A Comprehensive Testing Framework for Power BI Semantic Models: Evaluating Import, DirectQuery, and Composite Architectures

Executive Summary

The selection of a semantic model's storage mode represents a foundational architectural decision within any Microsoft Power BI implementation. This choice fundamentally dictates the resultant solution's performance profile, data freshness capabilities, scalability ceiling, and the degree of developmental flexibility afforded to creators. An incorrect choice at the project's inception can lead to significant performance bottlenecks, user dissatisfaction, and costly remediation efforts. This report provides a comprehensive, expert-level framework for systematically testing and evaluating the three primary storage modes in Power BI: Import, DirectQuery, and Composite. The objective is to equip development teams with the methodologies, tools, and analytical insights required to make informed, evidence-based architectural decisions that align with specific business and technical requirements.

The testing methodologies detailed herein are tailored to the unique characteristics of each storage mode. A singular, standardized testing approach is insufficient due to the fundamentally different ways each mode processes and retrieves data. For Import models, the testing protocol emphasizes memory footprint analysis, data refresh efficiency, and the optimization of in-memory query performance. For DirectQuery models, the focus shifts externally to the underlying data source, with tests designed to measure native query latency, network overhead, and the impact of query concurrency and connection throttling. Finally, for Composite models, the framework addresses the complexities of a hybrid architecture, concentrating on the performance of inter-source relationships, the effectiveness of aggregation strategies, and the governance implications of model chaining.

The core findings of this analysis will demonstrate a clear and predictable set of trade-offs. Import mode provides unparalleled interactive query performance by leveraging the in-memory VertiPaq engine but is constrained by model size and data latency dictated by refresh schedules. DirectQuery offers the capability to analyze massive datasets and deliver near real-time data but cedes performance control to the underlying source system and network infrastructure. Composite models present a powerful hybrid solution, enabling a balance between performance and data freshness, but introduce significant architectural complexity and potential performance pitfalls if not designed with precision. By engaging with the rigorous testing protocols outlined in this document, development teams will build a deep, mechanical understanding of these trade-offs, enabling them to construct a robust decision-making framework for selecting the optimal semantic model architecture for any given analytical challenge.

Foundational Principles of Power BI Semantic Model

Performance

A prerequisite to effective performance testing is a comprehensive understanding of the underlying engine architecture that governs how Power BI processes requests. The performance characteristics observed in different storage modes are not arbitrary; they are the direct result of a sophisticated, multi-component system designed to balance data preparation, calculation, and retrieval. A detailed analysis of this architecture reveals why certain design patterns excel while others falter.

Deconstructing the Power BI Engine Architecture

At its core, Power BI operates on an instance of the Analysis Services Tabular engine, whether locally within Power BI Desktop or as a cloud-based instance in the Power BI service. This engine is composed of two primary, cooperative components: the Formula Engine and the Storage Engine. The interplay between these two engines is the single most critical factor determining the performance of any DAX query.

The Dual-Engine Paradigm: Formula Engine (FE) vs. Storage Engine (SE)

When a user interacts with a Power BI report, one or more DAX queries are generated and sent to the Analysis Services engine. The engine processes these queries through a two-stage execution plan orchestrated by the Formula Engine and the Storage Engine.

- **Formula Engine (FE):** The Formula Engine can be conceptualized as the "brain" or orchestrator of a DAX query. Its responsibilities include parsing the DAX syntax, generating a query plan, and executing calculations. Crucially, the Formula Engine is a single-threaded process. This architectural characteristic means it processes operations sequentially and cannot leverage multiple CPU cores for a single request. Consequently, if a significant computational workload is forced upon the Formula Engine—such as complex calculations or joins that cannot be delegated—it becomes a severe performance bottleneck.
- **Storage Engine (SE):** The Storage Engine acts as the "muscle" of the query execution process, responsible for data retrieval. In stark contrast to the Formula Engine, the Storage Engine is multi-threaded and designed for high-performance, parallel operations. It can scan, filter, join, and aggregate large volumes of data with remarkable efficiency. The goal of any performance tuning effort is to maximize the work done by the multi-threaded Storage Engine and minimize the work required of the single-threaded Formula Engine.

The VertiPaq Engine: The Heart of Import Mode

For semantic models using Import mode, the Storage Engine is a proprietary, in-memory, columnar database technology known as VertiPaq. The exceptional performance of Import mode is a direct consequence of VertiPaq's design.

- **Columnar Storage and Compression:** Unlike traditional row-store databases, VertiPaq stores data in columns. This structure is highly optimized for analytical workloads, which typically involve aggregating values within a few columns over many rows. Because all the data in a single column is of the same type, it is highly compressible. VertiPaq

employs multiple compression algorithms, often achieving compression ratios of 10:1 or greater, which significantly reduces the model's memory footprint and allows vast amounts of data to be held in RAM.
- **Dictionaries and Relationships:** To further enhance performance, VertiPaq creates dictionary-encoded representations of each column. This process replaces raw data values with smaller integer substitutes, which are faster to process. It then builds data structures that materialize the relationships between tables, allowing for extremely fast joins and filter propagation entirely within the in-memory cache.

The DirectQuery Gateway: A Bridge to Live Data

In DirectQuery mode, Power BI does not store a copy of the data. Instead, the semantic model contains only metadata—the schema of the tables, relationships, and measures. The Storage Engine's role shifts from querying an in-memory cache to acting as a query translation and pass-through mechanism.

- **DAX to Native Query Translation:** When a visual generates a DAX query, the Storage Engine translates it into the native query language of the underlying data source, such as T-SQL for SQL Server or Snowflake SQL. This native query is then sent to the source system for execution.
- **Performance Dependencies:** The performance of a DirectQuery report is therefore not primarily a function of Power BI's capabilities, but rather a dependency on three external factors: the efficiency of the source database in executing the generated queries, the network latency between the Power BI service (or on-premises gateway) and the data source, and the number of concurrent queries being sent to the source. Poor performance in DirectQuery mode is almost always attributable to a bottleneck in one of these three areas.

The Composite Model Engine: A Hybrid Architecture

Composite models introduce a more complex architecture by allowing a single semantic model to contain multiple "source groups". A source group is a collection of tables from a single storage mode; for example, all imported tables belong to one source group, while tables from a DirectQuery connection to a SQL Server database belong to another.

- **Source Groups and Relationship Types:** Relationships *within* a single source group (e.g., between two imported tables or two tables from the same DirectQuery source) are known as "regular" relationships and are highly performant. However, relationships that span *across* different source groups (e.g., connecting an imported dimension table to a DirectQuery fact table) are designated as "limited" relationships. Limited relationships have weaker semantics and can result in less efficient join patterns, potentially impacting both performance and data integrity if referential integrity is not maintained between the sources.
- **Cross-Source Query Execution:** The most significant performance implication of composite models arises from cross-source group queries. When a single DAX query requires data from multiple source groups, the Storage Engine cannot push the entire join and aggregation logic to a single backend. Instead, it must retrieve separate result sets from each source group. These intermediate result sets are then passed to the single-threaded Formula Engine, which must perform the final join and aggregation operations locally. This architectural shift—moving the workload from a powerful,

multi-threaded source engine (like VertiPaq or a SQL database) to the constrained, single-threaded Formula Engine—is the primary reason that poorly designed composite models can suffer from severe performance degradation.

A Unified Framework for Performance Testing

To produce reliable and comparable results, all performance tests must be conducted within a controlled environment using a standardized toolkit. This framework establishes the baseline procedures and introduces the essential diagnostic tools required for a thorough analysis of any Power BI semantic model, regardless of its storage mode.

Establishing a Controlled and Repeatable Baseline

Before executing any performance test, it is imperative to mitigate the influence of caching, which can obscure true performance characteristics by serving pre-calculated results.

- **Clearing the Cache:** A two-stage cache-clearing process must be performed before each test run. First, the **visual cache** in Power BI Desktop should be cleared by navigating to a blank report page before starting a recording in the Performance Analyzer. This ensures that visuals are re-rendered from scratch. Second, the **data engine cache**, which stores the results of recent DAX queries, must be cleared. This can be accomplished either by restarting Power BI Desktop or, more efficiently, by using the "Clear Cache" command within DAX Studio. Failure to perform these steps will result in misleadingly fast query times that reflect cache retrieval speed rather than actual model performance.

The Core Performance Testing Toolkit

A suite of specialized tools is required to diagnose performance across the different layers of a Power BI solution. While each tool has a primary purpose, their true power is realized when used in concert to trace a performance issue from a high-level symptom down to its root cause.

Power BI Performance Analyzer

The Performance Analyzer is an integrated tool within Power BI Desktop that serves as the first line of defense for diagnosing performance issues at the report level. It records the time taken for each visual on a page to render, breaking down the duration into three key metrics.

- **DAX Query:** This metric measures the total time the Analysis Services engine takes to process the query for a visual. This includes time spent by the Formula Engine processing the request and time spent waiting for the Storage Engine to retrieve the data. A consistently high "DAX Query" duration is a primary indicator of an inefficient measure, a complex data model structure, or a slow underlying data source in DirectQuery mode.
- **Visual Display:** This metric isolates the time required for the visual to render on the canvas *after* the data has been returned by the engine. A high "Visual Display" time points to issues with the visual itself, such as rendering an excessive number of data points (e.g., a scatter plot with thousands of dots) or using a poorly optimized custom visual.
- **Other:** This category captures the time a visual spends waiting for other operations to complete, such as waiting for other visuals on the same page to finish rendering. A high "Other" duration for multiple visuals on a page often indicates a "noisy neighbor" problem,

where one particularly slow visual is creating a bottleneck and delaying the rendering of all other visuals.

The data captured by the Performance Analyzer can be exported to a JSON file, providing a mechanism for logging results, tracking optimizations over time, and performing more detailed offline analysis.

DAX Studio

DAX Studio is an essential, open-source external tool for advanced performance tuning and in-depth analysis of Power BI semantic models. Its capabilities extend far beyond what is available within the Power BI Desktop interface.

- **Server Timings:** This is arguably the most critical feature for DAX optimization. When a query is executed from DAX Studio with Server Timings enabled, it provides a detailed breakdown of the processing time, explicitly showing the time spent in the Formula Engine (FE) versus the Storage Engine (SE). This granular insight is indispensable for diagnosing bottlenecks. A high FE time relative to SE time is a definitive sign that the DAX calculation is inefficient and is not being effectively pushed to the more powerful, multi-threaded Storage Engine.
- **VertiPaq Analyzer:** For Import mode models, the VertiPaq Analyzer is an invaluable diagnostic tool. It connects to the data model and provides a comprehensive breakdown of its memory structure, including the size of each table and column, column cardinality (the number of unique values), and the compression method used. This allows developers to quickly identify which columns are consuming the most memory and are candidates for optimization.
- **Query Trace:** DAX Studio can capture a real-time trace of all DAX queries being executed against the model, which is useful for understanding the volume and type of queries generated by complex report interactions.

SQL Server Profiler & Log Analytics

For DirectQuery models, the performance bottleneck often lies outside of Power BI, within the source database itself. SQL Server Profiler and Azure Log Analytics are the primary tools for capturing and analyzing the native queries that Power BI generates and sends to the data source.

- **Methodology:** By connecting SQL Server Profiler to the local diagnostics port of Power BI Desktop, a developer can run a trace that captures all backend events. The key events to monitor are DirectQuery Begin and DirectQuery End. The TextData field for these events contains the exact native SQL query that was sent to the source. This captured query can then be executed directly in a database management tool like SQL Server Management Studio (SSMS) to analyze its performance and execution plan, completely isolating it from Power BI and network factors.
- **New Execution Metrics for Throttling:** Recent updates have introduced a new Execution Metrics event in Profiler and Log Analytics. This event exposes a critical new metric: datasourceConnectionThrottleTimeMs. A non-zero value for this metric indicates the amount of time, in milliseconds, that a DAX query spent waiting for an available connection to the data source to become free. This provides a definitive way to diagnose performance issues caused by connection pool exhaustion, a common problem in high-concurrency DirectQuery scenarios.

The following table provides a quick reference for selecting the appropriate tool for a given performance investigation.

Tool Name	Primary Use Case	Key Metrics Provided	Applicable Model(s)	Level of Detail
Power BI Performance Analyzer	High-level report and visual diagnostics	DAX Query, Visual Display, Other	All	High-Level
DAX Studio (Server Timings)	Deep-dive DAX query optimization	Formula Engine (FE) vs. Storage Engine (SE) time	All	Granular
DAX Studio (VertiPaq Analyzer)	Import model memory analysis	Table/Column Size, Cardinality, Compression	Import, Composite	Granular
SQL Server Profiler	DirectQuery native query analysis	Generated SQL, Query Duration, Throttling Time	DirectQuery, Composite	Deep/Source-Level

Test Plan for Import Mode Semantic Models

Objective

The primary objective of testing Import mode semantic models is to evaluate and optimize the efficiency of the in-memory VertiPaq engine. The test plan focuses on three critical areas: minimizing the model's memory footprint, reducing the duration and resource impact of data refresh operations, and ensuring rapid interactive query performance.

Key Performance Indicators (KPIs)

- **Memory Footprint:** The total RAM consumed by the semantic model when loaded into the Power BI service.
- **Model Size on Disk:** The size of the .pbix file, which is indicative of the compressed model size.
- **Data Refresh Duration:** The time required to complete both full and incremental data refreshes.
- **Query Response Time:** The time taken for user interactions to complete, with a specific focus on the processing split between the Formula Engine (FE) and Storage Engine (SE).

Test 1: Model Size and Compression Analysis

The efficiency of the VertiPaq engine is directly proportional to the size and structure of the data model it must manage. A smaller, more optimized model results in faster performance and lower resource consumption.

- **Methodology:** This test involves a comprehensive audit of the semantic model's structure using DAX Studio's VertiPaq Analyzer. This tool provides a detailed view of how memory is allocated across all tables and columns within the model.
- **Actions:**
    1. Connect DAX Studio to the Power BI Desktop file.
    2. Run the VertiPaq Analyzer and sort tables and columns by size to identify the

largest consumers of memory.

3. **Analyze High-Cardinality Columns:** Pay special attention to columns with a high number of unique values (high cardinality), particularly those with a text data type. These columns are notoriously difficult for the VertiPaq engine to compress and are often the primary cause of model bloat. Investigate if these columns are essential for analysis or if they can be removed or split (e.g., splitting a high-cardinality DateTime column into separate Date and Time columns).
4. **Optimize Data Types:** Review the data type of each column. Where possible, convert columns to the most efficient data type. For example, converting decimal numbers to fixed decimal or integer types, and replacing text-based keys with integer surrogate keys, can lead to significant compression gains and faster relationship processing.
5. **Validate Data Reduction:** Confirm that all non-essential columns (vertical filtering) and rows (horizontal filtering) have been removed in Power Query before being loaded into the model. Every column adds to the memory cost, so a lean model is a fast model.

- **Success Criteria:** The primary goal is to achieve the smallest possible memory footprint without compromising the analytical requirements of the report. A successful optimization can often result in a model size reduction of an order of magnitude, for instance, from several gigabytes down to a few hundred megabytes.

Test 2: Data Refresh Performance

For Import models, data freshness is maintained through scheduled refreshes. These operations can be resource-intensive, consuming significant CPU and memory on the Power BI capacity.

- **Methodology:** This test involves benchmarking the duration and resource consumption of different refresh strategies within the Power BI service.
- **Actions:**
    1. Publish the semantic model to a dedicated test workspace on a Premium or Fabric capacity.
    2. **Benchmark Full Refresh:** Trigger an on-demand full refresh and record the total duration. Using the Capacity Metrics app, monitor the peak memory usage during the refresh. A full refresh requires approximately twice the final memory size of the model because Power BI keeps the old version of the model online to serve queries while building the new one.
    3. **Implement and Test Incremental Refresh:** For large fact tables, configure an incremental refresh policy in Power BI Desktop. This involves setting up RangeStart and RangeEnd parameters in Power Query and defining the policy in the model view. After publishing, perform an initial (long-running) full refresh, followed by subsequent refreshes. Time these subsequent refreshes and compare their duration to the full refresh benchmark.
    4. **Test Parallel Loading:** In Power BI Desktop options, ensure the "Enable parallel loading of tables" setting is active. This allows the Mashup engine to refresh multiple tables concurrently, which can significantly reduce overall refresh time, though it may increase the load on the source data system.
- **Success Criteria:** The duration of an incremental refresh should be a small fraction of the full refresh time. The total refresh time must fit within the operational window required by

the business, and the peak memory consumption during refresh should not exceed the limits of the target capacity, which could cause refresh failures.

Test 3: Interactive Query and Visual Performance

This test evaluates the end-user experience by measuring the responsiveness of the report to common interactions.

- **Methodology:** A combination of Power BI's Performance Analyzer and DAX Studio is used to capture and analyze the performance of DAX queries generated by user actions.
- **Actions:**
    1. In Power BI Desktop, open the Performance Analyzer and start a recording session.
    2. Simulate typical user behavior: apply slicers, cross-filter visuals by selecting data points, and use drill-through functionality.
    3. In the Performance Analyzer results, identify any visuals with a "DAX Query" duration that exceeds an acceptable threshold (a common starting point for investigation is any query over 120 milliseconds). Copy the slow DAX query to the clipboard.
    4. Paste the query into DAX Studio and run it with "Server Timings" enabled.
    5. Analyze the Server Timings results to determine the time distribution between the Formula Engine (FE) and the Storage Engine (SE). If the FE time constitutes a significant portion of the total duration, it indicates that the DAX logic is inefficient. The calculation is being processed by the single-threaded FE instead of being pushed down to the multi-threaded VertiPaq SE. Common culprits include the misuse of iterator functions like SUMX or FILTER over large tables where simpler aggregate functions would suffice.
    6. For models on Premium/Fabric capacity, test the effect of **Query Caching**. Enable this feature in the semantic model settings in the Power BI service. Open the report, navigate through the pages, and then open it a second time. The load time for the initial page view should be significantly faster on the second visit as the query results are served from the capacity's local cache instead of being recomputed.
- **Success Criteria:** Interactive queries should feel instantaneous to the user, with response times consistently under a few seconds at most. The Server Timings analysis should confirm that the vast majority of processing time for all significant queries is spent within the Storage Engine.

The performance of an Import model is almost entirely determined by decisions made within the Power BI Desktop file itself. Unlike DirectQuery, there is no external database or network to blame for slowness. Therefore, a systematic diagnostic process is essential. When a user reports that an Import mode report is slow, the initial step is to use the Performance Analyzer to isolate the specific visual and its associated DAX query that is causing the delay. Once the problematic query is identified, the next step is to use DAX Studio's Server Timings feature. This tool provides the definitive evidence to distinguish between a data retrieval problem and a calculation problem. If the Storage Engine time is high, it suggests the query is scanning a very large amount of data, pointing to a data model issue (e.g., a missing relationship or an overly large table). Conversely, if the Formula Engine time is high, the data model is likely efficient, but the DAX measure itself is poorly written. This forces the single-threaded Formula Engine to perform complex, row-by-row calculations that could have been handled far more efficiently by the multi-threaded VertiPaq Storage Engine. The solution, in this case, is not to alter the data

model but to refactor the DAX code to push the computational load back to the Storage Engine.

Test Plan for DirectQuery Mode Semantic Models

Objective

The objective of testing DirectQuery models is to evaluate the performance, reliability, and scalability of a solution that queries data sources in real-time. Testing must focus on the entire end-to-end data path, including the performance of the underlying source database, the efficiency of the generated native queries, network latency, and the management of query concurrency.

Key Performance Indicators (KPIs)

- **Source Query Latency:** The time taken for the underlying data source to execute a single native query generated by Power BI.
- **DAX Query Duration:** The total time from when a DAX query is issued to when the result is returned, including network latency and any processing within Power BI.
- **Concurrent Query Throughput:** The number of simultaneous queries the system can handle before performance degrades.
- **Connection Throttling Events:** The frequency and duration of events where queries are queued due to a lack of available database connections.

Test 1: Source System and Native Query Performance

The performance of a DirectQuery report is fundamentally constrained by the performance of its underlying data source. This test isolates and benchmarks the source system's ability to handle the queries generated by Power BI.

- **Methodology:** This test requires using a source-side monitoring tool, such as SQL Server Profiler for SQL Server, in conjunction with Power BI's Performance Analyzer to capture and analyze the native queries.
- **Actions:**
    1. In Power BI Desktop, use the Performance Analyzer to identify a slow-loading visual and copy its DAX query.
    2. Start a trace in SQL Server Profiler, ensuring you are monitoring the DirectQuery Begin and DirectQuery End events. Connect the profiler to the local Power BI Desktop diagnostics port.
    3. Refresh the slow visual in Power BI Desktop. The Profiler trace will capture the exact SQL query generated by Power BI in the TextData field of the DirectQuery End event.
    4. Copy this SQL query and execute it directly against the source database using a native tool like SQL Server Management Studio (SSMS). Record the execution time. This measurement provides a crucial baseline of the source system's performance, independent of Power BI or network latency.
    5. Analyze the query's execution plan within the database tool. Look for performance bottlenecks such as full table scans, inefficient joins, or missing indexes. Work with database administrators to optimize the source database (e.g., by adding indexes,

creating materialized views, or tuning the query) to improve this baseline performance.

- **Success Criteria:** The execution time of the native query at the source should be significantly less than the total acceptable response time for a visual. For a target visual load time of 5 seconds, the source query should ideally complete in under 2-3 seconds to allow for network and rendering overhead.

Test 2: Concurrency, Throttling, and Gateway Performance

A common and often misunderstood cause of poor DirectQuery performance is connection throttling. Power BI limits the number of concurrent queries it sends to a single data source. If more queries are generated than available connections, they are queued, leading to significant delays that are not apparent from analyzing a single query's performance.

- **Methodology:** This test systematically evaluates the impact of query concurrency by manipulating the connection limit and monitoring for throttling events.
- **Actions:**
    1. In Power BI Desktop's options, navigate to Current File > DirectQuery and set the "Maximum connections per data source" to a low value, such as 1. This will force all queries to execute sequentially.
    2. Create a test report page containing multiple simple visuals (e.g., 5-10 card visuals), each displaying a different measure. This design ensures that refreshing the page will generate multiple concurrent queries.
    3. Start a SQL Server Profiler trace, ensuring the Execution Metrics event is captured.
    4. Refresh the report page. Observe the Profiler trace. The Query End events will show that while the DAX queries were initiated in parallel, their durations are staggered (e.g., 10s, 21s, 33s), indicating sequential execution.
    5. Examine the TextData of the Execution Metrics event for each query. Look for the datasourceConnectionThrottleTimeMs value. A non-zero value is definitive proof that the query was delayed, waiting for a connection to become available from the connection pool.
    6. Incrementally increase the "Maximum connections per data source" setting (e.g., to 5, then 10) and repeat the test, observing the impact on total page load time and the throttling metric. The goal is to find the optimal number of connections that the source system can handle without performance degradation.
    7. If an on-premises data gateway is part of the architecture, monitor its CPU and memory utilization during the test. A gateway that is under-provisioned can become a bottleneck, limiting throughput regardless of the source database's capacity.
- **Success Criteria:** For an optimized system, the datasourceConnectionThrottleTimeMs metric should be zero for all queries under typical load. The "Maximum connections" setting should be configured to a value that maximizes throughput without overwhelming the source database.

Test 3: DAX and Power Query Functional Limitations

DirectQuery mode imposes significant restrictions on both Power Query transformations and DAX functions, as not all operations can be efficiently translated into native source queries.

- **Methodology:** This test involves validating that all required data transformations and business calculations are supported and performant.

- **Actions:**
    1. **Validate Query Folding:** In the Power Query Editor, for each query connected to the DirectQuery source, navigate to the final step in the "Applied Steps" pane. Right-click and check if the "View Native Query" option is enabled. If it is greyed out, it signifies that a transformation step has "broken" query folding. This means the transformation cannot be translated to SQL and will be executed inefficiently by Power BI's engine, which should be avoided at all costs in DirectQuery.
    2. **Test DAX Function Support:** Attempt to use complex DAX functions, particularly time-intelligence functions like SAMEPERIODLASTYEAR. Many of these are not supported in DirectQuery unless a proper, well-structured date dimension exists in the source database and is marked as a date table in the model.
    3. **Test Calculated Columns:** Attempt to create a calculated column that references a column from a different table. This will fail, confirming that calculated columns in DirectQuery are limited to intra-row logic (they can only reference other columns within the same table).
- **Success Criteria:** All data preparation logic must be successfully folded into the native source query. All required DAX measures must be supported and translated into efficient native queries. Any business logic that cannot be implemented due to these limitations must be pushed upstream and materialized in the data source itself, for example, within a SQL view or as a physical column in a table.

The paradigm for optimizing DirectQuery performance shifts away from fine-tuning DAX measures and toward managing the volume and efficiency of the native queries sent to the data source. A common scenario involves a developer creating a report with numerous visuals, assuming that because each individual visual loads quickly in isolation, the full report page will also be fast. However, when the page is loaded, the performance is poor. The root cause is often not the complexity of any single query but the sheer concurrency of all queries being sent at once. The default limit of 10 maximum connections per data source for a Pro license means that if a report page has 15 visuals, 5 of those visuals' queries will be queued, waiting for one of the first 10 to complete. This queuing delay is invisible in the DAX query duration for any single visual but is the dominant factor in the overall page load time. The Execution Metrics event in SQL Profiler, with its datasourceConnectionThrottleTimeMs metric, is the key to diagnosing this specific problem. The solution lies not in rewriting the DAX, but in either redesigning the report to have fewer visuals, increasing the connection limit if the source system and capacity allow, or implementing techniques to consolidate queries before they are sent to the source.

Test Plan for Composite Mode Semantic Models

Objective

The objective of testing Composite models is to evaluate the performance and integrity of a hybrid architecture that combines different storage modes. The test plan must focus on the interactions between source groups, particularly the performance of cross-source relationships, and validate the effectiveness of specific Composite model features like aggregations and model chaining.

Key Performance Indicators (KPIs)

- **Cross-Source Group Query Performance:** The response time for visuals that combine data from tables in different storage modes (e.g., Import and DirectQuery).
- **Aggregation Hit Rate:** The percentage of queries that are successfully answered by an in-memory aggregation cache instead of being sent to the DirectQuery source.
- **Impact of Limited Relationships:** The performance overhead and potential data integrity issues arising from joins across different source groups.

Test 1: Storage Mode and Relationship Performance

This test evaluates the core functionality of a composite model: querying across tables with different storage modes.

- **Methodology:** Design and benchmark queries that join tables across different source groups to understand the performance characteristics of Dual mode and limited relationships.
- **Actions:**
    1. Construct a model following best practices: configure smaller dimension tables as **Dual** mode and the large fact table as **DirectQuery** mode.
    2. Create a report page with a slicer based on a column from a Dual-mode dimension table and a visual (e.g., a bar chart) showing a measure from the DirectQuery fact table.
    3. Use DAX Studio's Server Timings to analyze the queries. When interacting with the slicer, the query populating its values should be served instantly from the in-memory cache of the Dual-mode table. When a value is selected, a new DirectQuery should be sent to the fact table, filtered by the selection. This demonstrates the "best of both worlds" performance.
    4. To test the impact of limited relationships, create a relationship between a table in **Import** mode and a table in **DirectQuery** mode. Build a visual that requires data from both. Measure the query time and compare it to a similar visual that only uses tables from a single source group. The performance degradation is due to the less efficient join logic handled by the Formula Engine for cross-source queries.
- **Success Criteria:** The use of Dual-mode dimensions should provide a fast, interactive filtering experience. The performance cost associated with cross-source group (limited) relationships should be quantified, and these relationships should be used sparingly and only when necessary.

Test 2: Aggregation Effectiveness

Aggregations are a critical performance-tuning feature for composite models, allowing high-level queries to be served from a fast in-memory cache while detailed data remains in the DirectQuery source.

- **Methodology:** Implement a user-defined aggregation over a large DirectQuery fact table and use tracing tools to verify that queries are correctly routed to either the aggregation cache or the DirectQuery source.
- **Actions:**
    1. In Power Query, create a new summary table by grouping a large DirectQuery fact table by key dimensions (e.g., Date, Product Category, Region) and aggregating measures (e.g., SUM of Sales). Set the storage mode of this new table to **Import**.
    2. In the model view, right-click the aggregation table and select "Manage

aggregations." Configure the mapping between the columns in the aggregation table and the corresponding columns and summarizations in the detail (DirectQuery) table.
3. Create a summary-level report page with visuals that request data at a grain that can be satisfied by the aggregation table (e.g., Sales by Month and Region).
4. Use DAX Studio or SQL Profiler to monitor the backend queries while interacting with this summary page. No DirectQuery queries should be sent to the source. The trace will show that the queries are being satisfied by the VertiPaq engine, indicating a successful "aggregation hit".
5. Create a drill-through page that displays transactional details from the DirectQuery fact table. Verify that drilling through from a summary visual correctly triggers a DirectQuery to the source to retrieve the detailed data.

- **Success Criteria:** Summary-level visuals should exhibit near-instantaneous performance, characteristic of Import mode. A high aggregation hit rate should be observed for all high-level queries. DirectQuery queries should only be generated when a user explicitly requests a level of detail not present in the aggregation cache.

Test 3: Chaining and Governance Implications

A powerful feature of composite models is the ability to connect to a published Power BI semantic model using DirectQuery, effectively "chaining" models together. This enables a hub-and-spoke BI architecture but introduces performance and governance considerations.

- **Methodology:** Test the creation, performance, and maintainability of a chained composite model.
- **Actions:**
    1. Publish a well-structured "hub" semantic model to the Power BI service.
    2. In a new Power BI Desktop file, connect to this published semantic model. A message will appear in the status bar: "Connected live...". Click the link to "Make changes to this model". This action converts the live connection into a DirectQuery connection and creates a local model, allowing for further modifications.
    3. Import a new local data source, such as an Excel file containing departmental targets (the "spoke" data).
    4. Create a relationship between the local Excel table and a table from the remote "hub" semantic model.
    5. Build visuals that combine data from both the hub and spoke sources and measure their performance.
    6. Document the inherent limitations observed during development, such as the inability to view the Power Query transformations or underlying DAX measures of the remote hub model, and the hard limit of a three-model chain.
- **Success Criteria:** The chained model must be functional and performant. Critically, the governance challenges—such as the risk of the spoke model breaking if the hub model is changed, and the lack of transparency into the hub model's logic—must be documented as key risks to be managed through communication and documentation protocols.

The introduction of composite models fundamentally enables a federated, or "hub-and-spoke," BI architecture within an enterprise. This is not merely a technical capability but a strategic approach to balancing centralized governance with decentralized, self-service analytics. In this paradigm, a central BI team can build and publish certified, robust "hub" semantic models that represent the single source of truth for key business domains like sales or finance.

Departmental analysts or power users can then create their own "spoke" reports by connecting to these certified hubs via DirectQuery in a composite model, and augmenting them with their own local data, such as departmental budgets from an Excel file. This architecture empowers business users with flexibility while ensuring they are building upon a foundation of trusted, governed data. However, this creates a dependency chain. If the owner of the hub model renames a table or a measure, any downstream spoke models that reference it will break upon their next refresh. Furthermore, the analyst building the spoke model has no visibility into the complex DAX logic or data transformations within the black-box hub model, making it difficult to debug unexpected results. Therefore, testing a composite model architecture must extend beyond technical performance benchmarks to include the validation of governance processes. The tests must answer questions like: What is the change management protocol for the hub model? How is the logic within the hub model documented and made transparent to spoke model creators? This demonstrates that a successful composite model implementation is as much about robust governance and communication as it is about query performance.

Advanced Testing: Concurrency and Capacity Load Simulation

Single-user performance testing in Power BI Desktop is a necessary but insufficient step for validating an enterprise-grade solution. To truly understand how a report and its underlying capacity will behave in a production environment, it is essential to simulate a realistic, multi-user workload. This advanced testing phase moves from the developer's machine to the Power BI service to measure performance under stress.

Objective

The objective of this phase is to assess the scalability of the Power BI solution by simulating concurrent user activity. This testing aims to identify the capacity's breaking point, understand the impact of user load on report interactivity, and ensure the chosen capacity SKU is sufficient for the expected user base.

Methodology

The recommended approach for this level of testing is to use the official Microsoft Power BI Capacity Load Assessment Tool. This is a PowerShell-based solution that automates the process of opening multiple browser instances to generate a realistic load against a published Power BI report.

Test 1: Environment and Tool Setup

A dedicated and properly configured environment is critical for obtaining accurate load testing results.

- **Prerequisites:** The machine running the test must be configured correctly. This includes running PowerShell in an elevated (Administrator) mode, setting the script execution policy to Unrestricted to allow the tool's unsigned scripts to run, and installing the required MicrosoftPowerBIMgmt PowerShell modules.
- **Capacity Setup:** A dedicated Power BI Premium or Microsoft Fabric capacity must be

provisioned in Azure specifically for testing purposes. It is critical not to run load tests against a production capacity, as this could impact real users. The workspace containing the semantic models and reports to be tested must be assigned to this dedicated test capacity.

- **Monitoring Setup:** The Microsoft Fabric Capacity Metrics app is the primary tool for monitoring the health of the capacity during the test. This app must be installed by a capacity administrator and configured to monitor the dedicated test capacity ID. It provides detailed telemetry on CPU utilization, memory pressure, and overload events.

Test 2: Load Test Configuration and Execution

Simulating a realistic workload involves more than just repeatedly opening a report. The test must mimic actual user behavior to bypass caching mechanisms and generate a genuine load on the capacity's query engines.

- **Test Scenarios:** A simple load test that repeatedly opens the default view of a report will largely hit the Power BI service's cache, resulting in an artificially low load on the capacity. To conduct a meaningful stress test, the tool must be configured to simulate users interacting with the report. This involves programming the test script to apply different filter values, change slicer selections, and navigate through bookmarks for each simulated user. This ensures that unique DAX queries are generated for each interaction, bypassing the cache and forcing the capacity to perform actual computations.
- **PowerShell Script Execution:** The load test is initiated by running the Initiate_Load.ps1 script. The script will interactively prompt the user for necessary information, including:
    1. Authentication credentials for the Power BI service.
    2. The workspace and report to be tested.
    3. Filter parameters (Table, Column, Min/Max values) that the script will use to generate randomized filter contexts for each query, effectively circumventing the cache.
    4. The number of concurrent users to simulate, which corresponds to the number of browser instances the script will launch.
- **Scaling the Test:** It is advisable to conduct the load test in an incremental fashion. Begin with a small number of concurrent users (e.g., 5 or 10) and run the test for a sustained period. After analyzing the results, increase the user load in subsequent test runs until performance degradation is observed or the capacity reaches its limits.

Test 3: Analysis of Results in the Capacity Metrics App

After each load test run, the primary source for analysis is the Microsoft Fabric Capacity Metrics app. This app visualizes the performance telemetry collected from the capacity during the test.

- **Methodology:** The analysis focuses on identifying signs of resource exhaustion and throttling within the capacity.
- **Key Metrics to Analyze:**
    1. **CPU Utilization:** The "CPU Over Time" chart is the most critical visual. It displays the total CPU seconds consumed in 30-second evaluation windows. If the total CPU usage consistently exceeds 100% of the capacity's limit, the capacity is in a state of overload, and the Power BI service will begin to throttle operations, leading to increased wait times for users.
    2. **Interactive vs. Background Operations:** The metrics app distinguishes between

interactive operations (on-demand queries from user report interactions) and background operations (scheduled data refreshes). A key best practice for capacity management is to ensure that background operations do not consume an excessive portion of the CPU, leaving insufficient resources for interactive user queries. A common guideline is to keep background CPU load below 50% of the total capacity.

3. **Overload Events:** The app explicitly reports when the capacity enters an "overload" state. Any occurrence of overload during a test indicates that the simulated user load exceeded the capacity's resources, resulting in a degraded user experience.
4. **Memory Usage and Eviction:** For tests involving Import models, it is important to monitor memory usage. The metrics app can show if and when semantic models are being evicted from memory due to pressure. A high rate of dataset eviction will lead to slow performance, as users must wait for models to be reloaded into memory before their queries can be served.

- **Success Criteria:** A successful load test demonstrates that the capacity can handle the target number of concurrent users without entering a sustained overload state. Report response times, as measured by the load testing tool, should remain within acceptable service-level agreements (SLAs) throughout the test duration.

To effectively plan and interpret these tests, it is essential to understand the specific resource limits of the target capacity SKU. The choice of SKU directly impacts the available memory, CPU power, and, critically for DirectQuery models, the number of concurrent connections allowed.

SKU	v-Cores	Max Memory (GB)	Model Refresh Parallelism	Max Concurrent DirectQuery Connections
F64 / P1	8	25	40	50
F128 / P2	16	50	80	75
F256 / P3	32	100	160	100
F512 / P4	64	200	320	200

Data sourced from.

Understanding these hard limits is vital. For example, if a concurrency test on a DirectQuery model begins to fail with 15 simultaneous users while running on a Pro license (shared capacity), this table provides the immediate explanation: shared capacity is limited to 10 concurrent DirectQuery connections per semantic model. This data provides the concrete evidence needed to justify an upgrade to a higher capacity tier, such as an F64/P1, which supports up to 50 concurrent connections, thereby directly linking the test results to strategic infrastructure decisions.

Synthesis and Strategic Recommendations

The comprehensive testing of Import, DirectQuery, and Composite models yields a clear set of performance characteristics and architectural trade-offs. The synthesis of these results provides a strategic framework for selecting the optimal semantic model architecture based on specific business requirements for data volume, data freshness, query complexity, and governance.

Comparative Analysis of Storage Modes

The following decision matrix distills the findings from the detailed test plans into a high-level, actionable guide. It serves as a quick-reference tool for developers to make an initial assessment of the most suitable storage mode for a new project.

Decision Factor	Import Mode	DirectQuery Mode	Composite Mode
Data Volume	**Limited:** Ideal for models under 10GB (Pro) or up to the capacity limit (Premium).	**Very Large:** Suitable for datasets of any size, as data remains in the source.	**Flexible:** Best for large fact tables (DirectQuery) combined with smaller dimension tables (Import/Dual).
Data Freshness	**Latent:** Data is only as current as the last scheduled refresh.	**Real-Time:** Queries return the latest data directly from the source.	**Hybrid:** Enables real-time data for DirectQuery partitions while maintaining cached data for Import partitions.
Query Performance	**Very Fast:** In-memory querying via the VertiPaq engine provides the best interactive performance.	**Variable:** Performance is entirely dependent on the source database, network, and query concurrency.	**Hybrid:** Performance is fast for queries hitting the cache (Import/Dual tables, aggregations) but variable for queries hitting the DirectQuery source.
Query Complexity	**High:** Supports the full range of Power Query (M) and DAX functions, offering maximum design flexibility.	**Limited:** Power Query transformations and DAX functions are restricted to what can be translated to the native source query language.	**Mixed:** Full capabilities on Import tables; limited capabilities on DirectQuery tables. Cross-source queries add complexity.
Concurrency	**High:** Excellent for high user concurrency as queries are served from the highly optimized in-memory cache.	**Limited:** Constrained by the "Maximum connections per data source" setting and the capacity of the source system to handle parallel queries.	**Mixed:** Concurrency for queries against imported portions is high; concurrency against DirectQuery portions is limited.
Development Complexity	**Low to Medium:** The most straightforward development experience, focused on data modeling and DAX within a single environment.	**Medium to High:** Requires strong skills in source database optimization and an understanding of query folding and DAX limitations.	**High:** The most complex development experience, requiring careful management of storage modes, limited relationships, and potential performance pitfalls.

Decision Factor	Import Mode	DirectQuery Mode	Composite Mode
Governance Model	**Centralized:** Well-suited for creating self-contained, certified semantic models.	**Federated:** Relies on the governance and security of the underlying data source.	**Hub-and-Spoke:** The primary enabler for a federated architecture where governed "hub" models are extended by departmental "spoke" models.

A Decision Framework for Model Selection

Based on the analysis, a logical decision-making process emerges for selecting the appropriate storage mode.

1. **Start with Import Mode by Default:** For the vast majority of analytical scenarios, Import mode should be the default choice. It offers the best performance, the greatest design flexibility with unrestricted access to the full Power Query and DAX function libraries, and the most straightforward development experience.
2. **Use DirectQuery Only When Necessary:** Transition to DirectQuery mode only when faced with a clear constraint that Import mode cannot satisfy. The two primary drivers for using DirectQuery are when the data volume is too large to be practically imported and refreshed within the available capacity limits, or when there is a strict business requirement for near real-time data that cannot be met by scheduled refreshes. When choosing DirectQuery, developers must accept the trade-offs of reduced functional capabilities and performance dependency on the external source system.
3. **Leverage Composite Models for Hybrid Scenarios:** Composite models should be employed strategically to solve specific hybrid challenges. The most effective use cases are augmenting a large DirectQuery fact table with smaller, imported dimension tables to improve performance, and implementing aggregations to provide Import-level speed for summary queries while retaining DirectQuery access to granular detail.

Best Practices for a Hybrid Enterprise Architecture

For a large organization, a successful BI strategy will not rely on a single storage mode but will instead leverage all three to create a scalable, governable, and high-performing ecosystem.

- **Promote a Hub-and-Spoke Architecture:** The most effective model for enterprise BI involves creating a series of certified, foundational "hub" semantic models. These hubs, often built in Import mode for performance or as highly optimized DirectQuery models over a data warehouse, serve as the governed single source of truth for key business areas. Departmental analysts and power users are then empowered to create their own "spoke" reports. They achieve this by using Composite models to connect via DirectQuery to the certified hubs, which they can then augment with their own local or departmental data. This federated approach masterfully balances the need for centralized data governance and quality with the agility of decentralized, self-service analytics.
- **Mandate Aggregations for Large-Scale Models:** For any significant DirectQuery or Composite model intended for broad use, aggregations should be considered a mandatory architectural component, not an optional performance tweak. By creating pre-summarized tables in Import mode that sit on top of large DirectQuery fact tables, developers can ensure that the majority of high-level, interactive report queries are served

with the speed of an in-memory model. This reserves the slower, more resource-intensive DirectQuery queries for intentional drill-through and detailed analysis, providing an optimal user experience.

- **Institute Continuous Monitoring and Optimization:** Performance testing is not a one-time event conducted at the end of a project. It is a continuous process. As data volumes grow, the number of users increases, and report complexity evolves, the performance characteristics of a solution will change. Organizations must establish a process for ongoing monitoring of key metrics using tools like the Microsoft Fabric Capacity Metrics app. Regular reviews should be conducted to identify performance degradation, and models should be periodically re-evaluated and optimized to ensure they continue to meet business SLAs.

Works cited

1. Semantic model modes in the Power BI service - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/connect-data/service-dataset-modes-understand 2. Power BI Composite Models: When to Use DirectQuery vs Import ..., https://medium.com/techsutra/power-bi-composite-models-when-to-use-directquery-vs-import-mode-the-ultimate-2025-guide-53d270e7e422 3. Understanding Power BI Engine Architecture: A Deep Dive | by Avishek Ghosh (AV_DEVS), https://medium.com/microsoft-power-bi/understanding-power-bi-engine-architecture-a-deep-dive-c162dbcfa895 4. DirectQuery Connection in Power BI; How does it work? Limitations and Advantages, https://radacad.com/directquery-connection-in-power-bi-how-does-it-work-limitations-and-advantages/ 5. Live SQL Server Data in Power BI: Using DirectQuery - Learning Tree, https://www.learningtree.com/blog/live-sql-server-data-power-bi-using-directquery/ 6. Composite model guidance in Power BI Desktop - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/guidance/composite-model-guidance 7. Use composite models in Power BI Desktop - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/transform-model/desktop-composite-models 8. Level Up Your Reports: Mastering Power BI Performa... - Microsoft ..., https://community.fabric.microsoft.com/t5/Power-BI-Community-Blog/Level-Up-Your-Reports-Mastering-Power-BI-Performance-Analyzer/ba-p/4768359 9. Use Performance Analyzer to examine report element performance ..., https://learn.microsoft.com/en-us/power-bi/create-reports/desktop-performance-analyzer 10. Identify Power BI Performance Bottlenecks Using the Performance ..., https://b-eye.com/blog/power-bi-performance-analyzer-tutorial/ 11. Microsoft Power BI Performance Best Practices - BI Courses, https://bicourses.azurewebsites.net/eBooks/Microsoft%20Power%20BI%20Performance%20Best%20Practices.pdf 12. Optimizing Power BI - Measuring Performance using DAX Studio - YouTube, https://www.youtube.com/watch?v=jpGmTXkXq1A 13. How to perform concurrency testing of Power BI reports - Insight - Keyrus, https://keyrus.com/us/en/insights/how-to-perform-concurrency-testing-of-power-bi-reports 14. How to see Relational SQL generated by Power Bi Report., https://community.powerbi.com/t5/Desktop/How-to-see-Relational-SQL-generated-by-Power-Bi-Report/td-p/1373142 15. Troubleshoot DirectQuery models in Power BI Desktop - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-directquery-troubleshoot 16.

How to Optimize Power BI Data Models for Large Datasets [Advanced Step-by-Step Guide], https://b-eye.com/blog/optimize-power-bi-data-models-large-datasets/ 17. Power BI Best Practices - by Sam Campitiello - Medium, https://medium.com/@sam.campitiello/power-bi-best-practices-17896855704a 18. Power BI Performance Optimization Best Practices: 10X Faster Reports for the Enterprise, https://b-eye.com/blog/power-bi-performance-optimization-best-practices/ 19. Data refresh in Power BI - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/connect-data/refresh-data 20. How to Implement Incremental Refresh in Power BI - CloudFronts, https://www.cloudfronts.com/blog/power-bi/how-to-implement-incremental-refresh-in-power-bi/ 21. Configure incremental refresh for Power BI semantic models - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/connect-data/incremental-refresh-configure 22. 5 Best Practices of Performance Analyzer in Power BI - JourneyTeam, https://www.journeyteam.com/resources/blog/5-best-practices-of-performance-analyzer-in-power-bi/ 23. Optimizing Power BI Performance for Large Datasets - Abbacus Technologies, https://www.abbacustechnologies.com/optimizing-power-bi-performance-for-large-datasets/ 24. Optimizing Power BI DirectQuery performance - Practical guide part 1 - Max Wikström, https://www.maxwikstrom.se/performance/mastering-directquery-in-power-bi-a-deep-dive-into-performance-optimization-part-1/ 25. Directquery Performance And Each visual executing query in SQL Server - Power BI forums, https://community.powerbi.com/t5/Desktop/Directquery-Performance-And-Each-visual-executing-query-in-SQL/td-p/2349703 26. DirectQuery model guidance in Power BI Desktop - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/guidance/directquery-model-guidance 27. DirectQuery in Power BI - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-directquery-about 28. Direct Query Limitations in Power BI - Pragmatic Works, https://pragmaticworks.com/blog/direct-query-limitations-in-power-bi 29. Direct Query Calculated columns and rls - supported DAX functions - Power BI forums, https://community.powerbi.com/t5/Desktop/Direct-Query-Calculated-columns-and-rls-supported-DAX-functions/td-p/2839377 30. DirectQuery Limitation - Microsoft Fabric Community - Power BI forums, https://community.powerbi.com/t5/Desktop/DirectQuery-Limitation/td-p/2613827 31. Power BI Best Practices from the Field: Lessons from Enterprise Deployments - Datalere, https://datalere.com/articles/power-bi-best-practices-from-the-field-lessons-from-enterprise-deployments 32. Power BI Enterprise Project Good and Best Practices, https://sqlserverbi.blog/wp-content/uploads/2019/09/power-bi-enterprise-good-and-best-practices.pdf 33. Power BI Embedded Analytics Capacity Planning - Microsoft Learn, https://learn.microsoft.com/en-us/power-bi/developer/embedded/embedded-capacity-planning 34. Why should I complete Power BI Performance Load Testing & initial ..., https://www.fourmoo.com/2025/07/09/why-should-i-complete-power-bi-performance-load-testing-initial-setup/ 35. PowerBI-Tools-For-Capacities/LoadTestingPowerShellTool/Readme.md at master - GitHub, https://github.com/microsoft/PowerBI-Tools-For-Capacities/blob/master/LoadTestingPowerShellTool/Readme.md 36. Power BI Embedded: Stress Testing & Capacity Planning - Data on Wheels - Kristyna Ferris & Steve Hughes, https://dataonwheels.wordpress.com/2022/02/22/power-bi-embedded-stress-testing-capacity-planning/ 37. Matt Lakin - Get a Load of this - Realistic Load Testing for Power BI, explained! - YouTube, https://www.youtube.com/watch?v=FUQLivEgaeY 38. Why load testing Power BI is important - Chris Webb's BI Blog,

https://blog.crossjoin.co.uk/2023/04/23/why-load-testing-power-bi-is-important/ 39. Performance Testing with Power BI for Enterprise Management - JigNect, https://jignect.tech/project/streamlining-enterprise-management-a-performance-testing-journey-with-power-bi-dashboards/