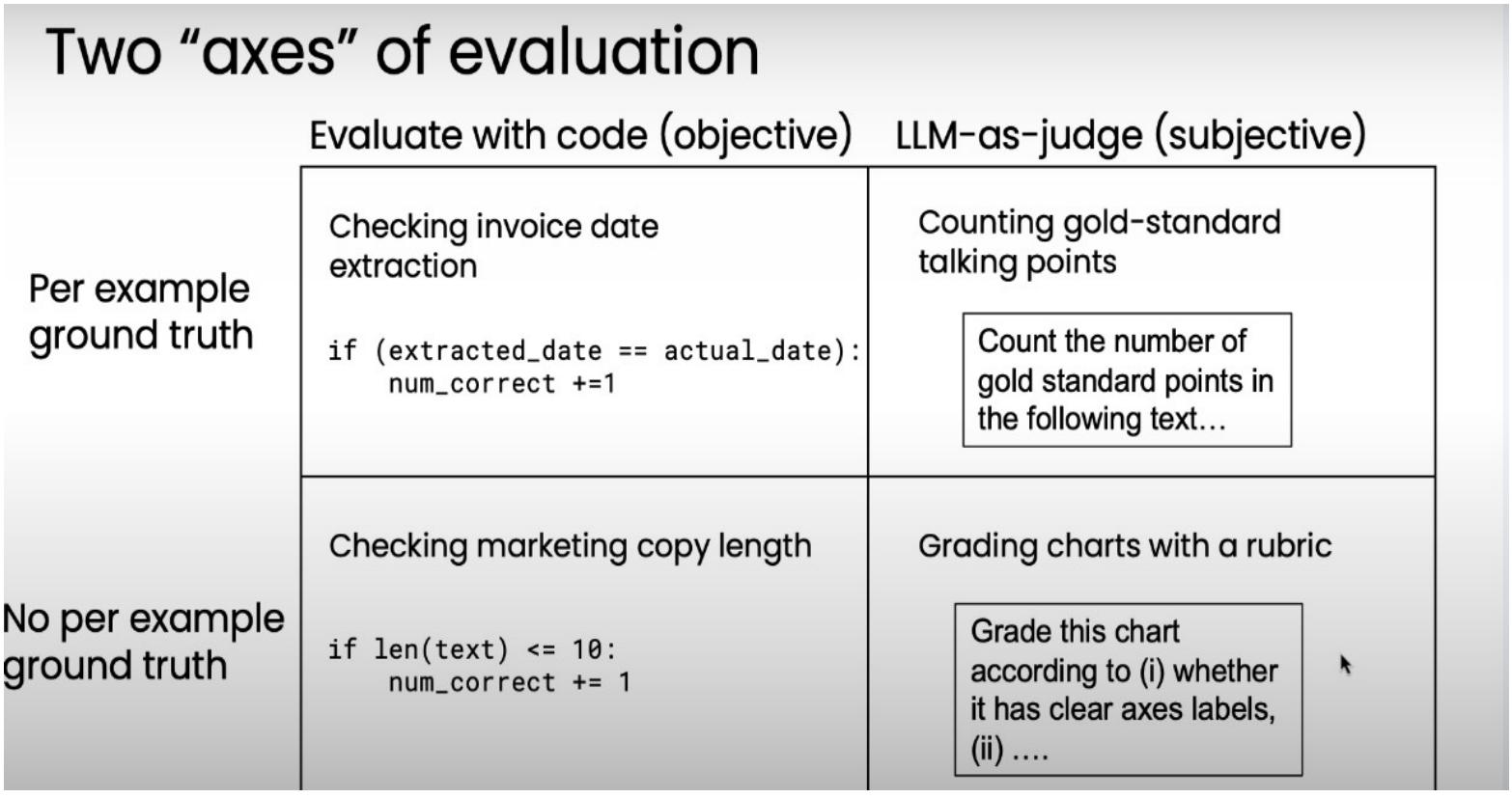
- **Is accurate, human-aligned grading really an emergent ability of LLMs?**
  The empirical results are promising but heterogeneous. There is no theoretical guarantee that a model capable of generating fluent text is capable of consistently evaluating it.
- **Judging quality varies significantly from one LLM to another.**
  Two models may grade the same output very differently, with sometimes worrying variance — especially on nuanced or ambiguous tasks.
- **Grades improve when the evaluator is forced to justify itself (CoT).**
  Adding chain-of-thought reduces variance and increases agreement with human annotators, but at the cost of longer, more expensive evaluations.
- **LLM-as-a-Judge works best with a precise, operationalized grading scale.**
  "What is a 1? What is a 5?" must be explicitly defined; otherwise the evaluator drifts, invents criteria, or applies inconsistent thresholds.
- **Providing reference data usually improves reliability.**
  Evaluators tend to hallucinate "qualities" when no ground truth is given.
- **This technique is most reliable when scoring a single criterion at a time.**
  The eternal problem of multi-criteria evaluation returns: LLMs struggle to aggregate multiple dimensions (e.g., factuality + coherence + conciseness) in a controlled way.
- **The same model may assign different grades across multiple runs.**
  Because the score is produced via token sampling, even with structured output, the model is sampling a *probability distribution* over {1,2,3,4,5}.
  If we sample → lack of robustness.
  If we argmax → we lose information about uncertainty.
  Some recent systems (e.g., *EvalGPT* for summarization) try to use the *full logit vector* to compute a calibrated score, which indeed works better.
- **LLM-as-a-Judge scores are better than random—but still below state-of-the-art trained metrics.**
  Fine-tuned evaluators like *Prometheus 1 / 2* attempt to explicitly teach a model to judge across many criteria, yet the correlations remain below what we would need to consider LLM judges as trustworthy or stable across domains.

3https://arxiv.org/abs/2306.05685

- **Judgement quality depends heavily on the prompt.**
  There is no established recipe for writing a “good” evaluation prompt. Small changes in wording, order of instructions, verbosity, or style can significantly shift scores.
  Prompt sensitivity is one of the most under-discussed fragilities of LLM evaluation.
- **LLM evaluators inherit all the biases of the underlying model.**
  This includes:
  – **Positional bias** (preferring the first or second option in pairwise tasks).
  – **Self-preferring bias** (favoring outputs from the same model family).
  – **Length bias** (rewarding verbosity over conciseness).
  – **Stylistic bias** (preferring outputs that resemble the judge’s own training distribution).
  These biases distort evaluation and create systematic unfairness between systems.
- **Optimizing a system to “please the judge” risks overfitting.**
  If LLM-as-a-Judge becomes the evaluation oracle, agents will evolve strategies to score highly with the evaluator rather than satisfy genuine human expectations. This “reward hacking” mirrors the Turing test problem: systems learn to exploit the evaluation protocol instead of improving at the task.

Even if LLM as a judge is not a mature evaluation system used as a numeric scorer, its still a viable solution when there is no other metric available to check the success of a run or a specific rubric. We can assume that a significant drop in the agent assessed performance over time is likely the sign of a degradation of the output’s actual quality / that this drop would also be caracterized by human judgements.

Part of the AI Engineer’s job is now to design adapted ways to test the general agentic systems he is implementing as well as its components leveraging the variety of tools he has at his disposal.[4]



TODO : paragraph on « Agent as a judge »

c) How to integrate evaluation during the development process of the agent.

Read paper on Evaluation Driven Development

Notion of control flow
« Reflection and error correction are two different mechanisms that go hand in hand. Reflection generates insights that help uncover errors to be corrected. »
Evaluation as reflexion ?

Execution and dependency as a graph

List systematically what could go wrong.
For tools bad choice, wrong arguments or wrong arguments values

An interesting mode of planning failure is caused by errors in reflection. The agent is convinced that it's accomplished a task when it hasn't.

Redefying what failure is vs classical software engineering.
Code crashed or misses values.

"*Manual inspection of data has probably the highest value-to-prestige ratio of any activity in machine learning.*" talk about LLM as a judge and its pitfalls.

CI CD for benchmarks ? Continuously evolving

Discussion on how to keep human in the loop
the need for synthetic data
A/B testing

CCL : are we not back to model evaluation after all ?

III) From Observation to Evolution

a) Observability is mandatory
terminology observability monitoring logging and telemetry

What does it imply.

=> aggregations and insight

b) From notebook to produciton

botteneck
Data governance

c) self improvement pipeline.

Quote Open AI