- How easily can an entirely new algorithmic approach be tested at full scale?

The paper posits that a new approach algorithmically will have issues being tested at full scale since the current ML development pipelines with glue code (estimated at 95% in the source material) and 5% ML engineering puts any new approaches square against the limitations of the packages gluing the model together. A special case of glue code, pipeline jungles often appear in data preparation. These can evolve organically, as new signals are identified and new information sources added incrementally. Without care, the resulting system for preparing data in an ML-friendly format may become a jungle of scrapes, joins, and sampling steps, often with intermediate files output. Managing these pipelines, detecting errors, and recovering from failures are all difficult and costly. Testing such pipelines often requires expensive end-to-end integration tests. All of this adds to technical debt of a system and makes further innovation more costly. inhibiting improvement and leveraging domain knowledge to test approaches. Robust domain knowledge application is another challenge unto itself as the bifurcation between researchers and developers results in teams implementing the models being largely black boxes which cause challenges in maintenance and updates. The paper recommends a hybrid research approach where engineers and researchers are embedded together on the same teams (and indeed, are often the same people) can help reduce this source of friction significantly.

- What is the transitive closure of all data dependencies?

The transitive closure of all data dependencies i.e., the root cause of the problem can be answered by observing the various types of data dependencies that may occur and in finding the smallest set of solutions to them. The paper suggests that dependency debt is a key contributor to code complexity and technical debt in classical software engineering settings. We have found that data dependencies in ML systems carry a similar capacity for building debt but may be more difficult to detect. Code dependencies can be identified via static analysis by compilers and linkers. Without similar tooling for data dependencies, it can be inappropriately easy to build large data dependency chains that can be difficult to untangle. According to the paper, one solution to the various types of data dependencies from unstable mapping from lookup tables to legacy and underutilized features can be identified by dependency graphs are tools for static analysis of data dependencies which, while are far less common, are essential for error checking, tracking down consumers, and enforcing migration and updates. One such tool is the automated feature management system which performs transitive closure and transitive reduction in knowledge mining graphs.

- How precisely can the impact of a new change to the system be measured?

Since there are no clearly defined metrics for impact and testable invariants are not always obvious given that many ML systems are intended to adapt over time, It is often necessary to pick a decision threshold for a given model to perform some action: to predict true or false, to mark an email as spam or not spam, to show or not show a given ad. One classic approach in machine learning is to choose a threshold from a set of possible thresholds, in order to get good tradeoffs on certain metrics, such as precision and recall. However, such thresholds are often manually set. Thus, if a model updates on new data, the old manually set threshold may be invalid. Manually updating many thresholds across many models is time-consuming and brittle. One mitigation strategy for this kind of problem appears in, in which thresholds are learned via simple evaluation on holdout validation data. Now, in defining success, it is important to consider the differences between business performance and model performance. With model performance being addressed above, to measure the impact of a new change to the system on a business strategy, analysis of key performance indicators would be one of the steps for a robust measure of model performance.

- Does improving one model or signal degrade others?

  Two types of data dependencies between models might actually hinder improvements and degrade overall performance. They have been explored below as discussed by the paper.

To move quickly, it is often convenient to consume signals as input
features that are produced by other systems. However, some input signals are *unstable*, meaning
that they qualitatively or quantitatively change behavior over time. This can happen implicitly,

when the input signal comes from another machine learning model itself that updates over time,
or a data-dependent lookup table, such as for computing TF/IDF scores or semantic mappings. It
can also happen explicitly, when the engineering ownership of the input signal is separate from the
engineering ownership of the model that consumes it. In such cases, updates to the input signal
may be made at any time. This is dangerous because even "improvements" to input signals may
have arbitrary detrimental effects in the consuming system that are costly to diagnose and address.
For example, consider the case in which an input signal was previously mis-calibrated. The model
consuming it likely fit to these mis-calibrations, and a silent update that corrects the signal will have
sudden ramifications for the model.
One common mitigation strategy for unstable data dependencies is to create a *versioned copy* of a
given signal. For example, rather than allowing a semantic mapping of words to topic clusters to
change over time, it might be reasonable to create a frozen version of this mapping and use it until
such a time as an updated version has been fully vetted. Versioning carries its own costs, however,
such as potential staleness and the cost to maintain multiple versions of the same signal over time.

Undeclared consumers may be difficult to detect unless the system is specifically designed to guard
against this case, for example with access restrictions or strict service-level agreements (SLAs). In
the absence of barriers, engineers will naturally use the most convenient signal at hand, especially
when working against deadline pressures.

- How quickly can new members of the team be brought up to speed?

While there is no strict metric, the process of speeding up onboarding new team members relies on team culture regarding pipelines and flexibility in the software cycle process. The paper says that there is sometimes a hard line between ML research and engineering, but this can be counter-productive for long-term system health. It is important to create team cultures that reward deletion of features, reduction of complexity, improvements in reproducibility, stability, and monitoring to the same degree that improvements in accuracy are valued. In our experience, this is most likely to occur within heterogeneous teams with strengths in both ML research and engineering. This helps debt management, code reviews and updating of products be more resilient and forward faced as one of the things that makes ML systems so fascinating is that they often interact directly with the external world. Experience has shown that the external world is rarely stable. This background rate of change creates ongoing maintenance cost and a team culture adapted and invested in this can make this more manageable.