Automatic Slide Updating with User-Defined Dynamic Templates and Natural Language Instructions

Kun Zhou[1,2], Jiakai He[3], Wenmian Yang[2,†], Zhensheng Wang[1,2], Yiquan Zhang[2], Weijia Jia[2,4,†]
[1]School of Artificial Intelligence, Beijing Normal University, Beijing, PR China
[2]Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai, PR China
[3]Faculty of Arts and Sciences, Beijing Normal University, Zhuhai, PR China
[4]Beijing Normal-Hong Kong Baptist University, Zhuhai, PR China
{zhoukun, hejiakai, jensenwang}@mail.bnu.edu.cn, {wenmianyang, jiawj}@bnu.edu.cn, zhangyq987@hotmail.com

Abstract

Presentation slides are a primary medium for data-driven reporting, yet keeping complex, analytics-style decks up to date remains labor-intensive. Existing automation methods mostly follow fixed template filling and cannot support dynamic updates for diverse, user-authored slide decks. We therefore define "Dynamic Slide Update via Natural Language Instructions on User-provided Templates" and introduce DynaSlide, a large-scale benchmark with 20,036 real-world instruction–execution triples (source slide, user instruction, target slide) grounded in a shared external database and built from business reporting slides under bring-your-own-template (BYO-template) conditions. To tackle this task, we propose SlideAgent, an agent-based framework that combines multimodal slide parsing, natural language instruction grounding, and tool-augmented reasoning for tables, charts, and textual conclusions. SlideAgent updates content while preserving layout and style, providing a strong reference baseline on DynaSlide. We further design end-to-end and component-level evaluation protocols that reveal key challenges and opportunities for future research. The dataset and code are available at https://github.com/XiaoZhou2024/SlideAgent.

1 Introduction

Presentation slides are a core medium for integrating data, logical argumentation, and information dissemination in enterprise decision-making, market communication, and academic exchange (Fu et al., 2022; Zheng et al., 2025; Ge et al., 2025). In data-driven settings such as business and operations analytics, high-quality slides must not only present key figures and conclusions accurately, but also preserve professional layout, narrative coherence, and stylistic consistency (Liao et al., 2025; Yang et al., 2025). However, current slide production remains a labor-intensive, multi-stage process. From template selection, heterogeneous data collection, cleaning, and analysis, to chart design, insight writing, and final style unification, the workflow is cumbersome, requires specialized expertise, and is prone to error(Ge et al., 2025; Chia et al., 2025).

A more critical bottleneck lies in the iterative updating of report-style slide decks. In periodic business reports (e.g., weekly, monthly, or quarterly), updates usually involve local data replacement and minor adjustments to conclusions, while the overall structure and visual style remain stable (Warner et al., 2023; Chen et al., 2025). As a result, valuable human resources are consumed by low-value "copy–paste–modify" routines. This creates a clear motivation: reuse existing high-quality slide decks as personalized templates, and allow users to issue natural language instructions that drive automatic, dynamic data updates and content regeneration—while preserving the original design with high fidelity (Yang et al., 2025; Ge et al., 2025; Masry et al., 2025). Such a capability would substantially improve production efficiency and reduce human error(Long et al., 2024).

Existing work falls short of this goal. Most related approaches adopt a fixed-template filling paradigm, where systems extract information from structured sources and inject it into rigid, predefined templates (Liao et al., 2025). These methods cannot handle diverse, user-authored slide decks with complex, idiosyncratic structures, and therefore fail to support realistic bring-your-own-template (BYO-template) scenarios. This gap limits the practical deployment of current document automation techniques (Livathinos et al., 2025; Ouyang et al., 2025).

In contrast to simple template filling, dynamic content update on user-provided slides is substantially more challenging (Faysse et al., 2024; Qin et al., 2026), which requires an end-to-end intel-

[†]Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 18263–18297

ligent workflow. First, the system must perform strong multimodal document understanding, constructing a structured, element-level representation of arbitrary slide decks: identifying title hierarchies, body text, numerical values in tables, underlying data series in charts, and summary-level conclusions, together with their layout and dependencies(Verma et al., 2025; Shen et al., 2022). Second, the system must ground natural language update instructions into concrete, executable operations over this representation and the underlying data sources. Third and most challenging, it must achieve intelligent content synchronization between old and new data. This goes far beyond simple value substitution: it forms a closed-loop perception–reasoning–execution process that may involve external code execution to recompute statistics, charting engines to re-render visualizations, and language models to infer revised analytical conclusions(Masry et al., 2025; Lompo and Haraoui, 2025). This tightly integrates multimodal understanding, user-intent modeling, data retrieval, tool invocation, and advanced reasoning into a technically demanding pipeline(Su et al., 2025; Ding et al., 2024). To our knowledge, this direction has received limited attention in prior work.

To systematically address this gap, we formally define the task of Dynamic Slide Update via Natural Language Instructions on User-provided Templates. The task evaluates a system's end-to-end ability, given an arbitrary user slide deck as a dynamic template, to (i) locate and query relevant data, (ii) update tables and charts, (iii) revise textual conclusions, and (iv) generate an updated slide deck that preserves the original design with high fidelity while reflecting the new data and instructions.

To support research on this task, we introduce DynaSlide, to our knowledge, the first benchmark for dynamic slide update under BYO-template conditions. Built on real-world real-estate analytics data from Beijing, Guangzhou, and Shenzhen (2020–2024), DynaSlide organizes raw data into structured PostgreSQL tables and curates diverse slide templates that mirror authentic business reporting scenarios. We construct 20,036 instruction–execution triples of the form "source slide deck – user instruction – target slide deck," all grounded in a shared external PostgreSQL database that provides the underlying real-estate transaction records. Each triple is annotated with executable update instructions and corresponding data queries, covering a broad spectrum of business themes and statistical operations. We provide standardized train/validation/test splits and task variants to facilitate model development, ablation studies, and fine-grained evaluation.

To probe these challenges in a realistic setting and provide a strong reference point, we develop SlideAgent, an agent-based baseline system that automates slide updates via structured reasoning and tool orchestration. SlideAgent uses a large language model as a central controller that plans and coordinates specialized tools across two tightly coupled stages: (1) slide parsing and representation, where it constructs a hierarchical representation of slide elements and layout that captures semantic content, visual structure, and data dependencies; and (2) instruction-driven content synchronization, where it interprets natural language update instructions, maps them to data and document operations, calls external tools for data querying, computation, and chart rendering, and synthesizes revised textual content and conclusions. Throughout this process, SlideAgent maintains structured logs of its actions, which we use for reproducibility and fine-grained evaluation and error analysis on DynaSlide.

The main contributions of this paper are as follows:

- We formally define Dynamic Slide Update via Natural Language Instructions on User-provided Templates and release DynaSlide, a large-scale benchmark with over 20k instruction–execution triples ("source slide – instruction – target slide") grounded in a shared external database and constructed from realistic business reports.
- We develop SlideAgent, an agent-based system that orchestrates multimodal slide parsing, instruction grounding, and tool execution, providing a strong reference baseline for this task.
- We design end-to-end and module-level evaluation protocols on DynaSlide, and conduct detailed error analysis that reveals key challenges for future methods.

2 Related Work

2.1 Automatic Slide Generation vs. Dynamic Update

Research in presentation automation (Mondal et al., 2024; Bandyopadhyay et al., 2024) predominantly

targets *document-to-slide generation*, converting long-form inputs into slide decks. Recent systems such as SlideSpawn (Kumar and Chowdary, 2024), AutoPresent (Ge et al., 2025), and PPTAgent (Zheng et al., 2025) extend this paradigm by incorporating layout planning and visual verification. However, these methods rely on fixed template libraries or generative layouts, treating slide creation as a one-off task. In contrast, our Dynamic Slide Update task addresses the novel challenge of updating user-provided templates while preserving their custom layouts and maintaining the logical dependencies between data, functions, and content elements.

2.2 LLM Agents for Slide Automation

LLM-based systems advance presentation automation through hierarchical text planning and multimodal parsing (Yue et al., 2024; Yang et al., 2023; Yao et al., 2022). Recent work integrates vision-language models for semantic understanding (Pang et al., 2025; Ge et al., 2025) and modular tool invocation (Schick et al., 2023; Qin et al., 2021; Zhang et al., 2023). Yet these frameworks remain fundamentally limited: they update surface-level content (text, layout) but cannot reconstruct the underlying computational dependencies—table formulas, parameter propagation, and database queries—required for coherent multimodal updates. They lack support for dynamic, user-defined templates demanding coordinated multimodal updates. Our framework addresses this gap by integrating structured instruction parsing, multimodal layout analysis, and tool invocation to enable data-consistent updates across complex, real-world slides.

3 DynaSlide Dataset

DynaSlide is a large-scale benchmark designed to evaluate dynamic slide update on user-provided templates. Each instance is formulated as an instruction–execution task, consisting of (i) a source slide, (ii) a natural language update instruction, and (iii) a target slide generated by executing the instruction over a shared external PostgreSQL database of real-estate transactions. All slides are constructed from a family of controllable templates that factorize each slide into four element types, i.e., Title, Table, Chart, and Summary, and are grounded in executable metadata (SQL templates, statistical functions, and layout annotations) stored in YAML files. All textual content in DynaSlide (titles, captions, headers, and summaries) is in English; we design templates in the original language used in real-world reports and then translate them into English following a controlled protocol (see Appendix A.1)

3.1 Data Collection

We collected large-scale residential transaction records spanning 2020–2024 from three major cities in China: Beijing, Guangzhou, and Shenzhen, covering both new and resale housing markets. Each record contains nine fields covering time, location, and transaction attributes (e.g., transaction date, city, block, project, supply volume, sales volume, transaction price, floor area, and unit price); detailed field descriptions are provided in Appendix A.2. Following previous work (Wang et al., 2025), we source data from official and public real-estate transaction platforms, and apply three cleaning filters: (1) excluding records with missing critical attributes; (2) normalizing city and block names to ensure spatial consistency; and (3) retaining only blocks with at least 500 transactions to avoid sparsity. After cleaning, we obtain 1.79 million valid transaction records. These records are partitioned by city $\times$ market-type (new vs. resale), resulting in six relational tables. We release the processed records as a PostgreSQL database dump, which serves as the shared external data source for all slide generation and instruction execution.

3.2 Template Design

Since dynamic updates in real-world reporting primarily operate on titles, tables, charts, and summaries, we restrict DynaSlide to these four element types, which jointly cover textual, tabular, and analytical visual content. In consultation with real-estate domain experts, we determine six commonly used presentation themes and define 34 sub-templates as atomic layout units (see Appendix A.3). Each sub-template specifies both the slide layout (element roles and positions) and the table-centric computation logic used to populate the slide. The template and example of each element is illustrated in Figure 1, and the detailed descriptions of templates are shown in Appendix A.4.

3.2.1 Title template design

For each theme, we design three title templates that expose the statistical methodology and analytical perspective. During source slide generation, each

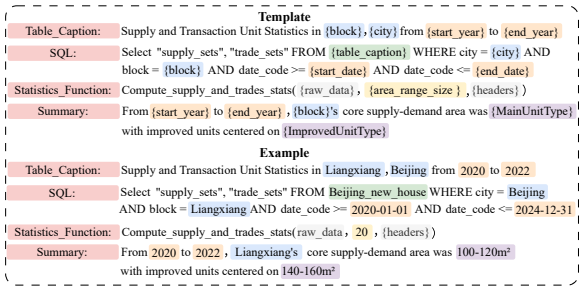


Figure 1: Illustration of the template filling.

instantiated sub-template randomly selects one title template to introduce lexical variation while preserving analytical intent.

3.2.2 Table template design

Tables consist of a caption and a table body. In practical business presentation, the table caption typically specifies the analysis topic and data scope, while the table body aggregates and statistically summarized results, e.g., shows monthly sales summaries across regions. For each theme, domain experts define a set of statistical analyses captured by 11 statistical functions F_k. Each table body is generated by pairing an SQL query template SQL_k with a statistical function F_k: SQL_k specifies which subset of records to retrieve from PostgreSQL under the slide's scope, while F_k specifies how to aggregate and summarize the retrieved data into table cells.

Caption generation. Captions describe the analytic scope (e.g., city/block and time span) using variable-based templates. As illustrated in Figure 1, each caption template combines fixed text (in black) with variable placeholders (in color). For each statistical function, we design three caption templates C_p (33 in total). We instantiate placeholders via a released "Field Mapping Dictionary" that maps template variables to concrete database fields and valid values (see Table 6 in Appendix A.4.2). Importantly, the same resolved variables are shared by the caption and the SQL template, ensuring that the caption's semantics and the retrieved data are aligned.

Table body generation and linguistic diversity. After resolving variables, we populate SQL_k to retrieve raw records, sample function-specific parameters from a released "Parameter Candidate Dictionary" (see Table 7 in Appendix A.4.2), and then apply F_k to compute the table body. To diversify surface forms without changing semantics, we decouple header generation using a released "Header Alias Dictionary" that maps canonical metrics to synonymous header variants (see Table 8 in Appendix A.4.2). Computed outputs are normalized into canonical table structures for consistent rendering and evaluation (see Figure 4 in Appendix A.4.2).

3.2.3 Chart template design

Charts are visual projections of the same computed data used in tables ("homologous data, heterogeneous presentation"). We reuse SQL_k and F_k to obtain the underlying series and render bar/line charts according to the chart type specified by the sub-template.

3.2.4 Summary template design

Summaries synthesize computed tables into textual insights. For each F_k, we select and calculate 2-4 key metrics (e.g., trends, growth rates, peak values) from the table output. Based on these metrics, we select combinations of three sets of metrics, resulting in a total of 33 summary templates $Summary_q$. Similar to table generation, summary templates are instantiated by leveraging the Field Mapping Dictionary to resolve variable placeholders, ensuring semantic consistency across titles, captions, tables, charts, and summaries within a slide.

3.2.5 Source slide Generation

We implement "EasySlide", a toolkit built on `python-pptx`, to render titles, tables, charts, and summaries while enforcing consistent layout and typography. For each source slide, we release a YAML file recording (i) `slide_filters` (database tables, SQL_k, F_k, resolved variables, and parameters) and (ii) `template_slide` (element roles and layout), providing precise grounding for downstream dynamic updates. More details for "EasySlide" and source slide generation, see Appendix A.5. This systematic process yields 7,685 distinct source slides as the foundation for our dataset. By applying multiple update instructions to each source slide, we further construct the final instruction–execution benchmark.

3.3 User Instruction and Target Slide Generation

Each instruction–execution triple shares the same PostgreSQL database (Section 3.1). We design two update scenarios: (1) Basic Replacement, which updates only core scope variables (e.g., time span or region) while keeping F_k unchanged; and (2)

Customized-Parameter, which additionally modifies constraint parameters (e.g., area segmentation or price granularity), triggering cascading updates in both SQL retrieval and statistical computation. Instructions are sampled from 24 templates, and all variable changes are stored as `query_filters` in YAML. Execution merges `slide_filters` with `query_filters` into `update_filters`, re-queries PostgreSQL, recomputes the table (via SQL_k and F_k), and synchronizes charts and summaries accordingly. Target slides are rendered to preserve the source slide's layout and style. More details are shown in Appendix A.6.

3.4 Dataset Statistics

Our DynaSlide contains 20,036 instruction–execution samples. We split the dataset into train/validation/test with a 6:2:2 ratio (12,022 / 4,007 / 4,007), and enforce zero overlap of sub-templates across splits to evaluate generalization to unseen layouts. We release all source/target slides (PPTX), the PostgreSQL database dump, YAML metadata, and the three template dictionaries, enabling reproducible end-to-end and component-level evaluation. More details are shown in Appendix A.7.

4 Method

To address the challenges posed by DynaSlide, we develop SlideAgent, an agent-based baseline system for automated slide updates. As shown in Figure 2, SlideAgent adopts a two-stage architecture. Stage 1 (Slide Understanding) parses the input slide into a structured representation that captures element locations, data sources, and functional logic. Stage 2 (Instruction-Driven Update) interprets user instructions, retrieves updated data, executes the required transformations, and regenerates consistent textual, tabular, and visual content.

4.1 Slide Parsing and Representation

Accurate automated slide updates require both precise element localization and explicit recovery of the underlying data logic. We decompose this stage into two sub-tasks: multimodal layout parsing, which identifies semantic roles and spatial structures, and logic extraction, which recovers the data sources and aggregation logic for tables and charts.

4.1.1 Multimodal Slide Layout Parsing

Accurate identification of slide elements is essential for automated updates. However, programmatic slide libraries such as `python-pptx` provide limited semantic information, making it difficult to distinguish functional roles (e.g., titles, captions, or summaries) in complex layouts.

SlideAgent addresses this limitation by combining a vision–language model (VLM) with structured PPTX parsing provided by the `python-pptx` library to jointly recover semantic and geometric information. Specifically, each slide is first rendered as a PNG image and processed by Qwen-72B-VL using structured prompts (see Appendix B.1). The VLM predicts semantic labels (e.g., title or table caption) along with approximate bounding boxes for visible elements. In parallel, `python-pptx` parses the underlying PPTX file to extract precise spatial coordinates and style metadata for all shape objects, as shown in Figure 2.

We then align VLM predictions with PPTX shapes using an Intersection over Union (IoU) matching strategy (Farhadi et al., 2018). When the IoU exceeds a threshold of 0.5, the predicted semantic label is associated with the shape's exact geometry and style attributes. The resulting structured representation records each element's type, content, bounding box, and style information (Details see Appendix B.2).

4.1.2 Table and Chart Logic Extraction

With accurately identified slide elements, in this section, we further reconstruct the underlying data logic for tables and charts, bridging the gap between the aggregated statistical data in the slides and the raw data. This requires reasoning over implicit data sources and statistical function logic. Since charts are visual representations of table data, we handle their logic extraction uniformly. This overall task presents two key challenges: (1) mapping aggregated statistics to the correct database sources and (2) recovering the aggregation logic used to generate them. We decompose this process into data source extraction and function logic extraction.

Data Source Extraction: Recovering the raw data for tables and charts requires precise database queries. However, slides typically display high-level aggregated statistics (e.g., quarterly sales summaries), whereas databases store low-level granular records (e.g., individual transactions). Consequently, directly generating SQL from slide text is error-prone, since schema details of databases, such as table names and specific fields, are often

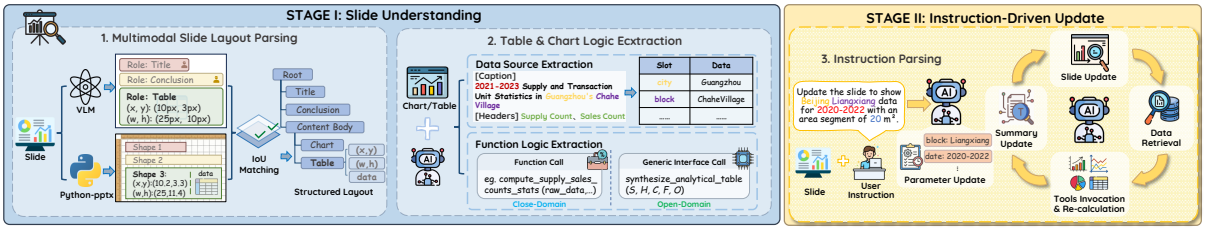


Figure 2: Overview of SlideAgent. The framework operates in two stages: (1) **Slide Understanding**, which parses layout and extracts logic; and (2) **Instruction-Driven Update**, which executes data queries and regenerates content.

implicit in the context of slides.

To address this, SlideAgent employs a schema-aligned slot-filling mechanism. We define nine standardized slots corresponding to the nine fields in the DynaSlide database (in Section 3.1). During extraction, schema cues are primarily drawn from two sources accessible via `python-pptx` in the slides: table captions, which encode analysis scope (e.g., time span or region), and table headers or chart series names, which indicate data dimensions. An LLM infers the target database table name and populates the corresponding slots based on the in-context learning (Details of prompts see Appendix B.1), producing a structured JSON object that grounds each slide element to a concrete database context.

Function Logic Extraction To recover the computational logic transforming raw data into final table presentations, we distinguish two scenarios based on the availability of the underlying function definitions: a closed-domain scenario, and an open-domain scenario.

In the closed-domain scenario, we assume that tables are generated by a finite set of statistical functions known to the system. Specifically, SlideAgent maintains exact definitions for the 11 predefined functions described in Section 3.2.2. These functions are exposed to the LLM as callable tools with explicit parameter schemas. Logic extraction is formulated as a function-calling task: given a target table, the LLM identifies the corresponding function and extracts its arguments. The recovered function and parameters are then used to directly invoke the intended computation logic. Details of tool descriptions and prompts see Appendix B.1.

In the open-domain scenario, the system has no prior knowledge of the specific function used to generate a table. This setting reflects real-world usage, where users may create arbitrary, custom analyses not covered by predefined templates. Although the specific generative function is unknown, the fundamental statistical methods (e.g., aggregation types, filtering metrics) remain analyzable.

We therefore design a generic processing interface, `synthesize_analytical_table`, which reconstructs aggregation logic from atomic components parsed by the LLM. The interface operates on *raw_data* and is parameterized by five components: *(i) Table Structure Type*, categorizing the layout into canonical table structures (see Appendix A.4.2); *(ii) Headers*, defining row and column dimensions; *(iii) Constraint Spec*, specifying binning or filtering logic with boundaries and step sizes; *(iv) Source Fields*, identifying required database columns (e.g., `dim_price`); and *(v) Operations*, defining aggregation functions (e.g., SUM, AVG, COUNT).

The LLM decomposes the unknown logic into these parameters, which are assembled into an executable computation pipeline (see Algorithm 1 in Appendix B.3 for implementation details).

4.2 Instruction-Driven Update

With the slide structure and logic extracted in Stage 1, Stage 2 performs dynamic updates following a four-step pipeline: instruction parsing, data retrieval, logic execution, and summary regeneration.

4.2.1 User Instruction Parsing

User instructions specify updated data constraints or logic modifications in natural language. We model this step as a *state update* task, as illustrated in Figure 2. The system takes the user instruction and the current parameter state (target table name, slots, and function parameters) as input, and prompts (see Appendix B.1) an LLM to update relevant fields while strictly preserving the JSON schema. The output is a new parameter state that serves as the executable specification for downstream processing.

4.2.2 SQL Generation and Data Retrieval

User instructions typically dictate specific data constraints or logic modifications. We thus model this task as a *state update* problem, where the system takes the user instruction and the *current parameter state* (a JSON object containing the target table name, slot values, and function logic parameters defined in Section 4.1.2) as input. Using structured prompts (see Appendix B.1), the LLM updates the relevant fields in the JSON object to reflect the user's instructions while preserving the schema structure. The output is a *target parameter state* containing updated slots and function arguments, which serves as the executable specification for subsequent steps.

4.2.3 Tool Invocation and Data Re-calculation

Using the updated raw data and function parameters, SlideAgent re-executes the computation logic. Closed-domain tables invoke the corresponding predefined functions, while open-domain tables invoke `synthesize_analytical_table` with reconstructed logic parameters. Both cases yield updated aggregated statistical data.

4.2.4 Summary Update and Final Rendering

Since data updates may invalidate existing summaries, we introduce a fact-aware summary generation module. Given the original summary, the pre-update data, and the post-update data, the LLM rewrites the summary to reflect the updated trends. Details of the prompts are shown in Appendix B.1.

Finally, SlideAgent repopulates the slide with updated tables, charts, and summaries using the spatial coordinates and style metadata extracted in Stage 1 (Section 4.1), ensuring strict preservation of the original layout and visual design.

5 Experiments

Dynamic slide update on user-provided templates is a newly introduced task and lacks directly comparable baselines. Accordingly, our experiments focus on: (1) comparing task success rates across different LLM backbones; (2) analyzing update accuracy across element types (Title, Table, Chart, Summary).

5.1 Implementation Details

We evaluate SlideAgent using five open-weight models as reasoning backbones: GPT-OSS (120B, 20B) and Qwen3-Instruct (80B, 30B, 14B). Qwen2.5-VL-72B is employed specifically for multimodal layout parsing. Detailed experimental configurations, including prompting strategies, software infrastructure (e.g., LangGraph, vLLM), and hardware specifications, are provided in Appendix C.1.

5.2 Evaluation Metrics

We adopt two quantitative metrics. Task Success Rate (SR) is the primary metric, defined as the percentage of test cases in which the generated slide exactly matches the ground truth in both content (text and data values) and layout structure. To further analyze modality-specific performance, we report Element-Level Accuracy, measuring the update success rates for Titles, Tables, Charts, and Summaries individually.

5.3 Main Results

Table 1 reports the performance of SlideAgent across six presentation themes under both closed-domain and open-domain settings. The results yield three key observations regarding the complexity of dynamic slide updates.

First, model scale strongly correlates with task performance. GPT-OSS-120B achieves the highest success rates, reaching 80.64% in closed-domain and 68.86% in open-domain scenarios, exceeding its 20B counterpart by over 11% and 12%, respectively. A similar trend is observed for Qwen3 models, where Qwen3-80B (75.33% / 63.91%) substantially outperforms Qwen3-14B (45.48% / 31.13%). This large performance gap reflects the intrinsic difficulty of dynamic slide updates, which require a long chain of perception, querying, and computation. Errors in any intermediate step can invalidate the final output, and smaller models often fail to maintain strict instruction adherence over extended reasoning trajectories. In contrast, larger models demonstrate greater robustness in handling long-context, multi-step workflows.

Second, open-domain scenarios consistently degrade performance, disproportionately affecting smaller models. While GPT-OSS-120B shows a moderate 14.6% relative decline (80.64%→68.86%), Qwen3-14B exhibits a much larger 31.5% relative decline (45.48%→31.13%). This pattern aligns with the differing requirements of the two logic extraction modes: closed-domain update mainly involves selecting from a predefined function library, whereas open-domain update requires reconstructing computation logic from scratch, where larger models show stronger

Model	Theme 1		Theme 2		Theme 3		Theme 4		Theme 5		Theme 6		Average	
	Cls	Opn	Cls	Opn	Cls	Opn	Cls	Opn	Cls	Opn	Cls	Opn	Cls	Opn
GPT-OSS-120B	90.12	78.29	71.83	70.91	78.48	62.56	77.03	58.17	81.01	72.31	85.36	70.89	80.64	68.86
GPT-OSS-20B	81.48	68.45	61.55	57.14	65.95	49.62	61.69	44.17	67.09	60.40	77.45	57.72	69.20	56.25
Qwen3-80B	87.65	75.69	70.14	65.68	73.42	60.27	67.53	53.67	69.62	62.84	83.64	65.32	75.33	63.91
Qwen3-30B	85.06	71.06	63.38	61.50	68.35	54.95	63.64	48.00	68.35	61.26	79.64	61.39	71.40	59.69
Qwen3-14B	50.74	29.09	50.70	29.09	34.18	25.11	38.96	31.67	45.57	35.01	52.73	37.22	45.48	31.13

Table 1: Task Success Rates (%) of SlideAgent across six presentation themes using different LLM backbones. Columns Cls and Opn denote Closed-Domain and Open-Domain update scenarios, respectively.

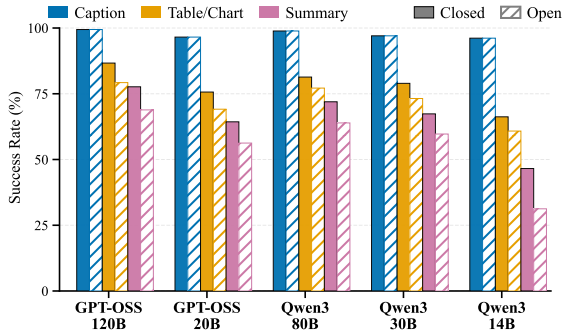


Figure 3: Element-level accuracy across closed-domain (Cls) and open-domain (Opn) scenarios.

generalization beyond template memorization.

Third, task difficulty varies significantly across presentation themes. Taking GPT-OSS-120B as an example, Theme 1 achieves 90.12% in closed-domain settings, whereas Themes 2, 3, and 4 pose substantially greater challenges, reaching only 71.83%, 78.48%, and 77.03%, respectively. This variation reflects the differing demands placed on the perception–querying–computation pipeline. Theme 1 primarily involves simple table structures and basic statistical operations, facilitating reliable layout parsing and field mapping. In contrast, Themes 2- 4 require complex cross-dimensional aggregations and multi-layer constraints, which amplify the likelihood of errors during logic extraction and SQL generation. These results indicate that, even under identical database schemas and model architectures, intrinsic task complexity remains a critical bottleneck for automation.

Figure 3 further breaks down performance by element type. Caption updates consistently achieve high accuracy across all models, as they largely involve direct substitution of constraint parameters (e.g., dates or locations) with minimal reasoning. In contrast, Table and Chart updates are substantially more challenging, with accuracy ranging from 66.23% to 86.67% in closed-domain settings and dropping to 60.82%–79.24% in open-domain settings. This degradation empirically supports the analysis in Section 4.1.2: open-domain scenarios require models to reverse-engineer aggregation logic from visual and textual cues rather than selecting predefined functions, increasing error accumulation during SQL generation. Summary updates exhibit the highest failure rates, particularly for smaller models (e.g., Qwen3-14B drops to 37.61% in open-domain settings), confirming that summary generation is a fact-aware reasoning task that requires strict consistency between recomputed values and the textual narrative—a capability that scales strongly with model size.

To provide deeper insights into the performance bottlenecks of dynamic slide updates, we conduct module-wise evaluations that isolate the accuracy of individual pipeline components. This analysis enables fine-grained understanding of where current systems succeed and where they encounter fundamental challenges.

The Multimodal Slide Layout Parsing module achieves 99.5% accuracy in our evaluation. Since this module consistently uses Qwen2.5-VL-72B for layout parsing across all settings, the layout parsing results remain identical for all model variants. Given that each PowerPoint slide typically contains only 4–6 elements, the layout structure is relatively simple, which also makes the parsing task more stable.

The accuracy of six core modules in open-domain scenarios is reported in Table 2. In this setting, the system is required to reconstruct computation logic without access to predefined function libraries. The evaluation assesses the fidelity of intermediate outputs at each stage of the SlideAgent pipeline, measuring how accurately each component executes its specific sub-task.

Several critical patterns emerge from these results. Function Logic Extraction and Data Source Extraction consistently achieve the highest accu-

Model	Func. Logic	Data Src.	Instr. Parse	SQL Gen.	Tool Inv.	Sum. Upd.	Task SR†
GPT-OSS-120B	88.34	90.37	85.69	85.56	79.24	68.44	68.86
GPT-OSS-20B	74.63	77.96	76.44	76.21	69.13	56.30	56.25
Qwen3-80B	87.19	86.14	82.63	82.43	77.16	64.72	63.91
Qwen3-30B	81.55	82.40	78.59	79.54	73.20	60.51	59.69
Qwen3-14B	63.38	70.17	70.35	73.09	60.82	31.25	31.13

Table 2: Module-wise accuracy (%) in open-domain scenarios. **Task SR**† denotes the end-to-end Task Success Rate from Table 1 for comparison. Abbreviations: **Func. Logic**: Function Logic Extraction, **Data Src.**: Data Source Extraction, **Instr. Parse**: User Instruction Parsing, **SQL Gen.**: SQL Generation, **Tool Inv.**: Tool Invocation, **Sum. Upd.**: Summary Update.

racy across all models, with GPT-OSS-120B reaching 88.34% and 90.37% respectively. This indicates that larger models possess strong capabilities in reverse-engineering statistical operations and mapping slide content to database schemas.

In contrast, Summary Update presents the most significant challenge, with accuracy dropping substantially across all model scales. GPT-OSS-120B achieves only 68.44% on this module, while smaller models like Qwen3-14B fall to 31.25%. This confirms our hypothesis that summary generation requires sophisticated fact-aware reasoning to ensure textual narratives faithfully reflect updated data trends—a capability that scales strongly with model capacity.

The performance gap between Function Logic Extraction and Summary Update (19.90 percentage points for GPT-OSS-120B) reveals a key insight: while models can effectively extract and execute computational logic, translating quantitative updates into coherent natural language conclusions remains a fundamental bottleneck. This suggests that future work should prioritize developing specialized modules for cross-modal consistency checking and data-grounded text generation.

Model scale demonstrates consistent impact across all modules. Comparing GPT-OSS-120B to GPT-OSS-20B, we observe improvements of 13.71 points in Function Logic Extraction, 12.41 points in Data Source Extraction, and 12.14 points in Summary Update. These gains highlight that high-order reasoning tasks in dynamic slide updates benefit substantially from increased model capacity.

To further assess generalization beyond the main benchmark setting, we conduct two supplementary experiments and summarize a representative case study before concluding. First, on 500 randomly sampled template-generated cases, stronger models continue to outperform weaker ones across both closed and open settings, with closed-source models remaining consistently ahead of open-weight alternatives. The relative ranking among open-weight models is also preserved, indicating that the end-to-end slide updating task is strongly capability-dependent. Second, we compare the same sampled cases with human-authored slides and observe broadly consistent trends, suggesting that DynaSlide evaluates generalizable slide-updating ability rather than template-specific matching. We additionally analyze representative failure modes of SlideAgent through a case study. The results show that failures mainly stem from error accumulation across the perception–reasoning–execution pipeline and from the difficulty of maintaining numerically precise summary rewriting in the final stage. Together, these observations further highlight that the central challenge lies not only in executing each module correctly, but also in preserving cross-module consistency through to the final narrative. Detailed results and discussion are provided in Appendix C.2, and Appendix C.3.

6 Conclusion

In this paper, we formalize the task of "Dynamic Slide Update via Natural Language Instructions on User-provided Templates", addressing realistic bring-your-own-template scenarios that are not supported by existing template-based methods. To facilitate systematic research, we introduce DynaSlide, a large-scale benchmark with 20,036 real-world instruction–execution triples grounded in a shared external database. We further propose SlideAgent, an agent-based framework that combines multimodal slide parsing, instruction grounding, and tool-augmented reasoning to update tables, charts, and textual conclusions while preserving layout and style. Extensive evaluations reveal key challenges in logic reconstruction, cross-modal consistency, and fact-aware summary generation.

Limitations

This study has several limitations that do not undermine its contributions. First, our benchmark focuses exclusively on the real estate domain (e.g., housing transaction data, price trends, supply-demand analysis). This is not a limitation of capability, but a deliberate design to ensure data complexity: real estate data exhibits rich interdependencies—such as how price changes affect sales volume, or how regional supply impacts unit pricing—that are ideal for stress-testing an agent's numerical reasoning and multi-variable coordination. The core mechanism—interpreting natural language instructions, querying a database, updating tables/charts, and regenerating summaries—is domain-agnostic and directly applicable to other data-intensive sectors (e.g., financial earnings reports, supply chain dashboards, clinical trial summaries) where similar logic consistency is required. We chose depth over breadth: mastering one complex domain provides stronger evidence of reasoning capability than superficial coverage of many simple domains.

Second, DynaSlide employs a family of controllable templates rather than uncurated slides scraped from the web. This decision sacrifices some stylistic diversity to ensure executable ground truth. In a benchmark setting, verifiable correctness is essential; using arbitrary wild files would preclude the automated, deterministic evaluation of table values and chart series. Our templates thus serve as a standardized testbed where layout fidelity and content accuracy can be rigorously measured, avoiding the evaluation noise inherent in open-ended generation tasks. Future work may explore adapting our logic-driven update engine to noisier, unstructured layout parsers once the core reasoning framework is established.

Third, our framework operates on the premise that slide elements are grounded to a structured database (PostgreSQL). This alignment reflects standard enterprise workflows, where periodic reports (e.g., monthly reviews) are typically derived from pre-established data warehouses. Consequently, our system addresses the update phase of the reporting lifecycle. The "cold start" problem—reconstructing structured databases from legacy static presentations—constitutes a distinct document understanding challenge requiring chart de-rendering and schema inference, which are orthogonal to our focus on logically consistent updates. Our approach targets the high-frequency scenario of maintaining reports with accessible data lineages, a scope covering a significant portion of automated reporting needs.

Finally, our work prioritizes the analytical core of presentations: tables, charts, and data-driven summaries. We do not currently handle decorative graphics or conceptual diagrams. This scope is motivated by the observation that in periodic reporting, analytical elements account for the vast majority (>90%) of content modifications. By focusing on these high-frequency update targets, we achieve broad multimodal coverage—spanning numerical reasoning, visual rendering, and natural language generation—while maintaining engineering tractability. Supporting non-analytical visual content would require additional perception modules (e.g., for semantic style transfer) that are complementary to, but distinct from, the logical update problem addressed here.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62272050 and the grant of Beijing Normal- Hong Kong Baptist University sponsored by Guangdong Provincial Department of Education; in part by Zhuhai Science-Tech Innovation Bureau under Grant No. 2320004002772 and the Interdisciplinary Intelligence Super Computer Center of Beijing Normal University (Zhuhai).

References

Sambaran Bandyopadhyay, Himanshu Maheshwari, Anandhavelu Natarajan, and Apoorv Saxena. 2024. Enhancing presentation slide generation by llms with a multi-staged end-to-end approach. *arXiv preprint arXiv:2406.06556*.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Yew Ken Chia, Liying Cheng, Hou Pong Chan, Maojia Song, Chaoqun Liu, Mahani Aljunied, Soujanya Poria, and Lidong Bing. 2025. M-longdoc: A benchmark for multimodal super-long document understanding and a retrieval-aware tuning framework. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9244–9261.

Xuanwen Ding, Jie Zhou, Liang Dou, Qin Chen, Yuanbin Wu, Arlene Chen, and Liang He. 2024. Boosting large language models with continual learning for aspect-based sentiment analysis. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4367–4377.

Ali Farhadi, Joseph Redmon, and 1 others. 2018. Yolov3: An incremental improvement. In *In Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1–6.

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. 2024. Colpali: Efficient document retrieval with vision language models. *arXiv preprint arXiv:2407.01449*.

Tsu-Jui Fu, William Yang Wang, Daniel McDuff, and Yale Song. 2022. Doc2ppt: Automatic presentation slides generation from scientific documents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 634–642.

Jiaxin Ge, Zora Zhiruo Wang, Xuhui Zhou, Yi-Hao Peng, Sanjay Subramanian, Qinyue Tan, Maarten Sap, Alane Suhr, Daniel Fried, Graham Neubig, and 1 others. 2025. Autopresent: Designing structured visuals from scratch. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 2902–2911.

Keshav Kumar and Ravindranath Chowdary. 2024. Slidespawn: An automatic slides generation system for research publications. *arXiv preprint arXiv:2411.17719*.

Wenhui Liao, Jiapeng Wang, Hongliang Li, Chengyu Wang, Jun Huang, and Lianwen Jin. 2025. Doclayllm: An efficient multi-modal extension of large language models for text-rich document understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4038–4049.

Nikolaos Livathinos, Christoph Auer, Maksym Lysak, Ahmed Nassar, Michele Dolfi, Panos Vagenas, Cesar Berrospi Ramis, Matteo Omenetti, Kasper Dinkla, Yusik Kim, and 1 others. 2025. Docling: An efficient open-source toolkit for ai-driven document conversion. *arXiv preprint arXiv:2501.17887*.

Boammani Aser Lompo and Marc Haraoui. 2025. Visual-tableqa: Open-domain benchmark for reasoning over table images. *arXiv preprint arXiv:2509.07966*.

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On llms-driven synthetic data generation, curation, and evaluation: A survey. *arXiv preprint arXiv:2406.15126*.

Ahmed Masry, Megh Thakkar, Aayush Bajaj, Aaryaman Kartha, Enamul Hoque, and Shafiq Joty. 2025. Chartgemma: Visual instruction-tuning for chart reasoning in the wild. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 625–643.

Ishani Mondal, S Shwetha, Anandhavelu Natarajan, Aparna Garimella, Sambaran Bandyopadhyay, and Jordan Boyd-Graber. 2024. Presentations by the humans and for the humans: Harnessing llms for generating persona-aware slides from documents. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2664–2684.

Linke Ouyang, Yuan Qu, Hongbin Zhou, Jiawei Zhu, Rui Zhang, Qunshu Lin, Bin Wang, Zhiyuan Zhao, Man Jiang, Xiaomeng Zhao, and 1 others. 2025. Omnidocbench: Benchmarking diverse pdf document parsing with comprehensive annotations. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24838–24848.

Wei Pang, Kevin Qinghong Lin, Xiangru Jian, Xi He, and Philip Torr. 2025. Paper2poster: Towards multimodal poster automation from scientific papers. *arXiv preprint arXiv:2505.21497*.

Libo Qin, Qiguang Chen, Xiachong Feng, Yang Wu, Yongheng Zhang, Yinghui Li, Min Li, Wanxiang Che, and Philip S Yu. 2026. Large language models meet nlp: A survey. *Frontiers of Computer Science*, 20(11):2011361.

Libo Qin, Zhouyang Li, Wanxiang Che, Minheng Ni, and Ting Liu. 2021. Co-gat: A co-interactive graph attention network for joint dialog act recognition and sentiment classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 13709–13717.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Zejiang Shen, Kyle Lo, Lucy Lu Wang, Bailey Kuehl, Daniel S Weld, and Doug Downey. 2022. Vila: Improving structured content extraction from scientific pdfs using visual layout groups. *Transactions of the Association for Computational Linguistics*, 10:376–392.

Jinwei Su, Yinghui Xia, Qizhen Lan, Xinyuan Song, Yang Jingsong, Lewei He, and Tianyu Shi. 2025. Difficulty-aware agent orchestration in llm-powered workflows. *arXiv e-prints*, pages arXiv–2509.

Abhigya Verma, Sriram Puttagunta, Seganrasan Subramanian, and Sravan Ramachandran. 2025. Graft: Graph and table reasoning for textual alignment–a benchmark for structured instruction following and visual reasoning. *arXiv preprint arXiv:2508.15690*.

Zhensheng Wang, Wenmian Yang, Kun Zhou, Yiquan Zhang, and Weijia Jia. 2025. Retqa: A large-scale open-domain tabular question answering dataset for real estate sector. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25452–25460.

Jeremy Warner, Amy Pavel, Tonya Nguyen, Maneesh Agrawala, and Bjoern Hartmann. 2023. Slidespecs: Automatic and interactive presentation feedback collation. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 695–709.

Yuheng Yang, Wenjia Jiang, Yang Wang, Yiwei Wang, and Chi Zhang. 2025. Auto-slides: An interactive multi-agent system for creating and customizing research presentations. *arXiv preprint arXiv:2509.11062*.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Murong Yue, Wenlin Yao, Haitao Mi, Dian Yu, Ziyu Yao, and Dong Yu. 2024. Dots: Learning to reason dynamically in llms via optimal reasoning trajectories search. *arXiv preprint arXiv:2410.03864*.

Zhuosheng Zhang, Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, Zuchao Li, and Hai Zhao. 2023. Universal multimodal representation for language understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7):9169–9185.

Hao Zheng, Xinyan Guan, Hao Kong, Wenkai Zhang, Jia Zheng, Weixiang Zhou, Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. 2025. Pptagent: Generating and evaluating presentations beyond text-to-slides. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14413–14429.

A Dataset Construction

A.1 Translation Protocol

Since the original slide templates and data schema were derived from the Chinese real estate market, all textual elements were initially in Chinese. We employed a three-step translation protocol to convert them into English:

Template Content Translation. We translated all static text in titles, captions, and summaries from Chinese to English, while preserving dynamic placeholder variables.

Database Schema and Dictionary Adaptation. We translated database schema names (table and column names) into English. For the Header Alias Dictionary, we curated diverse English synonyms for each metric (e.g., 'Avg Price', 'Unit Cost', 'Average Value' for the same price field), enabling linguistic variety in the final slides. The Field Mapping Dictionary was updated to link English template variables to database fields.

Terminology Verification. Domain experts reviewed all translated assets to ensure technical terms matched professional English standards in real estate.

A.2 Data Field Descriptions

The detailed schema of the real-estate database used in this study is presented in Table 3. This dataset compiles residential transaction records from major Chinese cities across both new and resale housing markets. Each entry contains nine key attributes that jointly encode spatial location (e.g., city, block), temporal information, and core transaction metrics (e.g., volume, floor area, price). These fields constitute the essential data backbone that underpins the statistical analyses and automated reporting workflows presented in the main text.

A.3 Template Specification

To ensure reproducibility and facilitate fine-grained evaluation, we provide the full specification of our template system in Table 12. This table enumerates the 34 atomic sub-templates across all identified business themes. Each entry explicitly defines the four core elements required for slide generation: (1) a set of **Title Templates** that introduce the analytical perspective; (2) a specific **Statistical Function** (F_k) that drives data retrieval and aggregation; (3) **Caption Templates** that define the scope of tables/charts; and (4) **Summary Templates** that synthesize quantitative insights into natural language. All templates utilize variable placeholders (e.g., `City`, `Start_Year`) denoted in `monospace`. These variables are automatically populated during generation using the real-estate market variable definitions provided in Table 9, ensuring semantic consistency across all slide elements.

A.4 Template Design

In professional real-estate presentations, slides typically consist of several key elements: titles, body text, tables, charts, images, and summary statements. For the slide auto-update task, we focus specifically on four core element types—Title, Table, Chart, and Summary—for two reasons: (1) they are the most frequently updated in real-world reporting workflows; (2) together they cover all

Table 3: Description of fields in the real-estate database.

Field Name	Data Type	Definition	Example
city	String	Name of the target city.	Beijing, Shanghai
block	String	Name of the administrative district or sub-region.	Chaoyang District
project	String	Name of the residential project or community.	Oceanwide Elite Estate
date_code	String	Temporal identifier.	2024-05-01
supply_sets	Integer	Volume of new housing units listed for sale.	1
trade_sets	Integer	Volume of housing units sold/transacted.	1
dim_area	Float	Floor area (m^2).	125.5
dim_price	Float	Total price (15.2k CNY).	15.2
dim_unit_price	Float	Price per square meter (CNY/m^2).	68000.0

principal content modalities (text, tabular data, visual analytics). Building on thorough domain analysis, we identify six recurring business analysis themes (see Appendix A.3), such as *Cross-Structure Analysis of Resale-House Transactions* and *Analysis of Neighborhood Price Trends*. For each theme M_i, we define a prototype slide composed of 2–7 configurable sub-templates T_j, yielding a total of 34 distinct sub-templates tailored to common real-estate analytic scenarios. These 34 sub-templates serve as the atomic layout units that we later use to build source slides and to define train/validation/test splits. The design logic for Title, Table, Chart, and Summary templates is detailed below.

A.4.1 Title template design

Titles in presentations serve as both guides and indices, directly exposing the statistical methodology and analytical perspective employed on each slide. For example, the title "Cross-Analysis of New Housing Transaction Structure" signals that the slide presents multi-dimensional cross-tabulated statistics. In designing the *Title*$_i$ template suite, we created three title templates for each analytical theme (18 in total; see Appendix A.3). During source slide generation, for each sub-template T_j within theme M_i, the system randomly assigns one of the three corresponding title templates. This random assignment introduces lexical and stylistic variation while preserving the underlying analytical intent of each sub-template.

A.4.2 Table template design

Tables consist of a caption and a data body. In practical business analytics, the table caption typically specifies the analysis topic and data scope (e.g., location and time period in real-estate contexts), while the table body presents not raw data directly, but aggregated and statistically summarized results. For example, a real-estate company typically shows monthly or quarterly sales summaries across regions, rather than transaction records for each individual building. Therefore, for each theme's particular statistical analysis requirements, we collaborate with real-estate domain experts to define the core statistical logic. For each theme M_i, we design 1–3 specific analytical dimensions and define 11 distinct statistical functions F_k. Each statistical function F_k corresponds one-to-one to a specific table template. Intuitively, SQL_k (defined below) specifies *what subset of records* to retrieve, while F_k encodes *how to aggregate and summarize* the retrieved data. During source slide generation, each sub-template T_j within theme M_i randomly samples a statistical function F_k from the theme's function set, applies it to generate results, and renders the output using the corresponding table body template.

Caption generation. We construct "caption + table body" generation templates based on these statistical functions. The caption explicitly specifies the analytic scope and context of the table (e.g., temporal span or geographic region). We adopt variable-based templates and, for each statistical function, we design three caption templates C_p (33 in total). As illustrated in Figure 1, each caption template combines fixed text (in black) with variable placeholders (in color). To instantiate these templates, we construct a **Field Mapping Dictionary** (released with the dataset; see Table 6) that links template-level variables to concrete database fields. During the instantiation process, the system queries this dictionary to resolve all placeholders. By retrieving the corresponding concrete values (e.g., "Beijing") from the database, it transforms the abstract template into a final caption with ex-

plicit business semantics.

Table body generation. The table body is primarily responsible for presenting core data derived from aggregation and statistical analysis. Structurally, the table body template is composed of an SQL query template SQL_k paired with a function template F_k, corresponding to the two primary generation stages: data retrieval and statistical computation. First, to ensure the table content aligns strictly with the caption's scope, we leverage shared variables: we populate the SQL query template SQL_k using the same variables resolved via the Field Mapping Dictionary during caption instantiation, thereby retrieving precise raw data. Subsequently, the system configures the computation logic by querying the **Parameter Candidate Dictionary** (released with the dataset; see Table 7) to randomly sample function-specific parameters (e.g., step sizes for binning) for F_k. In parallel, we specifically decouple header generation to introduce linguistic diversity. The system accesses the **Header Alias Dictionary** (released with the dataset; see Table 8)—which maps standard metrics to multiple synonymous variants—and randomly selects distinct aliases for both column and row headers. This mechanism ensures that tables presenting the same underlying data logic can exhibit varied terminological expressions. Finally, the system passes these configuration inputs—structural parameters and selected headers—alongside the retrieved raw data into F_k to generate the final table body.

Canonical table structures. For table presentation formats, we organize all computed results into three canonical table structures, each corresponding to distinct analytic perspectives:

- **Field-constraint type:** rows correspond to statistical fields/metrics (e.g., average price, supply, sales), while columns correspond to a single constraint dimension (e.g., year or area bin), supporting multi-metric comparison under aligned conditions.
- **Constraint-field type:** rows correspond to constraint values, while columns correspond to statistical fields/metrics, suitable when the number of metrics is small and the focus is comparison across constraint values.
- **Cross-constraint type:** both rows and columns encode different constraints, with cell values reflecting their intersections; this structure most strongly tests system capability for understanding compound constraints.

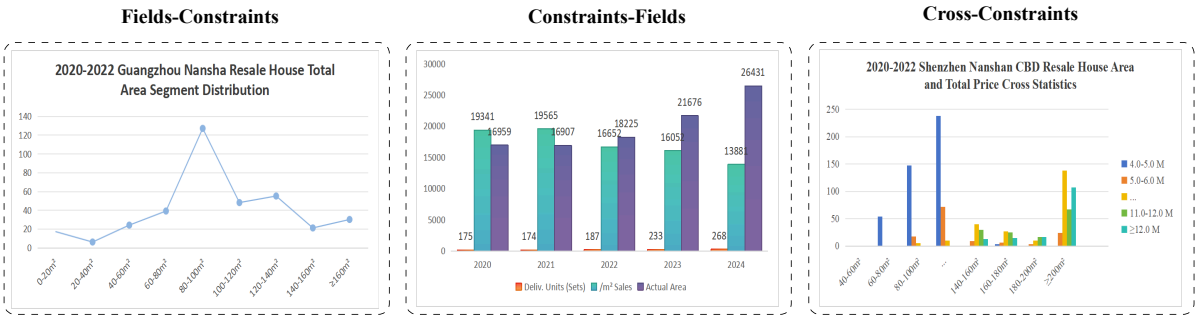
These three canonical structures(see Figure 4) allow us to systematically probe models' ability to interpret tables with simple constraints (Field-constraint and Constraint-field) and more complex, compositional constraints (Cross-constraint).

A.4.3 Chart template design

Charts are treated not as independent content units, but as visual projections of the same structured data used in tables. This approach allows for diverse presentation formats of the same underlying dataset. Consequently, the design process for chart templates mirrors that of table templates (Section 3.2.2), strictly adhering to the principle of "homologous data, heterogeneous presentation." Specifically, we reuse the statistical functions F_k and SQL retrieval logic originally defined for tables to obtain the underlying data. We then leverage chart rendering tools to render this data according to the specified chart type. In the current release, we support bar and line charts, which cover the predominant chart types observed in our real-world reference reports.

A.4.4 Summary template design

The Summary synthesizes complex data into textual insights. Analyzing real-world reports reveals that summaries typically use stable phrasing structures with dynamically injected data conclusions. To automate this, we implemented dedicated insight-extraction logic for each of the 11 statistical functions. For each function, we automatically compute a set of key metrics (e.g., trends, growth rates, peak values)—typically 2–4 core metrics per function—from the computed table data. We then designed 3 distinct summary templates $Summary_q$ for each function (33 in total) to ensure expressive diversity. Similar to caption and table body generation, summary templates are instantiated by leveraging the Field Mapping Dictionary to resolve variable placeholders (e.g., resolving city to "Beijing"), ensuring consistent data context across all three slide components. Across all business themes, these templates collectively cover approximately 30–40 distinct business indicator types (e.g., month-on-month growth in average price, distribution of unit prices across area bins; see Table 9). Crucially, this unified approach guarantees semantic consistency across titles, tables, charts, and summaries,



	Sales Count
0-20m²	17
20-40m²	6
40-60m²	24
60-80m²	39
80-100m²	127
100-120m²	48
120-140m²	55
140-160m²	21
≥160m²	30

(d)

	2020	2021	2022	2023	2024
Sales Count	175	174	187	233	268
/m² Sales	19,341	19,565	16,652	16,052	13,881
Total Area	16,959	16,907	18,225	21,676	26,431

(e)

	2.0-3.0 M	3.0-4.0 M	...	11.0-12.0 M	≥12.0 M
40-60m²	1	0	0	0	0
60-80m²	54	1	0	0	0
80-100m²	147	17	5	0	0
...	238	72	10	1	0
140-160m²	0	9	40	30	13
160-180m²	3	6	27	25	15
180-200m²	0	3	10	16	16
≥200m²	0	24	138	167	107

(f)

Figure 4: Illustration of three canonical table structures: Field-constraint, Constraint-field, and Cross-constraint.

which is essential for the dynamic update task we target.

A.5 Source slide generation

To support systematic template instantiation and slide generation, we develop **EasySlide**, a toolkit built on `python-pptx` (see Table 10). EasySlide provides simplified interfaces for common slide operations such as adding titles, inserting tables, and rendering charts, improving code maintainability and enabling efficient batch generation. The source slide generation process proceeds through four key steps:

- *Template instantiation:* The system selects a sub-template based on the business theme and populates placeholders (e.g., city) with concrete parameters to generate specific titles and captions.
- *Data generation:* Using these parameters, the system executes SQL queries to retrieve raw data, which is then processed by statistical functions to compute table values and extract key insights for summaries.
- *Content rendering:* The EasySlide toolkit renders the generated titles, tables, charts, and textual summaries onto slides based on the computed data, ensuring consistency in layout and style.
- *Metadata annotation:* For each slide, a YAML file is generated recording two essential components: $slide_filters$ captures the data pipeline (database tables, SQL templates, statistical functions, filter parameters), while $template_slide$ documents the slide layout (element types, positions, and content roles). Together, these structured annotations provide precise grounding for subsequent dynamic content updates.

We release these YAML files alongside the slide decks, so that future work can directly leverage the structured annotations for training, evaluation, and tool-based execution. This systematic process yields 7,685 distinct source slides as the foundation for our dataset.

A.6 User Instruction and Target Slide Generation

Based on the source slides, we further generate user instructions and target slides to construct complete instruction–execution triples. Each triple consists of a *source slide*, a *user instruction*, and a *target slide*. All triples share the same external PostgreSQL database described in Section 3.1, which serves as the underlying data source for both slide generation and instruction execution. We design two typical update scenarios based on the metadata of source slides (such as data filtering conditions

and statistical function configurations):

1. **Basic Replacement Instructions:** Update only core variables (e.g., temporal spans, geographic regions) while maintaining the statistical functions and analytical dimensions. For example: Generate an analysis of {city} {block} from {start year} to {end year}."

2. **Customized-Parameter Instructions:** Further modify constraint parameters (e.g., area segmentation, price granularity), triggering cascading updates across data queries and statistical computations. For example: Change the area segmentation to area range and price granularity to price range."

To support the diversity of these two scenarios, we construct approximately 24 instruction templates. To flexibly accommodate different parameter combinations, we introduce an instruction variable dictionary, stored as `query_filters` in the YAML metadata, that records the variables to be modified in the instruction and their new values. After instantiation, we employ Qwen3-80B to paraphrase the generated instructions, ensuring linguistic diversity and naturalness. To ensure one-to-one correspondence among source slides, user instructions, target slides, and metadata, we design the following three-step process:

Instruction sampling and filling. For each source slide, the system randomly selects an instruction template. Based on the parameter dictionary, the system randomly samples new parameter values (such as new geographic regions or statistical configurations), fills them into placeholders to generate a natural language instruction, and records the same new values in `query_filters` to keep the instruction text and executable parameters synchronized.

Data querying and recomputation. The system merges `slide_filters` (the original data query configuration from the source slide's YAML file) with `query_filters` (parameter changes from the instruction) to generate `update_filters`. According to `update_filters`, it then executes SQL queries to retrieve updated data and recomputes tables, charts, and textual insights in a single, consistent update pipeline.

Rendering and metadata recording. Using the EasySlide toolkit, the system renders the updated tables, charts, and textual summaries onto new slides. During this process, the system strictly preserves the layout, color scheme, and typography of the source slide to ensure continuity of user experience. Simultaneously, the system generates a corresponding YAML metadata file for the target slide, containing `query_filters` (instruction parameters), `update_filters` (merged filtering conditions), and `output_slide` (target slide information), forming a complete, traceable record. Together, the source slide, user instruction, target slide, shared PostgreSQL database, and associated YAML metadata form a fully specified instruction–execution instance.

A.7 Dataset Statistics

DynaSlide contains 20,036 instruction–execution samples, each represented as a triple ((*source slide*, *user instruction*, *target slide*)). Crucially, all samples are grounded in the shared structured database of real-estate transaction records described in Section 3.1, which provides the external data context for both slide generation and instruction execution. We split the dataset into train/validation/test sets with a 6:2:2 ratio (12,022 / 4,007 / 4,007) and enforce zero overlap of sub-templates (T_j) across splits to evaluate generalization to unseen layouts. The dataset covers six analytical themes, 11 statistical functions, and two instruction scenarios (basic variable replacement vs. parameter customization). Each sample is accompanied by structured YAML metadata and executable data dependencies (e.g., SQL templates and function parameters), enabling both end-to-end and component-level evaluation. Table 11 provides detailed statistics. We release (i) all source and target slide decks in PPTX format, (ii) a PostgreSQL database dump of the underlying transaction records, and (iii) YAML metadata files and template dictionaries (Field Mapping, Parameter Candidates, and Header Aliases), together with the EasySlide toolkit configuration. This complete release supports reproducible research on dynamic slide update and facilitates future model development, ablation studies, and fine-grained evaluation.

B Method Implementation Details

B.1 Prompts Used in SlideAgent

This section summarizes all prompts employed within the SlideAgent framework.

In the **Slide Parsing and Representation** stage, the Multimodal Layout Parsing module utilizes a

structured prompt (Figure 5) to guide the VLM in predicting semantic labels and bounding boxes. For **Logic Extraction**, we use distinct prompts based on the sub-task: the Data Source Extraction module employs a schema-aligned slot-filling prompt (Figure 6) to ground slide elements to database fields. The Function Logic Extraction module uses two separate prompts: a function-calling prompt for the closed-domain scenario (Figure 7) to identify predefined statistical functions, and a generic decomposition prompt for the open-domain scenario (Figure 8) to reconstruct aggregation logic from atomic components.

In the **Instruction-Driven Update** stage, the User Instruction Parsing module uses a state-update prompt (Figure 9) to modify the parameter state based on user input while preserving the JSON schema. Subsequently, the SQL Generation module employs a structured prompt (Figure 10) to convert these updated parameters into executable database queries. Finally, the Summary Update module utilizes a fact-aware rewriting prompt (Figure 11), which takes the original summary and pre/post-update data as input to generate factually consistent textual insights.

B.2 Details of Structured Slide Representation

We define the structured representation of a slide as a list of element objects, blending VLM semantic predictions with precise attributes from the PPTX file. Specifically, each element object contains the following fields: Each element is represented by five fields: **id**, a unique identifier within the slide; **type**, the underlying PPTX shape type (e.g., `textBox`); **role**, the semantic label predicted by the VLM and confirmed via IoU matching (e.g., `caption`, `title`); **text**, the textual content; and **layout**, the geometric attributes from PPTX metadata, including `x`, `y`, `width`, and `height`.

B.3 Algorithm for Open-Domain Logic Reconstruction

The process of the synthesize analytical table interface is outlined in Algorithm 1. Instead of hard-coding logic for diverse table types, we formulate table generation as a unified *Split–Apply–Combine* workflow. The system first slices raw data based on constraints, groups records by row and column headers, applies the target aggregation function, and finally reshapes the output into the target layout.

Symbol	Definition
D	Raw data set retrieved from the database, consisting of N records $\{r_1, \ldots, r_N\}$.
$\mathcal{L}$	The logic specification tuple: $\mathcal{L} = (S, H, C, F, O)$.
S	**Structure Type**. Determines the matrix layout strategy. • `FC` (Field-Constraint): Metrics as rows, constraints (e.g., time) as columns. • `CF` (Constraint-Field): Constraints as rows, metrics as columns. • `XC` (Cross-Constraint): Intersection of two constraints (e.g., City $\times$ Year).
H	**Headers**. $H = (H_{row}, H_{col})$. Defines the grouping keys for rows and columns.
C	**Constraint Spec**. A 5-element list specifying slicing logic and formatting: $C = [k, T, v_{start}, v_{end}, v_{step}]$. • k: **Constraint Type** (e.g., `"year"` or `"price"`). • T: **Parameterized Template** for dynamic formatting (e.g., `"-M"` or `"-"`). • v_{start}, v_{end}: **Boundary Values** defining the closed interval (e.g., `"2020"`, `"2023"`). • v_{step}: **Parameter Interval** or step size (e.g., `"1"` or `"price_range_size"`).
F	**Source Fields**. The database columns to be aggregated (e.g., `sales_vol`).
O	**Operations**. The aggregation functions applied to F (e.g., SUM, COUNT).

Table 4: Unified Notation for Logic Reconstruction Components.

C Additional Experiments

C.1 Experimental Setup Details

We employ few-shot prompting with task-specific demonstrations (typically 6–8 examples) and deterministic decoding (temperature=0.0) to ensure reproducibility. The system integrates `python-pptx` (v0.6.21) for slide manipulation and PostgreSQL (v14.5) for data management, orchestrated via LangGraph (v0.6.4). All experiments are conducted on NVIDIA H100 GPUs using the vLLM framework for high-throughput inference.

C.2 Supplementary Generalization Evidence

We conduct two additional experiments to assess generalization. First, since evaluating closed-source models on the full benchmark is prohibitively expensive, we compare open-weight and closed-source models on 500 randomly sampled cases from the template-generated test set. Closed-source models significantly outperform

Algorithm 1 Generic Logic Reconstruction Pipeline

Require: Raw Data D; Logic Tuple $\mathcal{L} = (S, H, C, F, O)$ defined in Table 4.

// Step 1: Constraint Application (Split)
if T indicates Range Binning (e.g., `"-"` or `"-M"`) **then**
  {Apply binning with boundaries $[v_{start}, v_{end}]$ and step v_{step}}
  $D' \leftarrow \text{ApplyBinning}(D, \text{field} = k, \text{start} = v_{start}, \text{end} = v_{end}, \text{step} = v_{step}, \text{fmt} = T)$
else
  {Case: Standard Filtering (e.g., k = City, v_{start} = "Beijing")}
  $D' \leftarrow \{r \in D \mid \text{ApplyFilter}(r[k], \text{op} = T, \text{val} = v_{start})\}$
end if
// Step 2: Determine Grouping Keys
Initialize grouping set $G \leftarrow \emptyset$
if $S =$ `XC` **then**
  $G \leftarrow H_{row} \cup H_{col}$
else if $S =$ `CF` **then**
  $G \leftarrow H_{row}$
else if $S =$ `FC` **then**
  $G \leftarrow H_{col}$
end if
// Step 3: Aggregation (Apply)
$A \leftarrow \text{GroupBy}(D', G).\text{Aggregate}(F, O)$
Note: A contains grouping columns G and computed metric columns.
// Step 4: Layout Reshaping (Combine)
if $S =$ `XC` **then**
  {Pivot: Map H_{row} to index, H_{col} to columns, using the first metric}
  $T_{out} \leftarrow \text{PivotTable}(A, \text{index} = H_{row}, \text{columns} = H_{col}, \text{values} = \text{Col}(F, O))$
else if $S =$ `FC` **then**
  $T_{out} \leftarrow A^{\top}$ {Transpose: Flip metrics to rows to match template}
else
  $T_{out} \leftarrow A$ {Standard layout, no reshape needed}
end if
return T_{out}

open-weight ones, with GPT-5.4 achieving the highest task success rate at 78.01% / 76.66% under the closed/open settings, followed by Qwen-MAX at 75.43% / 72.20%. Among open-weight models, the ranking remains unchanged: GPT-OSS-120B remains the strongest at 67.50% / 63.47%, slightly outperforming Qwen3-80B at 66.18% / 60.82%. These results further support that end-to-end slide updating is highly capability-dependent, with stronger models consistently achieving better performance. Second, we compare model performance on human-authored slides (see Section A.7) against the same 500 templated cases. Trends remain similar: GPT-OSS-120B achieves 67.71% / 61.46% and Qwen3-14B achieves 33.33% / 25.00% under the closed/open settings, preserving the same open-weight ranking as the benchmark test set. This consistency supports our assumption that report slide updates are primarily layout-preserving edits on element contents, suggesting our benchmark measures generalizable slide-updating capability rather than template-specific matching.

	(A) 500 template-generated		(B) Human-authored	
Model	**Closed**	**Open**	**Closed**	**Open**
GPT-OSS-120B	67.50	63.47	67.71	61.46
GPT-OSS-20B	46.80	35.66	45.83	35.42
Qwen3-80B	66.18	60.82	64.58	59.38
Qwen3-30B	50.24	43.58	48.96	41.67
Qwen3-14B	35.71	27.34	33.33	25.00
Qwen-MAX	75.43	72.20	73.96	69.79
GPT-5.4	78.01	76.66	77.08	75.00

Table 5: Supplementary generalization results (task success rate%).

C.3 Case Study

We observe two representative failure modes in SlideAgent. The first is error accumulation: once an upstream module makes a mistake in slide parsing, instruction grounding, SQL generation, or tool execution, the error propagates through later stages and eventually leads to inconsistency among tables, charts, and textual conclusions. In this sense, a local mistake rarely remains local; it instead distorts the entire update chain. The second is the low precision of Summary Update: even when the preceding structured steps are correct, the final summary may still fail to express the updated results with sufficient numerical and semantic fidelity. This weakness reflects both the intrinsic difficulty of fact-aware text generation and the strictness of our evaluation criterion. We therefore use exact match for summary rewriting to enforce semantic and numerical correctness. For example, if second-hand housing sales in Liangxiang, Beijing increase from 836 to 1,355, the correct increase is about 62.1%; a rewritten summary reporting 63.2% is still counted as incorrect. These two failure modes show that the main challenge lies not only in executing each module correctly, but also in maintaining precise cross-module consistency through to the final narrative.

Template Variable	Source Column	Description
start_year	date_code	Extracted year from the start date of the analysis period (e.g., 2023).
end_year	date_code	Extracted year from the end date of the analysis period (e.g., 2024).
month	date_code	Truncated to month-level precision to represent the specific reference month (e.g., 2024-01).
city	city	Direct mapping to the target city.
block	block	Direct mapping to the administrative block.
project	project	Direct mapping to the residential project.

Table 6: **Field Mapping Dictionary**. This dictionary defines how template-level variables are derived from concrete database columns.

Parameter Variable	Candidate Value Set	Description
area_bin_step	{10, 15, 20, 25, 30}	Sets the interval width (m^2) for grouping properties by floor area.
price_bin_step	{0.75, 1.0, 1.5, 1.75, 2.0}	Sets the interval width (Million CNY) for grouping properties by total price.

Table 7: **Parameter Candidate Dictionary**. This dictionary defines the valid search space for parameters variable used in statistical functions (F_k).

Header	**Alias Variants** (Professional Selection)	Description
supply volume	{Supply Volume, New Listings, Listing Count, Market Supply, Total Listings, Supply (Units), New Supply}	Number of residential units listed for sale.
trade volume	{Trade Volume, Sales Volume, Transaction Count, Units Sold, Deal Count, Total Sales, Turnover Volume}	Number of residential units sold.
area range counts	{Volume by Area, Sales by Size, Count by Area, Area Segment Volume, Units by Size, Size Distribution}	Transaction count grouped by floor area segments.
price range counts	{Volume by Price, Sales by Price, Count by Price, Price Segment Volume, Units by Price, Price Distribution}	Transaction count grouped by total price intervals.
total supply area	{Total Supply Area, Supply Floor Area, Total Listing Area, Aggregate Supply (m^2), Supply Size, Listed Area}	Aggregate floor area of listed properties.
total trade area	{Total Trade Area, Total Sales Area, Transaction Area, Sold Floor Area, Aggregate Sales (m^2), Sold Area}	Aggregate floor area of sold properties.
avg price	{Avg. Total Price, Avg. House Price, Mean Transaction Price, Avg. Unit Cost, Per-Unit Price, Avg. Deal Price}	Average price per housing unit (not per sqm).
unit price	{Unit Price, Price per m^2, Avg. Price ($/m^2$), Sqm Price, Transaction Price ($/m^2$), Market Rate}	Average price per square meter.
area_range	{Area Range, Floor Area (m^2), Size Band, Property Size, Area Segment, Unit Size}	Grouping dimension for floor area intervals.
price_range	{Price Range, Total Price (Band), Price Segment, Value Tier, Budget Range, Price Bracket}	Grouping dimension for total price intervals.

Table 8: **Header Alias Dictionary**. This dictionary defines synonymous aliases for column and row headers across different statistical analyses. Multiple variants of the same metric enable linguistic diversity in table presentations.

Table 9: Definitions of real-estate market analysis variables. All variables are automatically calculated from the transaction database.

Variable Name	Description
City	Name of the target city for analysis (e.g., Beijing).
Block	Name of the specific residential block or sub-district.
Start_Year	The starting year of the analysis period (e.g., 2020).
End_Year	The ending year of the analysis period (e.g., 2024).
Start_Month	The starting month for monthly trend analysis.
End_Month	The ending month for monthly trend analysis.
Seg_SupplyDemand_Core_Area	Core demand area segment: floor area range where primary housing demand is most concentrated, indicating the core supply–demand space.
Seg_SupplyDemand_Upgrade_Area	Upgrade demand area segment: floor area range for improved-living demand, indicating the upgraded housing segment.
Total_Transaction_Units	Total transaction units during the observed period, reflecting overall market activity.
Modal_Price_Segment	Dominant transaction price range with the highest volume, representing the prevailing market price level.
Modal_Area_Segment	Dominant transaction area range with the highest volume, indicating mainstream housing size preferences.
Peak_Segment_Volume	Peak cell transaction volume in the price–area matrix, highlighting the most popular specific product subtype.
Dominant_Area_Segment	Label of the area segment with the highest transaction concentration (e.g., 80–100 m^2).
Dominant_Area_Segment_Volume	Total transaction volume within the dominant area segment.
Dominant_Price_Segment	Label of the price segment with the highest transaction concentration.
Dominant_Price_Segment_Volume	Total transaction volume within the dominant price segment.
Base_Period_Traded_Area	Total annual traded area (m^2) in the base year.
Terminal_Period_Traded_Area	Total annual traded area (m^2) in the terminal year.
Base_Period_Avg_Price	Average transaction price (CNY/m^2) in the base year.
Terminal_Period_Avg_Price	Average transaction price (CNY/m^2) in the terminal year.
Total_Area_Change_Pct	Percentage change of total traded area from base to terminal year (e.g., 0.25 = 25%), indicating market expansion or contraction.
Total_Price_Change_Pct	Percentage change in average transaction price from base to terminal year.
Area_Trend_Direction	Direction of traded area trend: “Increasing” or “Decreasing”.
Price_Trend_Direction	Direction of price trend: “Increasing” or “Decreasing”.
Absolute_Price_Change	Absolute price change (CNY/m^2) between terminal and base periods.
Supply_Trend_Direction	Direction of supply trend: “Increasing” or “Decreasing”.
Transaction_Trend_Direction	Direction of transaction volume trend: “Increasing” or “Decreasing”.
Base_Period_Supply_Units	Total number of supplied units in the base year.
Terminal_Period_Supply_Units	Total number of supplied units in the terminal year.
Base_Period_Transaction_Units	Total transaction units in the base year.
Terminal_Period_Transaction_Units	Total transaction units in the terminal year.
Total_Supply_Change_Pct	Percentage change in supply volume between base and terminal years.
Total_Transaction_Change_Pct	Percentage change in total transactions between base and terminal years.
Transaction_Change_Units	Absolute change in transaction units (terminal minus base).
Base_Period_Transaction_Price	Transaction price level in the base year.
Terminal_Period_Transaction_Price	Transaction price level in the terminal year.
Transaction_Change_Price	Absolute change in transaction price between base and terminal periods.

Function	Description
`create_slide`	Initializes a new slide page within the presentation deck. This function acts as the container for subsequent elements, allowing users to specify an optional slide layout template (e.g., title-only, blank).
`add_title`	Adds a prominent title element to the current slide context. This is typically used to define the primary topic or section header for the slide.
`add_text`	Inserts a text box containing specified content at given coordinates. This function supports multi-line text, bullet points, and basic rich text formatting for detailed descriptions.
`add_table`	Renders a structured data table on the slide. The function automatically calculates and adjusts cell dimensions (row heights and column widths) to fit the provided dataset visually.
`add_line_chart`	Generates a line chart visualization to display trends over time or continuous variables. It supports plotting multiple data series for comparative analysis.
`add_bar_chart`	Creates a bar chart to compare quantitative values across distinct categories. Users can specify orientation (vertical or horizontal) to best fit the data distribution.

Table 10: Description of core functions for slide generation.

Category	Dimension	Quantity
Dataset Splits	Total Samples	20,036
	Train Set	12,022
	Validation Set	4,007
	Test Set	4,007
Template Statistics	Analysis Themes	6
	Statistical Operations	11
	Template Types	34
	Avg. Elements per Slide	4.2
	Source Slides (Seeds)	7,685
Instruction Statistics	Basic Instructions	13,016
	Customized Instructions	7,020
	Avg. Instruction Length	20.11
	Avg. Parameters per Instruction	3.8
Annotation Components	Natural Language Instructions	20,036
	Slide Titles and Captions	45,854
	SQL Queries	20,036
	Summaries	20,036
	Statistical Tables (Total)	25,041
	Base Table	4,919
	Line Chart Table	9,746
	Bar Chart Table	10,376

Table 11: Detailed statistics of the DynaSlide dataset. The dataset, derived from **7,685 source slides**, covers four core elements (titles, tables, charts, summaries) and includes full annotations for SQL, logic, and visual layouts.

Table 12: Full Template Specification for Real-Estate Market Text Generation. Variables are denoted in monospace and are automatically populated during generation.

Title Template	Function	Caption Template	Summary Template
1. Block Area Segment Distribution 2. Analysis of Block Area Division 3. Distribution of Block Area Segments	Supply-Transaction Unit Statistics	1. Start_Year-End_Year Supply and Transaction Unit Statistics in City's Block 2. City Block: Supply & Sales Volume Analysis, Start_Year-End_Year 3. Analysis of Property Units Supplied vs. Sold in City's Block (Start_Year-End_Year)	1. From Start_Year to End_Year, Block's core supply-demand area was Seg_SupplyDemand_Core_Area m^2, with the upgrade-oriented segment centered on Seg_SupplyDemand_Upgrade_Area m^2. 2. Between Start_Year and End_Year, the market structure in Block was defined by a core demand range of Seg_SupplyDemand_Core_Area m^2 and an upgrade tier of Seg_SupplyDemand_Upgrade_Area m^2. 3. The Block sector exhibited a dual-tier segmentation from Start_Year-End_Year: a primary volume cluster at Seg_SupplyDemand_Core_Area m^2 and a secondary upgrade cluster at Seg_SupplyDemand_Upgrade_Area m^2.

Continued on next page...

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. New-House Cross-Structure Analysis 2. New Residential Portfolio Composition 3. New Construction Inventory Structure Analysis	Area × Price Cross Pivot	1. Start_Year-End_Year City Block Area and Total Price Cross Statistics 2. Cross-Analysis of Property Size and Price Points in City's Block (Start_Year-End_Year) 3. City Block: Correlation between Unit Area and Total Price (Start_Year-End_Year)	1. From Start_Year to End_Year, a total of Total_Transaction_Units units were transacted, with the Modal_Price_Segment price segment and Modal_Area_Segment area segment having the highest transactions at Peak_Segment_Volume units. 2. Out of Total_Transaction_Units total transactions during Start_Year-End_Year, the peak velocity of Peak_Segment_Volume units occurred at the intersection of the Modal_Price_Segment price band and Modal_Area_Segment area band. 3. The period Start_Year-End_Year saw Total_Transaction_Units total sales; the most active cross-segment was Modal_Price_Segment combined with Modal_Area_Segment, contributing Peak_Segment_Volume units.
1. New-House Cross-Structure Analysis 2. New Residential Portfolio Composition 3. New Construction Inventory Structure Analysis	Area Segment Distribution	1.Start_Year-End_Year City Block Total Area Segment Distribution Statistics 2. Distribution of Transactions by Property Size Segment in City's Block (Start_Year-End_Year) 3. City Block: Analysis of Market Share by Unit Area Brackets (Start_Year-End_Year)	1. Mainstream types concentrate in Dominant_Area_Segment segments, totaling Dominant_Area_Segment_Volume units. 2. A volume of Dominant_Area_Segment_Volume units indicates that the Dominant_Area_Segment range represents the dominant area concentration. 3. The Dominant_Area_Segment typology emerged as the mainstream segment, amassing a total of Dominant_Area_Segment_Volume units.

Continued on next page...

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. New-House Cross-Structure Analysis 2. New Residential Portfolio Composition 3. New Construction Inventory Structure Analysis	Price Segment Distribution	1. Start_Year-End_Year City Block Total Price Segment Distribution Statistics 2. Distribution of Transactions by Price Point Segment in City's Block (Start_Year-End_Year) 3. Sales Breakdown by Price Range Categories for City's Block, Start_Year-End_Year	1. Mainstream types concentrate in Dominant_Price_Segment segments, totaling Dominant_Price_Segment_Volume units. 2. The Dominant_Price_Segment price bracket captured the majority of interest, accumulating Dominant_Price_Segment_Volume units. 3. With Dominant_Price_Segment_Volume units, the Dominant_Price_Segment segment constitutes the primary price concentration for the sector.
1. Resale-House Cross-Structure Analysis 2. Resale Residential Portfolio Assessment 3. Secondary Market Inventory Structure Study	Area × Price Cross Pivot	1. Start_Year-End_Year City Block Resale House Area and Total Price Cross Statistics 2. Resale Market: Cross-Analysis of Property Size and Price in City's Block (Start_Year-End_Year) 3. Statistical Profile of Resale Homes by Area vs. Total Price in City's Block (Start_Year-End_Year)	1. From Start_Year to End_Year, a total of Total_Transaction_Units units were transacted, with the Modal_Price_Segment price segment and Modal_Area_Segment area segment having the highest transactions at Peak_Segment_Volume units. 2. Resale activity for Start_Year-End_Year totaled Total_Transaction_Units units, peaked by Peak_Segment_Volume sales in the Modal_Price_Segment / Modal_Area_Segment cross-segment. 3. The Modal_Price_Segment and Modal_Area_Segment cohorts led the resale market with Peak_Segment_Volume units, driving a cumulative volume of Total_Transaction_Units.
1. Resale-House Cross-Structure Analysis 2. Resale Residential Portfolio Assessment 3. Secondary Market Inventory Structure Study	Area Segment Distribution	1. Start_Year-End_Year City Block Resale House Total Area Segment Distribution Statistics 2. Resale Market Transaction Distribution by Property Size in City's Block (Start_Year-End_Year) 3. Breakdown of Existing Home Sales by Size Category in City's Block, Start_Year-End_Year	1. Mainstream types concentrate in the Dominant_Area_Segment segments, totaling Dominant_Area_Segment_Volume units. 2. The resale inventory is heavily weighted in the Dominant_Area_Segment range, which accounts for Dominant_Area_Segment_Volume units. 3. Accounting for Dominant_Area_Segment_Volume units, the Dominant_Area_Segment category stands out as the primary resale typology.

Continued on next page...

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. Resale-House Cross-Structure Analysis 2. Resale Residential Portfolio Assessment 3. Secondary Market Inventory Structure Study	Price Segment Distribution	1. Start_Year-End_Year City Block Resale House Total Price Segment Distribution Statistics 2. Resale Market Transaction Distribution by Price Point in City's Block (Start_Year-End_Year) 3. Breakdown of Existing Home Sales by Price Range in City's Block, Start_Year-End_Year	1. Mainstream types concentrate in the Dominant_Price_Segment segments, totaling Dominant_Price_Segment_Volume units. 2. The Dominant_Price_Segment price tier represents the core resale market, comprising Dominant_Price_Segment_Volume units. 3. A total of Dominant_Price_Segment_Volume resale units clustered within the Dominant_Price_Segment_Volume price band.
1. New-House Market Capacity Analysis 2. New Construction Volume & Supply Capacity 3. Emerging Residential Market Scale Evaluation	Historical Capacity Summary	1. City Block Historical Capacity Summary Statistics (Start_Year-End_Year) 2. Historical Market Volume Summary for City's Block (Start_Year-End_Year) 3. Summary of Past Market Scale Statistics for City's Block, Start_Year-End_Year	1. From Start_Year to End_Year, Block's traded area Area_Trend_Direction Total_Area_Change_Pct% from Base_Period_Traded_Area m^2 to Terminal_Period_Traded_Area m^2, and the average valuation also Price_Trend_Direction Total_Price_Change_Pct% from Base_Period_Avg_Price to Terminal_Period_Avg_Price yuan/m^2. 2. Over the Start_Year-End_Year period, the sector saw traded area Area_Trend_Direction by Total_Area_Change_Pct% (moving from Base_Period_Traded_Area to Terminal_Period_Traded_Area m^2), while valuations Price_Trend_Direction by Total_Price_Change_Pct yuan/m^2. 3. Starting at Base_Period_Traded_Area m^2 and Base_Period_Avg_Price yuan/m^2, the market Area_Trend_Direction to Terminal_Period_Traded_Area m^2 and Terminal_Period_Avg_Price yuan/m^2 respectively, marking a volume shift of Total_Area_Change_Pct% and a price shift of Absolute_Price_Change yuan/m^2.

Continued on next page...

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. New-House Market Capacity Analysis 2. New Construction Volume & Supply Capacity 3. Emerging Residential Market Scale Evaluation	Annual Supply-Demand Comparison	1. City Block Annual Supply-Demand Comparison Analysis (Start_Year-End_Year) 2. City Block: Annual Comparison of Market Supply and Transaction Volume (Start_Year-End_Year) 3. Analysis of Annual Supply-Demand Balance in City's Block (Start_Year-End_Year)	1. From Start_Year to End_Year, new listings in this sector Supply_Trend_Direction from Base_Period_Supply_Units units to Terminal_Period_Supply_Units units (a Supply_Trend_Direction of Total_Supply_Change_Pct%), and transaction volume Transaction_Trend_Direction from Base_Period_Transaction_Units units to Terminal_Period_Transaction_Units units (a Transaction_Trend_Direction of Total_Transaction_Change_Pct%). 2. While listings Supply_Trend_Direction by Total_Supply_Change_Pct% (reaching Terminal_Period_Supply_Units units), transactions simultaneously Transaction_Trend_Direction by Total_Transaction_Change_Pct% (ending at Terminal_Period_Transaction_Units units) between Start_Year and End_Year. 3. Comparing Start_Year to End_Year, supply Supply_Trend_Direction to Terminal_Period_Supply_Units (Total_Supply_Change_Pct% Supply_Trend_Direction), and demand Transaction_Trend_Direction to Terminal_Period_Transaction_Units (Total_Transaction_Change_Pct% Transaction_Trend_Direction).

Continued on next page...

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. New-House Market Capacity Analysis 2. New Construction Volume & Supply Capacity 3. Emerging Residential Market Scale Evaluation	Supply-Transaction Area	1. City Block : Historical Supply and Transaction Area Statistics (Start_Year-End_Year) 2. Statistical Review of Historical Supply and Transaction Area for City's Block (Start_Year-End_Year) 3. Historical Data: Supplied vs. Sold Area in City's Block (Start_Year-End_Year)	1. From Start_Year to End_Year, inventory in this region Supply_Trend_Direction by Total_Supply_Change_Pct%, while the transaction area Transaction_Trend_Direction by Total_Transaction_Change_Pct%. 2. Area-wise inventory Supply_Trend_Direction at a Total_Supply_Change_Pct% rate, contrasting with the transaction area which Transaction_Trend_Direction by Total_Transaction_Change_Pct% through End_Year. 3. The region experienced a Total_Supply_Change_Pct% Supply_Trend_Direction in supply area and a Total_Transaction_Change_Pct% Transaction_Trend_Direction in sold area between Start_Year and End_Year.
1. Resale-House Capacity & Structure 2. Resale Market Scale & Product Breakdown 3. Secondary Market Stock & Unit Composition	Historical Delivery Metrics	1. City Block Resale House: Overview of Historical Delivery Metrics (Start_Year-End_Year) 2. Resale Market in City's Block: A Look at Historical Transaction Metrics (Start_Year-End_Year) 3. Historical Performance Overview for the Resale Market in City's Block (Start_Year-End_Year)	1. From Start_Year to End_Year, the number of resale transactions in this area Transaction_Trend_Direction from Base_Period_Transaction_Units units to Terminal_Period_Transaction_Units units, with a cumulative Transaction_Trend_Direction of Transaction_Change_Units units, representing a Transaction_Trend_Direction of Total_Transaction_Change_Pct%. 2. Resale volumes moved from Base_Period_Transaction_Units to Terminal_Period_Transaction_Units units (Start_Year-End_Year), marking a Transaction_Change_Units-unit Transaction_Trend_Direction or a Total_Transaction_Change_Pct% change. 3. The market observed a Transaction_Trend_Direction in resales to Transaction_Change_Units units, a Total_Transaction_Change_Pct% Transaction_Trend_Direction relative to the Base_Period_Transaction_Units-unit baseline in Start_Year.

Continued on next page...

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. Resale-House Capacity & Structure 2. Resale Market Scale & Product Breakdown 3. Secondary Market Stock & Unit Composition	Annual Delivery Unit Count	1. Historical Delivery Unit Count Statistics for City Block Resale House (Start_Year-End_Year) 2. Historical Transaction Volume (Units) for Resale Homes in City's Block (Start_Year-End_Year) 3. Unit Sales Statistics for the Existing Home Market in City's Block (Start_Year-End_Year)	1. From Start_Year to End_Year, the number of resale house transactions in this area Transaction_Trend_Direction from Base_Period_Transaction_Units units to Terminal_Period_Transaction_Units units, with a cumulative Transaction_Trend_Direction of Transaction_Change_Units units. 2. A net Transaction_Trend_Direction of Transaction_Change_Units units was recorded as resale volume Transaction_Trend_Direction from Base_Period_Transaction_Units to Terminal_Period_Transaction_Units between Start_Year and End_Year. 3. The trend Transaction_Trend_Direction brought resale figures to Terminal_Period_Transaction_Units by End_Year, a Transaction_Trend_Direction of Transaction_Change_Units from the start of the period.
1. Resale-House Capacity & Structure 2. Resale Market Scale & Product Breakdown 3. Secondary Market Stock & Unit Composition	Annual Average Price Trend	1.City Block Resale House: Start_Year-End_Year Annual Average Price Trend Analysis 2. Annual Average Price Evolution for Resale Homes in City's Block (Start_Year-End_Year) 3.Analysis of Annual Mean Price Fluctuation for Resale Properties in City's Block, Start_Year-End_Year	1. Resale average prices trended Price_Trend_Direction from Base_Period_Transaction_Price yuan/m^2 in Start_Year to Terminal_Period_Transaction_Price yuan/m^2 in End_Year (Transaction_Change_Price). 2. The market valuation followed a Price_Trend_Direction path, adjusting from Base_Period_Transaction_Price to Terminal_Period_Transaction_Price yuan/m^2 amid Transaction_Change_Price conditions. 3. Between Start_Year and End_Year, prices Price_Trend_Direction to Terminal_Period_Transaction_Price yuan/m^2 from a base of Base_Period_Transaction_Price, characterizing a Transaction_Change_Price market.

Continued on next page...

PPT Layout Parsing Prompt

You are an AI assistant specializing in document layout analysis. Your task is to analyze a PPT slide image and identify specific components: Main Title, Body Text, Chart Caption, and Table/Bar Chart/Line Chart. Please strictly follow the JSON format below to return the results. The output should be a list where each element represents a recognized component.

shape_type: Type of the recognized component. Allowed values: 'slide-title', 'body-text', 'caption', 'table', 'chart-bar', 'chart-line'.
content: The text content of the component. (Note: For 'table', 'chart-bar', and 'chart-line', keep this field empty string).
bbox: Bounding box coordinates of the component in the image, formatted as [x_min, y_min, x_max, y_max].

If a specific component type is not found, do not include it in the list. Do not output any explanations, extra text, or markdown formatting. Return only a raw JSON array.

Figure 5: The prompt used for multimodal layout parsing to predict semantic labels and bounding boxes.

Table 12 – continued from previous page

Title Template	Function	Caption Template	Summary Template
1. Neighborhood Price Trend 2. Regional Price Movement & Trajectory 3. Community Value Trend Analysis	Monthly Average Price Trend	1. `City` `Block` Monthly Average Price Trend Analysis, `Start_Month-End_Month` 2. Analysis of Monthly Average Price Movements in `City`'s `Block` (`Start_Month-End_Month`) 3. Month-over-Month Average Price Fluctuation for `City`'s `Block` (`Start_Month-End_Month`)	1. Over `Start_Month-End_Month`, the average transaction price of apartments in this community went from `Base_Period_Transaction_Price` yuan/m^2 `Transaction_Trend_Direction` to `Terminal_Period_Transaction_Price` yuan/m^2, with a cumulative `Price_Trend_Direction` of `Transaction_Change_Price` yuan/m^2. 2. Between `Start_Month` and `End_Month`, prices `Transaction_Trend_Direction` by `Transaction_Change_Price` yuan/m^2, shifting the average from `Base_Period_Transaction_Price` to `Terminal_Period_Transaction_Price` yuan/m^2. 3. A net `Transaction_Trend_Direction` of `Transaction_Change_Price` yuan/m^2 was observed Over `Start_Month-End_Month`, as prices moved to `Terminal_Period_Transaction_Price` yuan/m^2 from `Base_Period_Transaction_Price`.

Data Source Extraction Prompt

Please perform the following data extraction task. Based on the given slide params (including the caption title, row headers, and column headers), generate a JSON object that meets the requirements. Specific steps are as follows:

1. **Parse parameter information：** Carefully analyze the provided slide_params, extracting the caption, row_headers, and column_headers.

2. **Extract key elements：**
   - Table affiliation: Clearly identify the database table to which the data belongs.
   - Block: Identify geographic region or block information from the caption (e.g., "Liangxiang" or "Hengli Town").
   - Time range: Extract the start date (start_date) and end date (end_date) from the caption, in the format 'YYYY-MM-DD'.
   - Retrieval fields: Infer the required fields based on row_headers and column_headers.

3. **Specific parameter explanations:：**
   - Database: Includes ['beijing_new_house', 'beijing_resale_house'], where 'beijing_new_house' corresponds to Beijing new house data, 'beijing_resale_house' corresponds to Beijing resale house data, and other cities (Guangzhou, Shenzhen) follow the same pattern.
   - Select_columns: Includes ['date_code', 'supply_sets', 'trade_sets', 'dim_area', 'dim_price', 'dim_unit_price'].
     Among them, 'date_code' indicates the date, 'supply_sets' indicates supply units, 'trade_sets' indicates transaction units, 'dim_area' indicates property area, 'dim_price' indicates property price, 'dim_unit_price' indicates price per square meter, and 'project' indicates neighborhood project name.

Examples:

Slide_params:{
"caption": "2020-2022 Beijing Liangxiang Area and Total Price Cross Statistics",
"row_headers": ["0-20m²", "20-40m²", "40-60m²", "60-80m²", "80-100m²", "100-120m²", "120-140m²", "140-160m²", "160-180m²", "180-200m²", "200-220m²", "220-240m²", "240-260m²", "260-280m²", "280-300m²", "≥300m²"],
"column_headers": ["Supply Count (Sets)", "Deliv. Count (Sets)"]
}

Response:

```
{{
  "connection": {{
        "table": ["beijing_new_house"],
        }},
  "select_columns": ["supply_sets", "trade_sets", "dim_area"],
  "filters": {{
        "city": "Beijing",
        "block": "Liangxiang",
        "start_date": "2020-01-01",
        "end_date": "2022-12-31",
        }},
}}
```

Figure 6: The prompt for data source extraction, mapping slide content to database schema slots.

Function Logic Extraction Prompt

Please execute the following data extraction task. Based on the provided slide_params (including titles, row/column headers), select the most appropriate function from the 11 predefined options and generate a compliant JSON object.

A. Function Library Definitions

Select the fun_tool.fun that best matches the input from the following list:

1. get_supply_sales_counts_stats:
Trigger Condition: Row headers represent Area Ranges (e.g., "80-100m²"), and column headers contain counts for both Supply and Transaction (Trade).
Usage: Analyzing the supply-demand structure (supply units vs. transaction units) across different area segments.

2. compute_area_num_stats:
Trigger Condition: Row headers represent Area Ranges, and column headers involve only Total Distribution (e.g., "Count", "Distribution") without distinguishing between supply and demand.
Usage: Statistical distribution of housing unit counts across different area segments.

3. compute_price_num_stats:
Trigger Condition: Row headers represent Total Price Ranges (e.g., "3.0-4.0M").
Usage: Statistical distribution of housing unit counts across different total price segments.

4. compute_area_price_cross_stats:
Trigger Condition: A two-dimensional cross-analysis. Typically, row and column headers correspond to Area Segments and Total Price Segments respectively, or the title explicitly mentions "Cross Analysis" or "Area vs. Total Price."
Usage: Generating a cross-tabulation heat map (Area Range × Total Price Range).

5. compute_annual_traded_units:
Trigger Condition: Row headers represent Years, and column headers focus on Units/Sets, containing both supply and transactions.
Usage: Comparison of annual trends in supply and demand units.

6. compute_annual_traded_area:
Trigger Condition: Row headers represent Years, and column headers focus on Area (sqm).
Usage: Comparison of annual trends in supply and demand area.

7. compute_market_capacity:
Trigger Condition: Row headers represent Years, and column headers are comprehensive (containing multi-dimensional metrics such as Area, Units, Average Price).
Usage: Comprehensive annual market capacity analysis.

8. compute_house_total_and_avg_price:
Trigger Condition: Row headers represent Years, and column headers include Transaction Volume and Average Price, etc.
Usage: Overview of historical performance metrics (trade_counts, avg_unit_price, dim_area).

9. compute_house_transaction_count_distribution:
Trigger Condition: Row headers represent Years, and column headers focus solely on Transaction Units.
Usage: Analysis of annual transaction volume trends.

10. compute_house_avg_price_distribution:
Trigger Condition: Row headers represent Years, and column headers focus solely on Average Price.
Usage: Analysis of annual average price trends.

11. get_housing_price_trend:
Trigger Condition: Row headers represent Monthly Dates (e.g., "2023-01", "2023-02") rather than pure years.
Usage: Monthly price trend analysis.

B. Argument Inference

Calculate fun_tool.args based on the content of row_headers:
area_range_size: If row headers are area segments (e.g., "80-100m²"), calculate the step size (100 - 80 = 20). Ignore if not applicable.
price_range_size: If row headers are price segments (e.g., "3.0-3.5M"), calculate the step size (3.5 - 3.0 = 0.5). Ignore if not applicable.
Note: Even for trend-related functions, if specific interval filtering is implied in the context, these parameters should be extracted; otherwise, leave them empty.

Figure 7: The prompt for closed-domain function logic extraction via predefined tool invocation.

Function Logic Extraction Prompt

Please execute the following table understanding task and strictly return the result as a valid JSON array only:

1) **Parameter Extraction:**
Analyze the input table_caption and table_data to extract key parameters. Organize these parameters into a 4-tuple structure: [table_type, header_info, source_field, statistic_method]. Return this tuple as an element within a JSON array.

2) **Tuple Schema Definition:**
- **Table_type** : Reflects the structural logic of the table.
  - Enum: ["field-constraint", "constraint-field", "cross-constraint"]
  - Definitions:
    - field-constraint: Column headers represent specific constraints (e.g., ranges), while row headers represent metrics/fields.
    - constraint-field: Column headers represent metrics/fields, while row headers represent constraints.
    - cross-constraint: Both row and column headers represent different constraints, with the intersection being the statistical value.

- **Header_info** : Contains details about the headers and their implied constraints.
  - headers (Array): The original text of the headers extracted directly from the table without modification.
  - constraint_detail (Object): The parsed logic of the constraints.
    - type (String): The category of the constraint. Enum: ["area_range", "price_range", "year", "month"].
    - template (String): Parameterized template for dynamic range formatting (e.g., {{}}-{{}}M).
    - bounds (Array): An array defining the boundary values [start, end].
    - step (Number): The hyperparameter indicating the step size (e.g., 0.75).
    - Note: For cross-constraint types, this field should be an array containing two distinct constraint objects.

- **Source_field**: The inferred database field corresponding to the table content.
  - Enum: ["supply_sets", "trade_sets", "dim_area", "dim_price", "dim_unit_price"]
  - Mappings:
    - supply_sets: Number of supplied units.
    - trade_sets: Number of traded/sold units.
    - dim_area: Property area.
    - dim_price: Property total price.
    - dim_unit_price: Price per square meter.
- **Statistic_method** : The method used for data aggregation.
  - Enum: ["count", "sum", "mean"]
  - Mappings:
    - count: Frequency or count (e.g., number of units).
    - sum: Total summation (e.g., total area).
    - mean: Average value (e.g., average price).

3) **Output Format:**
Output ONLY a valid JSON array containing the 4-tuple(s).Strictly NO additional text, preambles, explanations, markdown code blocks (e.g., ```json), or trailing whitespace.

Figure 8: The prompt for open-domain logic extraction, decomposing unknown logic into atomic parameters.

User Instruction Parsing Prompt

Based on the user's query, extract the following information and generate a strict JSON object. Fields include:

city: city name (string)
block: block name (string)
project: project name (string)
start_date: start date (string, format 'YYYY-MM-DD')
end_date: end date (string, format 'YYYY-MM-DD')
area_range_size: area range size (number or string, such as "20")
price_range_size: price range size (number or string, such as "1")
General rule: If a field is not explicitly mentioned in the user's query, fill it with default.

Requirements:

1.Only output a valid JSON object, without any explanations, extra text, or code block markers.
2.All dates must use the 'YYYY-MM-DD' format.
3.JSON keys must exactly match the above field names, with values enclosed in double quotes.

Examples:
Query: Based on this slide, please generate a slide for the Beijing Huairou District sector from 2021 to 2023.
Response:

```
{{
  "city": "Beijing",
  "block": "Huairou District",
  "project": "default",
  "start_date": "2021-01-01",
  "end_date": "2023-12-31",
  "area_range_size": "default",
  "price_range_size": "default"
}}
```

Figure 9: The prompt for user instruction parsing, modeled as a parameter state update task.

SQL Extraction Prompt

Please execute the following SQL query generation task and strictly return the result according to the requirement of "only output a valid JSON array":

Task requirements:

1) Parse parameters：
   - Extract connection.table, select columns, and filters from slide_params.
2) Build WHERE clause：
   - Condition fields: city, block, project, date_code (generated from start_date/end_date).
   - Ignore conditions with value default (do not include in WHERE).
   - For non-default string values (such as city/region/block/project name), enclose in single quotes, e.g., city = 'Beijing'.
   - Date range: If start_date and/or end_date are provided, use:
     - Both provided: date_code >= 'YYYY-MM-DD' AND date_code <= 'YYYY-MM-DD'
     - Only start_date: date_code >= 'YYYY-MM-DD'
     - Only end_date: date_code <= 'YYYY-MM-DD'
3) Generate SQL：
   - FROM uses schema prefix: public.<table>
   - SELECT columns output in input order.
4) Output format (extremely important：
   - Only output a valid JSON array, with array elements as strings (one SQL string per element).
   - Use standard JSON double quotes to wrap strings; single quotes inside strings are allowed (SQL text).
   - Do not output anything other than JSON: no prefix/suffix text, no Chinese quotes, no special quotes, no comments, no Markdown code block markers, no extra line breaks.

Examples：

slide_params:{{
'connection': {{'table': ['beijing_new_house']}},
'select_columns': ['supply_sets', 'trade_sets', 'dim_area'],
'filters': {{
'city': Beijing,
'block': Huairou District,
'project': default,
'start_date': 2020-01-01,
'end_date': 2022-12-31,
'area_range_size': default,
'price_range_size': default
}}
}}

Output： "SELECT supply_sets, trade_sets, dim_area FROM public.beijing_new_house WHERE city = 'Beijing' AND block = 'Huairou District' AND date_code >= '2020-01-01' AND date_code <= '2022-12-31'"

Figure 10: The prompt for SQL generation, converting updated parameters into executable database queries.

Summary Update Prompt

You need to refer to the provided template_data and template_conclusion to generate a corresponding conclusion for the new data. Please strictly follow the following steps to ensure the output is professional, concise, and highly matched to the data:

(1) Data Understanding: Carefully analyze the structure of the template data, including key variables (such as location, time, numerical indicators, trend descriptions, etc.), and identify the generation logic of the template conclusion, such as how to highlight market characteristics, compare changes, or summarize insights based on the data. Pay special attention to professional terms in the real estate field, for example, "generally, units above 140㎡ are called upgraded units."

(2)Information Replacement and Adaptation: For the new data, map and accurately replace the key elements in the template conclusion one by one, including locations (such as the phrasing "this area" or "this community"), numerical indicators (such as area, price, transaction volume), and trend descriptions (such as growth rate or proportion). If the new data ntroduces new variables (such as different housing type distributions or market dynamics), adjust accordingly to maintain logical consistency, while retaining the overall narrative style and professional definitions of the template.

(3)Conclusion Generation: Ensure that the generated new conclusion maintains the logical structure of the original template conclusion while being completely consistent with the new data, and present it in clear and professional language.

Examples:

template_data:
2021-2023 Supply and Transaction Unit Statistics in Beijing's Dougezhuang-Heizhuanghu

	Supply Count (Sets)	Deliv. Count (Sets)
0-20m²	136	18
20-40m²	18	0
40-60m²	15	21
60-80m²	122	264
.....		
240-260m²	28	7
260-280m²	9	28
280-300m²	0	8
≥300m²	70	101

template_conclusion:
From 2021-2023, Dougezhuang-Heizhuanghu's core supply-demand area was 80-100m², with upgraded units centered on 160-180m².

data:
2020-2022 Supply and Transaction Unit Statistics in Beijing's Beiqijia

	supply_counts	trade_counts
0-20m²	136	18
20-40m²	18	0
40-60m²	15	21
60-80m²	122	264
80-100m²	230	206
.....		
240-260m²	28	7
260-280m²	9	28
280-300m²	0	8
≥300m²	70	101

conclusion:
From 2020-2022, Beiqijia's core supply-demand area was 80-100m², with upgraded units centered on ≥300m².

Figure 11: The prompt for fact-aware summary update based on data changes.