PBI Grid: A Dashboard as Code Tool for Enhancing Maintainability and GovBR Compliance in Power BI Reports

Fernando Maia da Mota
UFMS
Campo Grande, Brazil
fernando.mota@ufms.br

Eos Xavier
UFMS
Campo Grande, Brazil
eos.xavier@ufms.br

Debora Paiva
UFMS
Campo Grande, Brazil
debora.paiva@ufms.br

Edson Takashi Matsubara
UFMS
Campo Grande, Brazil
edson.matsubara@ufms.br

Ricardo Marcacini
ICMC/USP
São Carlos, Brazil
ricardo.marcacini@icmc.usp.br

Marcelo Turine
UFMS
Campo Grande, Brazil
marcelo.turine@ufms.br

Vanessa Borges
UFMS
Campo Grande, Brazil
vanessa.a.borges@ufms.br

Abstract

Power BI is widely used by Brazilian federal agencies to publish public-sector dashboards. Despite the existence of the Brazilian Government Digital Standard (GovBR), many of these dashboards do not comply with its visual identity, accessibility, and interface consistency requirements. The Power BI Enhanced Report Format (PBIR) exposes pages and visuals as JSON files but still relies on absolute pixel coordinates and lacks declarative layout abstractions. This paper presents PBI Grid, an open-source command-line tool that compiles declarative YAML specifications into PBIR artifacts. The tool combines three main elements: a declarative language based on 12-column grids, a deterministic layout engine that resolves declarative specifications into PBIR-compatible coordinates, and an extensible theming system with a reference `govbr` package that codifies the GovBR visual identity for Power BI dashboards. PBI Grid was used to reconstruct a production CAPES dashboard under the GovBR standard; on a bundled, redistributable synthetic example (`countries_population`), it reduces the layout specification by approximately 89% while remaining amenable to code review and continuous integration. An `extract` command enables incremental adoption by reconstructing YAML specifications from existing PBIR reports.

Demo video: https://zenodo.org/records/21462034?preview_file=PBI-Grid.Video.mp4

Keywords

Dashboard-as-Code, Power BI, PBIR, Design Systems, Public-Sector Dashboards, GovBR, Tools

1 Introduction

Power BI has been widely adopted by Brazilian federal agencies for publishing transparency dashboards, operational indicators, and public policy outcomes, becoming one of the main presentation layers for government data [6, 14, 23]. However, many of these dashboards do not comply with the Brazilian Government Digital Standard (GovBR) [9], which standardizes the visual identity and accessibility of federal digital services. The result is a fragmented landscape of public dashboards with inconsistencies in palette, typography, and navigation that compromise user experience.

In 2024, Microsoft introduced the Power BI Enhanced Report Format (PBIR) [17], which became the Power BI service default in January 2026 and the Desktop default with the May 2026 release. PBIR exposes pages, visuals, and configurations as hierarchically organized JSON files [16], making reports editable and version-controllable as code. Yet PBIR still positions visuals via absolute coordinates on a fixed canvas, providing no declarative layout abstraction comparable to CSS Grid [8] or Bootstrap [3], so programmatic generation remains tied to manual geometric calculations.

This paper presents PBI Grid, an open-source command-line tool that compiles declarative YAML specifications based on 12-column grids into PBIR `.Report` artifacts. The tool combines a layout engine that resolves grid spans into PBIR coordinates with bit-for-bit determinism, an extensible theming system with a reference `govbr` package codifying GovBR, and an `extract` command that reconstructs YAML from existing PBIR reports for incremental adoption. The paper demonstrates PBI Grid on a production CAPES dashboard; on the bundled, redistributable `countries_population` example, the YAML specification is 89% more compact than the hand-authored PBIR source it was extracted from.

The remainder of this paper is organized as follows: Section 2 presents background concepts; Section 3 discusses related work; Section 4 describes PBI Grid; Section 5 demonstrates it on a GovBR case study; and Sections 6 and 7 report limitations, conclusions, and future work.

2 Background

PBIR is Microsoft's metadata serialization format for Power BI reports. A `.Report` folder contains files such as `report.json` and `pages.json`, in addition to a directory for each report page containing its corresponding `page.json` and visuals. Each visual is represented by a `visual.json` file that stores absolute coordinates and visual configurations. Since PBIR is based on human-readable JSON files, it enables external tools to read, modify, and programmatically generate reports.

PBI Grid builds on three additional concepts. Dashboard as Code (DaC) is the authoring approach: dashboards are described through versionable text files that compile into renderable artifacts, instead of being constructed via interactive drag-and-drop in Power BI Desktop, following principles similar to those of Infrastructure as Code [18]. *Design tokens* [7] are named values for visual properties (colors, sizes, interaction states) declared once and reused across visuals, allowing brand or palette changes to propagate via a single edit; the reference `govbr` package provides such tokens for GovBR.

Internally, the generation process follows a model-transformation pipeline [5], in which declarative specifications are compiled into PBIR artifacts compatible with the Power BI ecosystem.

3 Related Work

Declarative compilation for data visualization has already been explored in different visualization tools and models. Polaris [22] compiles tabular specifications into relational queries, while Vega-Lite [20] transforms declarative specifications into executable Vega representations, following concepts from Wilkinson's *grammar of graphics* [26] and its layered formulation proposed by Wickham [25]. D3 [4], in turn, binds data directly to native runtime representations. These approaches share the idea of separating visualization definition from rendering, enabling structural reuse, programmatic generation, and specification versioning [11], principles aligned with the DaC approach adopted by PBI Grid.

In declarative dashboard generation, Tundo et al. [24] use metamodels to compose dashboards from dynamic sets of indicators. However, their focus is on systems-of-systems monitoring environments, whereas PBI Grid focuses on declarative layout generation, systematic application of the GovBR visual identity, and PBIR artifact compilation for Power BI reports. PBI Grid extends the DaC approach to the PBIR ecosystem, where layout definition has historically remained dependent on absolute coordinates.

Within the PBIR ecosystem, several tools already operate directly on the format. The PBIR utilities `pbir-utils`[1], `pbir-cli`[2], `pbi-tools`[3], and Microsoft's `fabric-cicd`[4] expose CLI or programmatic interfaces for validation, bulk modification, deployment, and parameter substitution, but treat layout as fixed input rather than as a declarative specification. Microsoft's `semantic-link-labs`[5] library exposes a `create_report_from_reportjson` function that creates a Power BI report from a raw `report.json` dictionary, providing programmatic report creation but without a higher-level layout abstraction or theming system. TemplateMechanics[6] ships a collection of AI prompts that gives coding assistants such as GitHub Copilot and Claude Code domain knowledge of Power BI's textual source formats (PBIP, TMDL, and PBIR), enabling them to generate and modify reports from natural-language prompts; because the generation happens inside an external large language model rather than in the tool itself, the resulting changes are non-deterministic and require manual auditing. UI-based template toolkits such as Numerro[7] provide layout templates inside Power BI Desktop, but operate above the PBIR layer and do not expose a programmatic interface.

Table 1 characterizes the PBIR-targeting ecosystem along six dimensions grouped into three pairs of increasing abstraction. The *baseline pair* combines *PBIR ops* (direct reading or modification of PBIR files) and *Automatable* (a CLI or programmatic API), establishing the minimum support for code-based Power BI workflows. The *compiler pair* combines *PBIR gen* (generation of PBIR artifacts from a specification) and *Spec* (the specification format, such as YAML or natural language), distinguishing compilation from operations on hand-authored input. The *design-system pair* combines *Theme* (a programmable theming system) and *Extract* (reverse extraction of specifications from PBIR), capturing the standardization and round-trip capabilities required to adopt a visual identity such as GovBR. The ecosystem is well populated at the baseline but thins as abstraction rises: only PBI Grid and `semantic-link-labs` generate PBIR directly (TemplateMechanics does so only indirectly, through an external LLM), and only PBI Grid covers all three pairs. PBI Grid also provides the bit-for-bit deterministic generation discussed in Section 4.1, a property that none of the surveyed tools documents.

Table 1: Feature comparison across PBIR-targeting tools along layout-generation and theming dimensions.

Tool	PBIR ops	PBIR gen	Spec	Theme†	Extract‡	Auto.
`pbir-utils`	Yes	No	n/a	No	No	Yes
`pbir-cli`	Yes	No	n/a	No	No	Yes
`pbi-tools`	Yes	No	n/a	No	No	Yes
`fabric-cicd`	Yes	No	n/a	No	No	Yes
`semantic-link-labs`	Yes	Yes§	JSON	No	No	Yes
TemplateMechanics	Yes*	Yes*	NL*	No	No	No
PBI Grid	**Yes**	**Yes**	**YAML**	**Yes**	**Yes**	**Yes**

† Programmable token system, not a fixed selection of predefined themes. ‡ Spec reconstruction from PBIR, not PBIX extraction. § From a raw `report.json` dict, not a higher-level abstraction. * Indirect via external LLM (e.g., GitHub Copilot, Claude Code).

The maintainability promise of PBI Grid also connects to software engineering research beyond visualization. Interview studies with industry practitioners report that techniques for testing and maintaining infrastructure-as-code artifacts remain scarce [10], and empirical accounts of model-driven engineering describe the discipline that PBI Grid supports by construction: the generated artifact is treated as derived, and maintenance happens in the specification, followed by regeneration [13]. The `extract` command addresses the round-trip engineering problem of keeping specification and artifact synchronized when both change, in the lightweight iterative style proposed for framework-specific modeling languages [1]. Deterministic generation makes the compiled artifacts verifiable in continuous integration pipelines, whose benefits are documented at scale [12]. Grid layouts, design tokens, declarative specifications, and round-trip extraction are established concepts individually; the contribution of PBI Grid lies in bringing them to the PBIR format, which offers no declarative layout abstraction of its own.

4 Tool Description

PBI Grid is an open-source tool released under the MIT License that receives as input a YAML layout specification together with an optional theme package, producing as output a complete PBIR `.Report` folder ready to be opened in Power BI. Each column defined in the YAML may represent one of three content types: a `component`, resolved by the engine into one or more visuals, such as navigation menus; a `name`, used to reference an existing PBIR visual by its identifier while preserving data bindings and modifying only its position; or a `visual` type used to create basic placeholders. When a `source` report is provided, the tool overlays the grid-computed positions onto the existing filters, formatting, and data bindings from the original report, regenerating only the layout.

[1] https://github.com/akhilannan/PBIR-Utils
[2] https://github.com/maxanatsko/pbir.tools
[3] https://pbi.tools
[4] https://github.com/microsoft/fabric-cicd
[5] https://github.com/microsoft/semantic-link-labs
[6] https://github.com/TemplateMechanics/pbi-pilot
[7] https://numerro.io

The tool is organized into two layers, illustrated in Figure 1. **Layer 1** is the **CORE**, a theme-agnostic deterministic compiler that runs four sequential stages (parser, grid engine, component resolver, and PBIR renderer) to transform a layout YAML into a PBIR `.Report` folder. **Layer 2** consists of optional **theme packages** that supply design tokens and reusable layout templates, applying visual identity to the visuals produced by the CORE. New visual types can be added by registering Python classes in the central component registry, allowing the component catalog to evolve without changes to the CORE. This separation decouples layout definition, PBIR generation, and visual identity, promoting dashboard reuse and standardization.

4.1 Layer 1: CORE Compilation Pipeline

The first stage of the CORE is the **parser**, responsible for reading the YAML, validating the specification against the layout schema, and resolving `ref:` entries into shared component definitions. Next, the **grid engine** converts the processed specification into a `Report` object: a tree of `Page` and `Visual` domain models that maps directly to the PBIR folder structure. A specification defines a canvas C with dimensions (W, H), a sequence of rows $R_1, \ldots, R_n$ each with height h_i, and for each row a sequence of columns $C_{i1}, \ldots, C_{im}$ each with span s_{ij} where $\sum_j s_{ij} = 12$. The 12-column choice follows Bootstrap and CSS Grid conventions. Absolute positions are computed as:

$$x(C_{ij}) = \frac{W}{12} \sum_{k<j} s_{ik}, \quad y(C_{ij}) = \sum_{k<i} h_k$$

Row-spanning components (such as a sidebar menu) count toward the 12-column budget of every row they cross. Each `Visual` carries a `Position` with horizontal and vertical offsets, width, height, stacking order, and a tab-order index emitted to support assistive technologies in keyboard navigation.

The third stage is the **component resolver**, which dispatches each component entry to its registered implementation. Components share a common interface in Listing 1:

Listing 1: Abstract component interface.

```
class Component(ABC):
    @abstractmethod
    def resolve(self, cell: Cell) -> list[Visual]: ...
```

The bundled `menu` component supports two-level hierarchical navigation: group entries render as non-clickable section headers, and their children as indented buttons; each leaf item carries an optional description that the renderer emits both as a tooltip and as the visual's `altText` for screen readers, providing assistive metadata intended to support accessibility audits.

The CORE bundles two additional components used by the `govbr` theme. The `header` component renders a brand bar with a title, an optional subtitle, and an accent stripe at the bottom, drawing colors and sizes from the active theme. The `footer` component renders a logo (registered in the PBIR resource package), columns of category links that point to either internal pages or external URLs, and a legal text strip at the base.

Finally, the **PBIR renderer** serializes the `Report` into the `.Report` folder structure. Visual identifiers are deterministic 20-character hexadecimal strings produced by truncating an MD5 hash of the page identifier, row identifier, visual type, and z-index; MD5 is used here non-cryptographically as a compact identifier scheme, with collision risk within a single report being negligible. Z-indexes are assigned in increments of one thousand, matching the spacing observed in PBIR reports generated by Power BI Desktop, so that generated visuals coexist cleanly with hand-edited ones when developers later mix the two. The engine balances strict determinism with PBIR's identity-preservation constraint: it mints a fresh hex name only when no existing PBIR name is supplied, and preserves supplied names verbatim, so regeneration does not break bookmarks, page-to-page selection sync, or measures written in DAX (Data Analysis Expressions, Power BI's formula language).

The four stages compose a pure function: the same inputs, the YAML and the optional source report, always produce the same PBIR output, with byte-identical visual identifiers and positions across executions. PBIR-format knowledge is concentrated in the renderer and in the `to_pbir`/`from_pbir` methods of the domain models, so schema drift in the format can be absorbed at the serialization layer rather than touching the engine, the component registry, or the theme packages.

4.2 Layer 2: Theme Packages

Theme packages decouple visual identity from the CORE pipeline. Each package lives under `themes/{name}/` and bundles two artifacts: a `tokens.yaml` file declaring design tokens (colors and interaction states per component), and one or more reusable layout templates. Tokens are loaded by the CORE before component resolution; when a token is absent, components fall back to Power BI's built-in theme expressions (`ThemeDataColor`), keeping the CORE functional without any active package.

The reference `govbr` package codifies the GovBR visual identity across four surfaces: navigation menu tokens (default, hover, and selected states); a GovBR-aligned chart palette applied uniformly to the visuals; the header rendered above the canvas; and the footer rendered below the canvas. The package ships two reusable layout templates: `report.yaml`, a multi-page report with a sidebar navigation menu spanning all rows, and `dashboard.yaml`, a single-page layout with a header, Key Performance Indicator (KPI) row, and chart grid.

The `scaffold` command provides an interactive entry point for theme adoption: it writes the `package:` field and shared component definitions (header, footer, menu) into the YAML, avoiding manual authoring.

4.3 Round-Trip via Extractor

The `extract` command inverts the generation pipeline: it clusters visuals by y-coordinate, infers rows and column spans by rounding visual widths to the nearest 12-column fraction, and produces a YAML file that regenerates the source bit-for-bit when widths already align with a 12-column fraction. The `--merge` flag preserves hand-authored component definitions while appending newly detected pages and folding newly added visuals into existing pages as rows, enabling incremental adoption.

Two scenarios motivate this round-trip design. The first is the migration of an existing hand-authored dashboard into declarative control: `extract` produces a starting YAML from which the

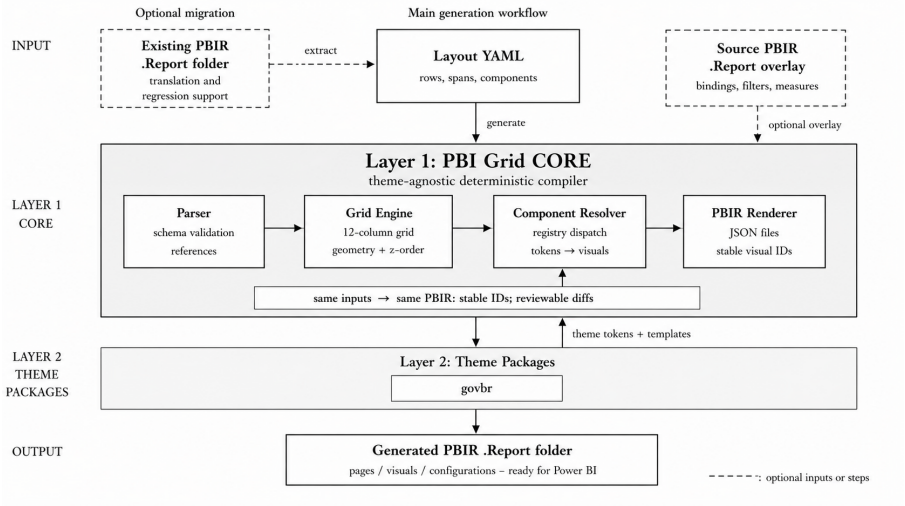

Figure 1: PBI Grid architecture and compilation pipeline.

dashboard becomes editable as code. The second is collaboration between authors who work in different modalities, for instance, when business analysts add pages directly in Power BI Desktop while developers maintain the YAML; in that case, `extract --merge` reads the modified report and folds any newly added pages or visuals into the specification, so prior declarative content is never overwritten. The round-trip is lossy on layouts whose visual widths do not align to a 12-column fraction: such widths snap to the nearest fraction during extraction, which can shift sub-twelfth-pixel offsets. These cases are uncommon in dashboards authored under PBI Grid but typical of legacy reports during the first extraction.

5 Demonstration

This section illustrates the declarative generation flow and demonstrates PBI Grid on the reconstruction of a production CAPES dashboard under the GovBR standard.

5.1 Installation and CLI Commands

PBI Grid is a Python 3.11+ package installed via `pip` from the source repository. Listing 2 shows the installation and the three CLI commands that drive the declarative dashboard pipeline: `extract`, `scaffold`, and `generate`.

Listing 2: CLI commands of PBI Grid.

```
# Clone repository
git clone https://github.com/OESNPG/pbi-grid.git
cd pbi-grid

# Install in editable mode
pip install -e .

# Extract YAML from an existing .Report
pbi-grid extract --report analytics/MyReport.Report

# Interactively configure govbr theme and shared components (
      header, footer, menu)
pbi-grid scaffold --layout dashboard_layout.yaml

# Generate PBIR from YAML
pbi-grid generate --layout dashboard_layout.yaml
```

Listing 3 shows a simplified excerpt of the `countries_population` reference layout. A `canvas:` block sets the rendering target (lines 2–4). The `shared.components` section declares three reusable components: a `page_header` with a title (lines 7–10), a `page_footer` with a logo (lines 11–14), and a `nav_menu` with hierarchical items (lines 15–22). The `pages:` section defines the report; the excerpted page reuses the header and menu through `ref:` entries and places a card and a bar chart in the remaining grid columns (lines 23–35).

Listing 3: Excerpt of the countries_population layout.

```
package: govbr
canvas:
  width: 1280
  height: 720
shared:
  components:
    page_header:
      span: 12
      component: header
      title: "Sample GovBR Theme"
    page_footer:
      span: 12
      component: footer
      logo_path: "govbr.png"
    nav_menu:
      span: 2
      component: menu
      items:
        - page: Home
        - page: Group
          items:
            - page: Item
pages:
  - display_name: Home
    rows:
      - height: 64
        cols:
          - ref: page_header
      - height: 120
        cols:
          - ref: nav_menu
          - span: 5
            visual: cardVisual
          - span: 5
            visual: barChart
```

5.2 Case Study: GovBR Dashboard Generation

We applied PBI Grid to reconstruct a production dashboard for a federal graduate research program supported by CAPES. The dashboard visualizes analytics on Brazilian graduate research programs using data collected through the Sucupira platform [21], CAPES's official data-collection system for graduate programs. The report spans eight pages with a persistent hierarchical navigation menu, Key Performance Indicator (KPI) cards, a Brazil map, bar and column charts, and a donut chart. This configuration falls within the strategic and operational dashboard categories identified by Sarikaya et al. [19] and incorporates composition patterns such as hierarchical navigation, KPI rows, and chart grids surveyed by Bach et al. [2]. Before adoption, the dashboard was authored manually in Power BI Desktop and used Power BI's default Fluent theme, which follows Microsoft's Fluent design system and is not aligned with the GovBR visual identity. Figure 2 contrasts the two states of the report. The original layout (1) sits on the raw Power BI canvas with the default Fluent palette, lacking any header, sidebar menu, or footer aligned with the federal visual identity. The regenerated layout (2) adds the GovBR header band (A), the hierarchical sidebar menu rendered by the `menu` component (B), the GovBR-aligned chart palette applied uniformly across visuals (C), and the GovBR footer (D); both panels show the same report and visuals.

Table 2 reports the line-count comparison measured on the bundled six-page `countries_population` example, a synthetic report representative of the CAPES dashboard structure; we measure the bundled example rather than the CAPES report because the latter cannot be redistributed (see Artifact Availability).

Table 2: Specification size measured on the `countries_population` example.

Artifact	Files	Lines
Hand-authored PBIR	19	1,664
PBI Grid YAML specification	1	176
Generated PBIR (all pages)	123	16,678

The YAML specification requires 176 lines in a single file, an 89% reduction from the 1,664-line hand-authored PBIR layout source; the YAML captures layout structure, while data bindings remain in the Power BI Semantic Model, the dataset layer that holds tables, relationships, and measures (optionally complemented by an overlaid source PBIR, Section 4). Beyond this reduction, we noted two qualitative aspects from authoring and regenerating the example. First, its full regeneration from the YAML specification ran in a few seconds, and repeated executions of `generate` on the same YAML in our environment produced byte-identical PBIR artifacts. Second, typical modifications, such as a new page, a different menu orientation, or a theme switch, became localized YAML edits rather than repetitive manual manipulation in Power BI Desktop.

More fundamentally, the CAPES specification captures the *intent* of the layout (*hierarchical menu spanning all pages, two KPI rows, a Brazil map, and chart breakdowns*) rather than thousands of lines of computed coordinates, and changes appear as line diffs reviewable in code reviews and CI pipelines. The hand-authored original, a sequence of drag-and-drops, leaves no comparable audit trail.

On the bundled six-page example, each shared component is defined once and regenerated on every page that references it: the header definition (7 lines in the full layout) expands into 30 `visual.json` files (4,163 lines) and the 18-line footer into 54 files (8,597 lines), both spanning all six pages, while the 8-line navigation menu expands into 10 files (2,456 lines) across the pages that carry it. Editing any of them is a single change to those few YAML lines instead of to dozens of generated files kept mutually consistent; likewise, adding one navigation entry is a single YAML line that the engine turns into the corresponding button on each page carrying the menu.

To illustrate incremental, mixed-modality adoption, we reproduced a common collaboration pattern on the report bundled with the tool. A report author adds a new column chart directly in Power BI Desktop, as an analyst unfamiliar with the YAML would, and a developer then runs `extract --merge` on the modified `.Report`. Matching visuals by their PBIR identifiers, the command folds the newly added chart into its page as an appended grid row and leaves the existing `shared` definitions and hand-authored rows untouched; regenerating from the merged specification reproduces the report with the analyst's chart preserved, its width snapped to the nearest grid fraction. The round-trip keeps specification and report synchronized without either modality overwriting the other.

To bring an existing dashboard into GovBR alignment, three steps suffice: (1) extract the current layout into YAML; (2) run `scaffold` on the extracted layout to select the `govbr` theme and configure shared components (header, footer, and navigation menu); and (3) run `generate` to apply the GovBR palette and navigation states to every generated component without re-authoring report content. The 3–5-minute demo video[8] walks through this workflow and the merge scenario on the `countries_population` example, which uses synthetic data.

6 Limitations

PBI Grid was evaluated on a single production dashboard built by the same team. Evaluation across different governmental contexts, with larger reports and distinct authors, remains open as future empirical work.

The current implementation does not yet cover AppSource custom visuals, complex *drill-through* chains, or embedded HTML. Performance on reports substantially larger than the eight-page case study is untested, and a systematic eMAG (the Brazilian e-government accessibility model) and WCAG conformance audit of the generated dashboard remains future work: the tool emits accessibility metadata (a tab-order index on every visual and `altText` on menu items), but verifying it against each success criterion, together with in-visual data-label contrast and screen-reader behavior, has not been done.

Theme expressivity is bounded by Power BI's native expression model: anything beyond what the runtime exposes, such as custom CSS, animations, or component templates, is unreachable. PBI Grid's token model deliberately avoids promising fidelity the runtime cannot deliver, codifying the parts of GovBR that Power BI can represent natively while keeping unsupported features explicit rather than silently degraded.

[8] https://youtu.be/3WtDYCx9z-k

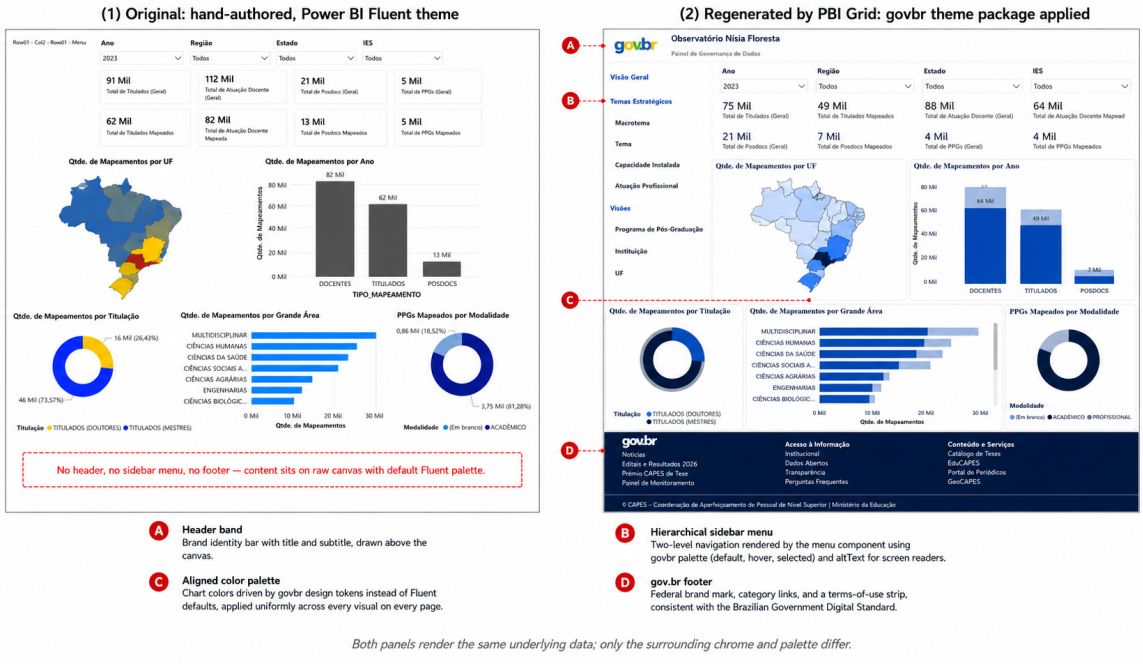

Figure 2: CAPES dashboard hand-authored in Power BI Desktop (1) versus regenerated by PBI Grid under the govbr theme (2).

7 Conclusion and Future Work

In this work, we presented PBI Grid, an open-source Dashboard-as-Code tool for Power BI that compiles declarative YAML specifications into PBIR artifacts aligned with the GovBR standard. Within the PBIR ecosystem, PBI Grid differs from the surveyed tools by combining declarative layout compilation, deterministic generation, and govbr theming in a single pipeline.

The approach targets maintainability as characterized in the ISO/IEC 25010 model [15]; visual standardization follows from the theming system. The use of YAML specifications reduces layout modifications to small textual changes that are automatically propagated across pages, while the govbr package enables the consistent application of the Brazilian federal government visual identity in Power BI dashboards.

The `countries_population` example provides a verifiable baseline: the declarative YAML specification is 89% more compact than the hand-authored PBIR source it was extracted from, while preserving auditability and textual versioning. The CAPES case study illustrated the same workflow in a real governmental context. The govbr package also offers a starting point that public institutions can audit and extend. Institutions that maintain Power BI dashboards without GovBR alignment can adopt PBI Grid incrementally, typically without manual reconstruction, for reports within the supported feature set.

Future work will address the coverage and evaluation gaps noted in Section 6, including the eMAG and WCAG conformance audit of generated dashboards and the application of PBI Grid across governmental contexts. Beyond these gaps, we plan to expand the component catalog beyond navigation menus (KPI rows, sidebar slicers, section titles), to investigate responsive *canvas* configurations in which a single specification generates PBIR artifacts adapted to different screen dimensions, and to integrate with `pbi-tools`, positioning declarative layout compilation as an upstream step in CI/CD pipelines already used for Power BI dashboards.

Artifact Availability

PBI Grid is open source under the MIT License. The release ships the sample layout at `examples/countries_population/` together with a Power BI Semantic Model, allowing the dashboard to be regenerated from a clean clone; the line counts in Table 2 can be reproduced via `wc -l` on the layout YAML, the source report's `definition/` folder, and the generated report's `definition/` folder. The CAPES report YAML is not redistributed for branding and data-source reasons. Permanent locations:

- Archived release (Zenodo DOI): https://doi.org/10.5281/zenodo.22135423
- Video demonstration: https://zenodo.org/records/21462034?preview_file=PBI-Grid.Video.mp4
- Source code repository: https://github.com/OESNPG/pbi-grid

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), Finance Code 001 and TED n. 14984/2024 - UFMS/CAPES; by the Fundação de Apoio ao Desenvolvimento do Ensino, Ciência e Tecnologia do Estado de Mato Grosso do Sul (FUNDECT), Call No. 42/2024 and by the Federal University of Mato Grosso do Sul (UFMS).

References

[1] Michal Antkiewicz and Krzysztof Czarnecki. 2006. Framework-Specific Modeling Languages with Round-Trip Engineering. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS) (Lecture Notes in Computer Science, Vol. 4199)*. Springer, 692–706. doi:10.1007/11880240_48

[2] Benjamin Bach, Euan Freeman, Alfie Abdul-Rahman, Cagatay Turkay, Saiful Khan, Yulei Fan, and Min Chen. 2023. Dashboard Design Patterns. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2023), 342–352. doi:10.1109/TVCG.2022.3209448

[3] Bootstrap Contributors. 2023. Bootstrap CSS Grid System (Version 5.3). https://getbootstrap.com/docs/5.3/layout/grid/. https://getbootstrap.com/docs/5.3/layout/grid/

[4] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2301–2309. doi:10.1109/TVCG.2011.185

[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice* (second ed.). Morgan & Claypool Publishers, San Rafael, CA. doi:10.2200/S00751ED2V01Y201701SWE004

[6] CAPES. 2026. Painéis de Monitoramento e Indicadores. https://www.gov.br/capes/pt-br/assuntos/painel-de-monitoramento-e-indicadores Accessed: 2026-05-12.

[7] Design Tokens Community Group. 2025. Design Tokens Format Module. W3C Community Group Draft Report. https://www.designtokens.org/tr/drafts/format/ First stable version, October 2025.

[8] Elika Etemad and Tab Atkins Jr. 2020. *CSS Grid Layout Module Level 1*. W3C Recommendation. W3C. https://www.w3.org/TR/css-grid-1/

[9] Governo Federal do Brasil – SECOM/SGD. 2024. Padrão Digital de Governo – Design System GovBR. https://www.gov.br/ds/home. https://www.gov.br/ds/home

[10] Michele Guerriero, Martin Garriga, Damian A. Tamburri, and Fabio Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 580–589. doi:10.1109/ICSME.2019.00092

[11] Jeffrey Heer and Michael Bostock. 2010. Declarative Language Design for Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 1149–1156. doi:10.1109/TVCG.2010.144

[12] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 426–437. doi:10.1145/2970276.2970358

[13] John Hutchinson, Jon Whittle, and Mark Rouncefield. 2014. Model-Driven Engineering Practices in Industry: Social, Organizational and Managerial Factors that Lead to Success or Failure. *Science of Computer Programming* 89 (2014), 144–161. doi:10.1016/j.scico.2013.03.017

[14] INEP. 2026. Painéis Educacionais. https://www.gov.br/inep/pt-br/acesso-a-informacao/dados-abertos/indicadores-educacionais Accessed: 2026-05-12.

[15] International Organization for Standardization. 2023. *ISO/IEC 25010:2023 — Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model*. ISO. https://www.iso.org/standard/78176.html

[16] Microsoft Corporation. 2025. *Create a Power BI report in enhanced report format (PBIR)*. Microsoft Learn. https://learn.microsoft.com/en-us/power-bi/developer/embedded/projects-enhanced-report-format Accessed: 2026-05-12.

[17] Microsoft Corporation. 2026. Power BI Desktop projects (PBIP) overview. https://learn.microsoft.com/en-us/power-bi/developer/projects/projects-overview

[18] Akond Rahman, Effat Farhana, and Laurie Williams. 2020. The 'as code' activities: development anti-patterns for infrastructure as code. *Empirical Software Engineering* 25, 5 (2020), 3430–3467. doi:10.1007/s10664-020-09841-8

[19] Alper Sarikaya, Michael Correll, Lyn Bartram, Melanie Tory, and Danyel Fisher. 2019. What Do We Talk About When We Talk About Dashboards? *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 682–692. doi:10.1109/TVCG.2018.2864903

[20] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 341–350. doi:10.1109/TVCG.2016.2599030

[21] Manoel Brod Siqueira. 2019. Sucupira – A Platform for the Evaluation of Graduate Education in Brazil. *Procedia Computer Science* 146 (2019), 247–255. doi:10.1016/j.procs.2019.01.081

[22] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (2002), 52–65. doi:10.1109/2945.981851

[23] Tesouro Nacional. 2026. Painel do Tesouro Nacional. https://www.tesourotransparente.gov.br/ Accessed: 2026-05-12.

[24] Alessandro Tundo, Chiara Castelnovo, Marco Mobilio, Oliviero Riganelli, and Leonardo Mariani. 2020. Declarative Dashboard Generation. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, Piscataway, NJ, USA, 215–218. doi:10.1109/ISSREW51248.2020.00075

[25] Hadley Wickham. 2010. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics* 19, 1 (2010), 3–28. doi:10.1198/jcgs.2009.07098

[26] Leland Wilkinson. 2005. *The Grammar of Graphics* (second ed.). Springer, New York, NY. doi:10.1007/0-387-28695-0