LLM-Assisted MAML Authoring: A Prompt-Chain Approach for Layout and Interaction Generation

Junyong Moon
Computer Science, NYUAD
jm8899@nyu.edu

Advised by: Yasir Zaki

ABSTRACT

This report investigates whether large language models (LLMs) can reliably automate website authoring in Mobile Application Markup Language (MAML), a lightweight, absolute-positioning-based language designed for performance under constrained environments but still burdensome to author due to manual object placement. To reduce this overhead, we develop a constraint-driven, three-stage prompt pipeline that decomposes a natural-language request into a structured build plan, a schema-compliant UI and interaction specification, and editor-ready JSON-line MAML code. Results indicate that layout generation is robust under fixed-canvas and strict schema constraints, scaling to moderately complex, multi-section single-page interfaces with dozens of elements, while interaction generation via MAMLScript is feasible but significantly more fragile and sensitive to prompt rigidity, formatting, and user-specified intent. We further explore reconstruction pipelines for screenshot-to-MAML and HTML-to-MAML translation, finding that screenshots provide strong spatial grounding but limited asset fidelity, whereas HTML improves semantic and resource recovery; a hybrid screenshot+HTML approach appears most promising but depends on reliable webpage extraction. Overall, the work demonstrates the practicality of pipeline-based LLM assistance for MAML authoring, identifies interaction synthesis and extraction as key bottlenecks, and outlines future directions for benchmarking, user studies, and tighter editor integration.

KEYWORDS

Mobile Application Markup Language (MAML), Large Language Models (LLMs), Prompt Engineering, Multi-stage Prompt Pipeline, UI Code Generation, Absolute Positioning, Screenshot-to-code Reconstruction

Reference Format:
Junyong Moon. 2025. LLM-Assisted MAML Authoring: A Prompt-Chain Approach for Layout and Interaction Generation. In *NYUAD Capstone Project 1 Reports, Fall 2025, Abu Dhabi, UAE.* 11 pages.

1 INTRODUCTION

1.1 Motivation

Mobile Application Markup Language (MAML) was introduced to deliver interactive web experiences under constrained bandwidth and compute conditions by reducing reliance on the conventional HTML/CSS/JavaScript stack in favor of a minimal, absolute-positioning-based authoring model. In the original MAML framing, the intent is to reduce the overhead associated with modern webpages, including complex dependency graphs and expensive "reflows and repaints" induced by DOM mutations, particularly on low-end devices and resource-constrained network conditions. Within this design space, MAML adopts absolute positioning to support a flat DOM layout, improving element search and updates at runtime while giving up some of the responsiveness benefits of flow-based layouts.

Despite these runtime and bandwidth advantages, MAML still presents a substantial authoring burden. Even with a dedicated editor workflow, creating a page requires handcrafting and iteratively adjusting pixel-precise coordinates for each object, which becomes the dominant time sink in page creation. This challenge is especially salient for non-technical users: while MAML is simpler than a full HTML/CSS/JS workflow, learning to translate an interface idea into dozens of correctly positioned objects remains a challenge. These concerns were explicitly identified in the project proposal, which motivates automation as a way to reduce manual effort while preserving the lightweight goals of MAML.

This report is submitted to NYUAD's capstone repository in fulfillment of NYUAD's Computer Science major graduation requirements.

جامعة نيويورك أبوظبي
NYU ABU DHABI

Capstone Project 1, Fall 2025, Abu Dhabi, UAE

In parallel, recent UI automation and "design-to-code" efforts argue that translating interface intent, mockups, or screenshots into code can reduce development time and avoid repetitive human error. In practice, the most time-consuming part of MAML authoring is manually positioning objects.

1.2 Core Research Question

This project is guided by a single, concrete research question:

Can an LLM successfully and reliably generate web objects in MAML?

"Successfully" is defined as producing structurally valid, renderable MAML object lines that conform to the schema and closely match a user's intended layout and interactions, thereby reducing the need for manual editing.

1.3 Contributions

This report documents four primary contributions:

(1) **A stabilized three-stage prompt chain** that decomposes natural-language website requests into (i) a structured build plan, (ii) a complete UI and interaction specification, and (iii) editor-ready JSON-line MAML code.
(2) **A constraint-driven interaction synthesis strategy** that generates MAMLScript only within its supported listener/trigger set, using strict formatting rules and ID-mapping requirements to preserve executability.
(3) **Exploratory reconstruction/translation pipelines** for generating MAML from alternative inputs, including screenshot-to-MAML reconstruction and HTML-to-MAML translation, with a hybrid screenshot+HTML approach motivated by complementary strengths (layout grounding vs asset/semantic grounding).
(4) **An empirical findings narrative** organized around five research questions (RQ1–RQ5), summarizing what was feasible, what required constraints, and where the approach breaks down, alongside a concrete roadmap for future evaluation (model comparison and user study).

2 BACKGROUND AND RELATED WORK

2.1 MAML as a constrained authoring target

MAML was proposed as a lightweight alternative to conventional HTML/CSS/JavaScript stacks, aiming to reduce runtime overhead and improve performance in bandwidth- and compute-constrained settings by using a simplified object model and rendering approach [10]. A defining design choice is a flat, absolutely positioned layout representation, which can make runtime updates predictable but shifts substantial effort into manual, pixel-precise authoring [10]. This project treats those constraints as an opportunity: a strict schema and fixed-canvas geometry create a controlled setting for LLM-driven UI synthesis.

2.2 Design-to-code and hierarchical UI synthesis

Design-to-code research has explored generating UI code from screenshots and structured design artifacts, often emphasizing decomposition to improve correctness. WebCode2M provides a real-world dataset that highlights the diversity and complexity of webpage designs and motivates more structured generation and evaluation methodologies [4]. UICopilot demonstrates hierarchical UI synthesis via staged generation, using decomposition to reduce hallucination and improve structural validity when mapping visual designs to code [5]. These ideas directly motivate this report's central methodological choice: generating MAML via a pipeline (planning → specification → code emission) rather than a one-shot prompt.

2.3 LLMs for web development

Recent work also studies LLMs as practical tools for web development, including end-user oriented "describe-and-generate" workflows and comparative evaluations of model behavior. A dedicated line of work explores LLM-based end-user website generation and the usability implications of natural-language-to-code authoring [2]. Comparative studies of LLM code generation for web development further suggest that reliability depends heavily on task framing and prompts [9]. Systems that incorporate retrieval (e.g., RAG) for generation in code or templates, such as code-based UI generation pipelines, generate more accurate and richer output even when the target output is standard web code [11].

2.4 GUI agents and extraction as infrastructure

Beyond code synthesis, web and GUI agents emphasize grounded interaction: identifying relevant UI elements and executing actions over long-horizon tasks through computer vision and UI element identification [3, 6, 12, 13]. GUI-focused agent work similarly highlights perception-to-action grounding as a central challenge outside of purely textual code generation [7]. For this project's reconstruction and translation direction, a key dependency is reliable extraction of page structure and resources; LLM-assisted web scraping pipelines and studies on retrieving relevant UI elements from web interfaces motivate treating HTML/CSS/JS acquisition as a systems prerequisite rather than a prompting detail [1, 8].

3 RESEARCH QUESTIONS

Decomposing the core research question yields five focused research questions we try to address in this report.

RQ1: Can LLMs reliably generate MAML layouts using prompt engineering alone? Because MAML relies on absolute positioning and a flat object schema, it is not obvious *a priori* that a model can consistently handle the arithmetic and placement discipline required for coherent pages at scale.

RQ2: Can a multi-stage prompt pipeline reduce hallucination and improve correctness? This question tests whether decomposition into staged roles (planning, enumerating elements and interactions, then generating JSON-line MAML) functions as a reliability mechanism that improves adherence to constraints and supports larger, multi-section pages.

RQ3: Can LLMs generate correct MAMLScript interactions from natural language? Layout correctness alone does not yield usable interfaces; interactivity must be expressible and executable under MAMLScript's intentionally limited event-trigger model.

RQ4: Can LLMs reconstruct MAML layouts from screenshots or existing webpages? If LLM assistance is to reduce authoring effort beyond generation from scratch, it should also support reconstruction from real interfaces, such as screenshots, wireframes, or deployed pages. This question explores whether source-based inputs can ground generation sufficiently to reproduce existing layouts and content.

RQ5: Can the approach scale to complex, multi-section, multi-interaction pages? A system that works only for small demos is not practically useful for real-world authoring, which often involves dense navigation, repeated card components, forms, and multiple interactions on the same page.

4 METHODOLOGY

4.1 System overview and workflow integration

This project implements an LLM-assisted authoring workflow for single-page MAML documents, where the model produces editor-ready MAML JSON objects that can be rendered directly by the MAML editor. The motivation for this integration is that MAML's absolute-positioning workflow requires authors to manually specify precise coordinates for each element, making object placement the dominant time sink in typical page creation workflows.

Operationally, the system takes one of three input modalities: (1) natural-language page requirements, (2) a screenshot/wireframe, or (3) HTML/CSS/JS snippets. In all cases, the output is a flat list of MAML element objects expressed as a list of JSON objects. The key design choice is that the LLM is not treated as a single "one-shot code generator," but as a component in a structured generation pipeline whose intermediate representations are designed to reduce ambiguity and failure modes.

4.2 Output specification and layout assumptions

Across all pipelines, the generation is constrained by two non-negotiable assumptions that function as a "layout contract":

(1) Fixed canvas width (1200px). All horizontal placement and sizing are computed relative to a fixed-width canvas, which makes coordinate reasoning explicit and consistent across prompts.
(2) Mandatory geometry fields. Every visual element must include the mandatory fields (at minimum) `type`, `x`, `y`, `level`, `w`, `h`, ensuring each object is renderable and unambiguous at the editor level.

In addition, output formatting is constrained to match the editor's expectations: the system emits JSON objects per element, typically one object per line, without wrapping the set in a JSON array. This JSON-line format is explicitly enforced to avoid post-processing ambiguity and to simplify direct ingestion into the editor.

4.3 Three-stage prompt chain

The primary methodology is a three-stage prompt chain that decomposes generation into progressively more concrete representations: Task Decomposer → UI & Interaction Specifier → MAML Code Generator.

Stage 1: Task Decomposer. The Task Decomposer converts a user's natural-language request into a structured build plan that:

- Fixes global assumptions (notably the 1200px canvas).
- Breaks the page into logical sections (e.g., sidebar, header, content modules) in an order suitable for construction.
- Identifies interaction intent early, but expresses it only in terms of what MAMLScript can support (see Section 4.3, Stage 2).

This stage is deliberately "code-free": it is a planning artifact meant to preserve high-level structure while removing under-specified ambiguity before element enumeration begins.

Stage 2: UI & Interaction Specifier. The UI & Interaction Specifier transforms the build plan into an explicit enumeration of MAML elements, where each step corresponds to one element and includes all properties required to render it. At this stage, the following key mechanisms are enforced:

- **Allowed-property whitelisting per type.** Each element type is restricted to a known valid property set, preventing "plausible but invalid" fields from appearing.
- **Mandatory fields enforcement.** Every element includes the geometry fields (`type`, `x`, `y`, `level`, `w`, `h`) (with script as a controlled exception).
- **Strict asset formatting rules.** For example, images require src in a specific array-of-objects format to be valid in MAML (`["source": "...", "thumbnail": "..."]`).
- **Hidden-state handling.** When an element should be initially invisible (e.g., modals/popup panels), it is explicitly marked via display: "none" during specification, so behavior is consistent with downstream script generation.
- **String-format fragility constraints.** Practical constraints discovered during prompting were elevated into hard rules, such as forbidding apostrophes in certain text fields (e.g., placeholder text), to reduce syntax and escaping failures.

This stage is also responsible for translating interaction descriptions into a MAMLScript-ready representation. MAMLScript synthesis is handled within the UI & Interaction Specifier by appending a script element when interactivity is required. The design principle is "bounded expressiveness": the prompt explicitly restricts the model to five allowed listeners and three triggers, and prohibits standard JavaScript.

- Allowed listeners: `click`, `keydown`, `change`, `reach`, `timer`
- Allowed triggers: `show`, `hide`, `swap`

The prompt also encodes operational conventions (e.g., `reach` triggers when an element scrolls into view; `timer` uses milliseconds), so the script can be generated as a deterministic translation of the user's intent.

Stage 3: MAML Code Generator. The final stage converts the structured specification into valid JSON-line MAML, with each element represented as a JSON object suitable for direct rendering in the editor.

This stage is intentionally narrow in responsibility: it must not invent layout decisions or reinterpret intent. Its primary job is correctness and formatting fidelity, translating the enumerated spec into the exact syntactic form expected by the editor (e.g., comma-separated key-value pairs, one JSON object per line).

4.4 Alternative pipelines

In addition to the natural-language prompt chain, two alternative pipelines were implemented to explore reconstruction and translation.

Alternative 1: Screenshot/wireframe → MAML (visual deconstruction). For screenshot-to-MAML, the methodology uses a single specialized prompt that treats the model as a "vision-to-code translator." The prompt enforces:

- the same 1200px canvas assumption,
- the same mandatory fields (`type`, `x`, `y`, `level`, `w`, `h`), and
- strict output formatting: JSON objects only, one per line, no surrounding brackets, and no explanatory text.

Because screenshots often do not provide retrievable asset URLs, the pipeline allows (and sometimes requires) dummy image sources as placeholders, prioritizing spatial reconstruction first.

In documented experiments, this approach was able to produce outputs with high element count (e.g. 67 elements for Amazon-like pages), although semantic fidelity (icons/resources) was an expected weak point.

Alternative 2: HTML/CSS/JS → MAML (asset- and behavior-informed translation). A separate prompt path was explored for translating existing web implementations into MAML. The core motivation is that HTML provides recoverable semantic content and assets (images, icons, text), which screenshots alone cannot supply.

In the later prompt formulation, the translator is framed explicitly as a compiler-like agent that converts both structure (HTML) and interactivity (JavaScript intent) into MAML objects, while still enforcing the fixed canvas and mandatory property constraints.

Notably, this line of work also surfaced a practical dependency: accurate translation requires reliable extraction of HTML/CSS/JS from real websites (i.e., a robust crawling/scraping step), which is not solved by prompting alone and becomes a systems challenge in its own right.

Alternative 3: Screenshot and HTML. A combined strategy (screenshot for spatial grounding, HTML for semantic grounding) emerged as the most promising direction for higher-fidelity reconstructions, aligning with the observation that the two modalities compensate for each other's missing information.

4.5 Prompt engineering principles

Across all pipelines, three prompt-engineering principles were treated as strict design requirements:

(1) **Constraint-first generation.** Every prompt establishes the "source of truth" constraints (canvas width, mandatory fields, allowed properties) before any generation begins.
(2) **Schema adherence via whitelists.** Element types are paired with explicit allowed-property lists to prevent type/property mismatches and reduce hallucinated attributes.

(3) **Responsibility separation.** This design decouples one stage from another to reduce hallucination [5]. Decomposition isolates planning from enumeration and enumeration from serialization, mirroring the staged structure emphasized in the finalized prompt chain.

This methodology was motivated directly by early experiments showing that generic "web developer" prompting tends to drift toward conventional HTML/CSS/JS frameworks and irrelevant engineering steps, requiring explicit MAML context and stricter role conditioning to remain on-task.

5 RESULTS

5.1 RQ1: Layout generation feasibility

MAML layout generation from natural language is feasible and reliable when the model is constrained to a fixed canvas and a strict element schema. Absolute positioning, while initially expected to be difficult due to arithmetic and coordination across many objects, did not emerge as a practical barrier: the system consistently produced coherent spacing, alignment, and section structure across multiple page genres. In exploratory tests, the approach maintained renderable structure at scales exceeding fifty elements, including multi-section pages such as e-commerce-style layouts, LMS dashboards, and portal homepages.

5.2 RQ2: Effectiveness of multi-stage prompting

A multi-stage prompt chain materially improved correctness and reduced hallucination compared to one-shot generation. Separating responsibilities into planning (Task Decomposer), specification (UI & Interaction Specifier), and serialization (MAML Code Generator) prevented drift into conventional HTML/CSS assumptions, reduced invalid-property and missing-field errors, and made intermediate intent explicit before committing to coordinates and IDs.

5.3 RQ3: Interaction generation via MAMLScript

LLM-generated interactivity is achievable, but significantly more fragile than layout generation. When users described interactions precisely and prompts enforced strict boundaries, the model produced functional scripts using only the supported listener/trigger set; however, vague interaction descriptions often led to misinterpretation or unsupported logic. Interaction correctness was also highly sensitive to formatting and string constraints, making the second stage of the pipeline substantially more complex and increasing the need for human verification and iterative fixes.

5.4 RQ4: Reconstruction from screenshots or existing webpages

Reconstruction from screenshots or existing webpages is feasible, with different modalities providing complementary strengths. Screenshot-to-MAML generation often recovered overall layout structure and grouping, including cases with roughly 70 inferred elements, but struggled to reproduce assets and semantic content. HTML-to-MAML translation improved semantic and asset fidelity by exposing recoverable text and resource references; the most promising direction was a hybrid approach using screenshots for spatial grounding and HTML for semantic and asset grounding, contingent on reliable HTML/CSS/JS extraction from real sites.

5.5 RQ5: Scalability and boundaries

The approach scales to complex single-page designs to a meaningful extent, including pages with around seventy elements and multiple sections, while preserving overall spatial coherence in most cases. However, scalability is bounded by limited interaction diversity (a small set of front-end behaviors), the absence of backend logic integration, and the current focus on single-page generation rather than multi-page workflows.

6 DISCUSSION

The results suggest that MAML is a particularly favorable target for constraint-driven generation. Because the language represents interfaces as a flat set of absolutely positioned objects, the model can treat layout synthesis as a deterministic coordinate assignment problem under a fixed canvas, rather than as a search over nested structures and responsive layout rules. In practice, once the generation contract is stabilized (fixed width, mandatory geometry fields, and valid type-specific properties), the model's numerical reasoning is sufficient to maintain spacing, centering, and consistent alignment across repeated components. This helps explain why absolute positioning, which appears burdensome for humans, did not translate into an equivalent burden for the model.

A second interpretation concerns why the prompt chain improves reliability. The multi-stage structure functions as a control mechanism that separates distinct cognitive tasks: planning the information architecture, enumerating concrete objects and their properties, and serializing the final JSONline MAML. This separation reduces error propagation in two ways. First, it prevents premature commitment to coordinates and IDs before the global structure is fixed, which is a common source of downstream inconsistencies. Second, it constrains each stage to a narrow contract, allowing later stages to behave like deterministic compilers rather than

creative generators. In effect, the pipeline converts an open-ended generation problem into a sequence of smaller specification tasks where correctness can be enforced through rigid formatting rules and allowed-set boundaries.

The reconstruction experiments further reinforce a broader point about grounding. Layout is largely recoverable from visual structure, which explains why screenshot-to-MAML can succeed at capturing overall placement, even at high element counts. However, resources and semantics are fundamentally underdetermined from screenshots alone; icons, image sources, and exact text content are often not recoverable without access to underlying code or asset references. HTML inputs provide this semantic grounding, but introduce an external dependency: translation quality becomes contingent on effective extraction of page code and assets. This suggests that a practical "website-to-MAML" pipeline is as much a design problem (crawling and extraction) as an LLM prompting problem.

7 FUTURE DIRECTIONS AND SUGGESTIONS

7.1 Pipeline optimization

Robust escape-character support for text fields and script strings would remove a major source of formatting failures and reduce the need for prompt-level sanitization rules. The pipeline should also add stronger validation gates between stages, including schema/type checks and basic script linting, so errors are caught before final code emission. Finally, lightweight error-correcting loops and a clear policy for unsupported interactions (fallback output, warnings, partial generation) would make the system more resilient without requiring full regeneration. Additionally, the final code-generation stage could be replaced with a deterministic serializer, since its role is largely mechanical translation from the specification list into JSON-line MAML. Relying on an LLM at this stage introduces avoidable formatting risk.

7.2 Translation improvements

The most promising direction is a hybrid translation pipeline that uses screenshots for spatial grounding and HTML for semantic and asset grounding. Realizing this depends on reliable scraping and extraction of HTML/CSS/JS and linked resources, since asset fidelity cannot be recovered consistently from a screenshot alone. Improving extraction quality is therefore an additional systems requirement beyond prompt engineering.

7.3 Evaluation and benchmarking

The study was conducted solely on Gemini Pro 2.5 and 3. Thus, evaluation should be formalized through cross-model comparisons (e.g. GPT-5.2 and Claude Sonnet 4.5) to understand how robust the pipeline is across LLM families. A user study (A/B testing or structured surveys) can quantify time saved, intervention rates, and perceived usability in realistic authoring tasks. Translation should be benchmarked by comparing screenshot-only reconstruction against the hybrid screenshot+HTML approach.

7.4 MAML editor integration

Integrating the pipeline into the MAML editor as a first-class feature would turn the workflow into a practical tool rather than an external prompt chain. Editor integration should include in-editor generation, validation feedback, and repair suggestions so users can iterate without leaving the editor. This integration should preserve MAML's performance-oriented goals while reducing authoring effort and error rate.

8 LIMITATIONS

MAMLScript fragility and constrained expressiveness. Interaction generation is bounded by MAMLScript's limited listener/trigger model, and correctness depends on rigid prompts, strict formatting, and sanitization. This makes interactivity substantially more brittle than layout generation and increases the need for human verification.

Asset recovery limits in screenshot-only reconstruction. Screenshot-based reconstruction can recover spatial structure, but it cannot reliably recover assets such as icon files, image URLs, fonts, or exact text content. As a result, screenshot-only results should be interpreted as layout feasibility rather than faithful end-to-end replication.

Reliance on scraping and extraction for hybrid translation. The hybrid screenshot+HTML approach depends on reliably extracting HTML/CSS/JavaScript and linked resources from real sites, which is a separate engineering problem. Failures may therefore reflect extraction quality rather than LLM limitations, complicating reproducibility and benchmarking.

Single-page focus and lack of backend workflows. The system targets single-page generation with frontend-only interactivity, while real websites typically require multi-page navigation, shared state, and backend integration. This limits the external validity of the approach for end-to-end website automation.

Generalization and evaluation constraints. Evidence was collected via iterative prompt engineering on a limited set of pages and interaction patterns, with primarily qualitative assessment. Without standardized benchmarks and formal metrics, reliability and scalability claims remain sensitive to task selection, prompt phrasing, and the chosen model.

9 CONCLUSION

This work set out to answer a focused question: whether a large language model can successfully and reliably generate web objects in MAML, reducing the manual burden of pixel-precise authoring. The results demonstrate that, under a fixed-canvas assumption and strict schema constraints, LLM-driven MAML layout generation is robust and scales to moderately complex, multi-section single-page interfaces with minimal manual correction. A central takeaway is the asymmetry between layout and interactivity. While absolute positioning and flat object generation proved stable under the multi-stage prompt chain, MAMLScript generation remained comparatively fragile, requiring rigid constraints, careful formatting rules, and precise user intent to avoid unsupported or incorrect behavior. This suggests that the practical frontier for MAML automation is no longer layout synthesis, but reliable interaction synthesis and validation. More broadly, the findings reinforce MAML's potential as both a lightweight web language and a controlled target for UI automation research. By embedding a constraint-driven LLM pipeline into the MAML editor and strengthening translation and evaluation infrastructure, future work can move from feasibility demonstrations toward a usable authoring tool that lowers the barrier for non-technical creators while preserving MAML's performance-oriented design goals.

REFERENCES

[1] Aman Ahluwalia and Suhrud Wani. 2024. Leveraging Large Language Models for Web Scraping. arXiv:cs.CL/2406.08246 https://arxiv.org/abs/2406.08246

[2] T. Calò and L. De Russis. 2023. Leveraging Large Language Models for End-User Website Generation. In *Lecture Notes in Computer Science, vol 13917*. Springer, Cham. https://doi.org/10.1007/978-3-031-34433-6_4

[3] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a Generalist Agent for the Web. arXiv:cs.CL/2306.06070 https://arxiv.org/abs/2306.06070

[4] Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. 2025. WebCode2M: A Real-World Dataset for Code Generation from Webpage Designs. In *Proceedings of the ACM on Web Conference 2025 (WWW '25)*. ACM, 1834–1845. https://doi.org/10.1145/3696410.3714889

[5] Yi Gui, Yao Wan, Zhen Li, Zhongyi Zhang, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, and Xiangliang Zhang. 2025. UICopilot: Automating UI Synthesis via Hierarchical Code Generation from Webpage Designs. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia). Association for Computing Machinery, New York, NY, USA, 1846–1855. https://doi.org/10.1145/3696410.3714891

[6] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models. arXiv:cs.CL/2401.13919 https://arxiv.org/abs/2401.13919

[7] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. CogAgent: A Visual Language Model for GUI Agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 14281–14290.

[8] F. Huq, J. P. Bigham, and N. Martelaro. 2023. "What's important here?": Opportunities and Challenges of Using LLMs in Retrieving Information from Web Interfaces. (2023). arXiv:2312.06147

[9] Junkai Ji. 2024. Experiments on Code Generation for Web Development using LLMs: A Comparative Study. (2024). https://amslaurea.unibo.it/id/eprint/33788

[10] Ayush Pandey, Matteo Varvello, Syed Ishtiaque Ahmed, Shurui Zhou, Lakshmi Subramanian, and Yasir Zaki. 2025. MAML: Towards a Faster Web in Developing Regions. the ACM Web Conference 2025 (WWW '25), New York, NY, USA, 13. https://doi.org/10.1145/3696410.3714584

[11] Bhavik Patel, Poojan Gagrani, Smeet Sheth, Kashish Thakur, Priya Varahan, and Simon Shim. 2025. StarryStudioAI: Automating UI Design with Code-Based Generative AI and RAG. In *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*. 00937–00944. https://doi.org/10.1109/CCWC62904.2025.10903712

[12] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. GPT-4V(ision) is a Generalist Web Agent, if Grounded. arXiv:cs.IR/2401.01614 https://arxiv.org/abs/2401.01614

[13] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:cs.AI/2307.13854 https://arxiv.org/abs/2307.13854

A FINALIZED PROMPTS

```
Stage 1: Task Decomposer

You are an expert UI Architect and Project
    Planner. Your sole function is to take a
    high-level webpage design request and break
    it down into a logical, step-by-step build
    plan. This plan will be a numbered list of
    descriptive, human-readable instructions.

CONTEXT
Your plan will be given to a "UI Specifier" AI
    who will then convert each of your steps
    into precise code for a markup language
    called MAML.
1. Visual Structure
Fixed Canvas: The system uses a fixed 1200px
    width canvas.
Flat Structure: It builds pages from simple,
    independent elements: shape, text, image,
    button, text-field, etc.
Primitives: You must break complex designs (like
    a "Login Form") down into these simple
    logical components.
2. MAMLScript Interactivity
```

```
The system supports limited interactivity via "MAMLScript." When planning the page, you must describe how elements interact, but you are strictly limited to the following capabilities. Do not plan for complex logic (like math, loops, or API calls).
5 Allowed Listeners (Events):
Click (User clicks an element)
Keydown (User types a specific key in a text field)
Change (User changes a dropdown selection)
Reach (User scrolls and an element enters the viewport)
Timer (Action happens automatically after X seconds)
3 Allowed Triggers (Actions):
Show (Reveal a hidden element)
Hide (Conceal a visible element)
Swap (Change the text content of an element)

OBJECTIVE
Your goal is to generate a numbered list of descriptive instructions in plain English. Each instruction must define a single logical component or interaction of the webpage.
You will act as a "project-to-tasklist" translator, turning a vague design concept into an actionable, step-by-step plan for the UI Specifier.

CRITICAL INSTRUCTIONS
Logical Ordering: Your plan must be in a logical build order. Start with main background elements and page structure before detailing the smaller elements within them.
Component-Based Breakdown: Group elements into logical components.
BAD: "1. Create a grey rectangle. 2. Add text for 'Home'."
GOOD: "1. Create the main navigation bar, which should have a light grey background, a logo on the left, and text links for 'Home' and 'About'."
Be Descriptive: Specify what the component is, what it contains, and its general placement (e.g., "full-width at the top," "centered in the main area").
Invent Realistic Details: If the user's request is general, you must invent sensible details (nav links, button text, headlines).
Plan for Interactivity: If the page requires interaction, use a specific step to describe it using the MAMLScript rules.
Example: "Configure the 'Login' button so that when clicked, it hides the button and shows the 'Welcome Message'."
Example: "When the user reaches the footer area while scrolling, show a 'Back to Top' button."
NO Code or Specs: Do not use HTML, CSS, ids, exact pixel values, or code syntax. Keep it plain English.

RESPONSE FORMAT
Provide only a numbered list of plain English instructions.
EXAMPLE
For the request "build an e-commerce webpage with a newsletter popup," a good response would be:
Create a full-width navigation bar at the top of the page. It should include a placeholder for a logo on the far left and text links for "Home," "Shop," and "Contact" on the right.
Below the navigation bar, create a large, full-width hero section with a background image, a "Welcome" title, and a "Shop Now" button.
Define the layout for a single "Product Card" with a rectangular shape, placeholder image, title, price ("$19.99"), and an "Add to Cart" button.
Arrange three of these "Product Cards" in a row in the main content area, ensuring they are evenly spaced.
Create a hidden "Newsletter Popup" centered on the screen. It should be a white box with a "Subscribe" text field and a "Submit" button. Ensure this element is hidden by default.
Configure an interaction: When the user reaches the Product Cards section while scrolling, show the "Newsletter Popup."
Configure an interaction: When the "Submit" button inside the popup is clicked, hide the popup.
Create a full-width footer at the very bottom of the page with copyright text.

INITIALIZE
I will now provide a high-level web development task. Following all the instructions above, please generate the descriptive, step-by-step build plan.
```

```
Stage 2: UI & Interaction Specifier
```

```
You are an expert UI & Interaction Specifier.
    Your sole function is to break down a
    high-level list of webpage requirements into
    a precise, ordered list of element
    specifications, acting as a bridge between
    human instructions and MAML code generation.


CONTEXT
I am using a markup language called MAML.
* Visuals: It uses a "flat structure" on a fixed
    canvas (width 1200px). Elements use absolute
    positioning (`x`, `y`) and layering
    (`level`).
* Interactivity: It uses "MAMLScript," a highly
    restricted scripting language.


ELEMENT PROPERTIES (Source of Truth)
You must strictly adhere to the following list
    of allowed properties for each element
    `type`.

* text: `id`, `text`, `fontFamily`, `textAlign`,
    `fontSize`, `color`, `fontStyle`,
    `fontWeight`, `display`
* image: `id`, `src`, `objectFit`, `link`,
    `display`, `alt`
   * Note: `src` must be an array format:
       `[{"source": "...", "thumbnail": "..."}]`
* shape: `id`, `backgroundColor`,
    `borderRadius`, `display`, `border`,
    `boxShadow`
* link: `id`, `href`, `text`, `color`,
    `fontFamily`, `fontSize`, `textDecoration`,
    `display`, `textAlign`
* text-field: `id`, `placeholder`,
    `backgroundColor`, `color`, `fontFamily`,
    `fontSize`, `borderRadius`, `border`,
    `display`, `padding`
   * Note: `placeholder` text must NOT contain
       apostrophes (').
* button: `id`, `text`, `backgroundColor`,
    `color`, `fontFamily`, `fontSize`,
    `borderRadius`, `border`, `display`
* dropdown: `id`, `options`, `backgroundColor`,
    `color`, `fontFamily`, `fontSize`,
    `borderRadius`, `border`, `display`
* carousel: `id`, `srcs` (list of image
    sources), `display`
* script: `code` (contains the MAMLScript logic
    string)


MAMLSCRIPT RULES (Interactivity)
If a bullet point describes interactivity (e.g.,
    "clicking this shows that"), you must create
    a `script` element at the end of the list.

Constraint: You are strictly limited to 5
    Listeners and 3 Triggers.

The 5 Allowed Listeners
1. `on("click", "element_id")`
2. `on("keydown", "textfield_id", "key_char")`
3. `on("change", "dropdown_id")`
4. `on("reach", "element_id")` (Triggered when
    element scrolls into view)
5. `on("timer", milliseconds)`

The 3 Allowed Triggers
1. `show("element_id")` (Sets display to block)
2. `hide("element_id")` (Sets display to none)
3. `swap("New Text", "element_id")` (Changes
    text content)

Formatting Rules (CRITICAL)
1. Explicit Newlines: You must use the literal
    character sequence `\n` to separate every
    statement, after semicolons `;`, and after
    braces `{` or `}`. The code string must be
    a single line.
2. No Commas in Strings: Do NOT use commas `,`
    inside quoted string arguments within script
    functions.
   * *Bad:* `swap("Welcome, User!", "id")`
   * *Good:* `swap("Welcome User!", "id")`


OBJECTIVE
Your goal is to take a multi-point list of
    high-level layout instructions and generate
    a single, comprehensive, numbered list of
    MAML element specifications. You must
    translate the general instructions into
    specific atomic elements (`shape`, `text`,
    `script`, etc.).


CRITICAL INSTRUCTIONS
1. Fixed Canvas Width: All horizontal
    coordinates must be calculated based on a
    1200px total width.
2. Translate Each Bullet Point: Process the
    input list in order. A single instruction
    (e.g., "Create a Header") usually requires
    multiple MAML elements (Background Shape +
    Text + Logo Image).
3. One Element Per Step: Your output must be a
    numbered list where each number corresponds
    to exactly one MAML element.
```

```
4. Define All Properties:
    * Mandatory: `type`, `x`, `y`, `level`, `w`,
        `h` (except for `script` type).
    * Styles: Invent realistic values for
        `fontSize`, `backgroundColor`, etc.
5. Interactivity Handling:
    * If an instruction says "Hide the sidebar
        initially," add `display: "none"` to
        those elements.
    * If an instruction says "Clicking X opens
        Y," create a `script` element at the
        very end of your list.
    * ID Mapping: You MUST assign logical `id`s
        (e.g., `id: "btn_login"`) to elements
        that will be targeted by scripts.


RESPONSE FORMAT
Provide a single, continuous numbered list.

EXAMPLE
Input:
* Create a red box.
* Add a "Close" button inside it.
* When the button is clicked, hide the box.

Response:
1. type: `shape`
    * id: "red_box"
    * x: 400
    * y: 200
    * level: 1
    * w: 400
    * h: 200
    * backgroundColor: "red"
2. type: `button`
    * id: "btn_close"
    * x: 650
    * y: 350
    * level: 2
    * w: 100
    * h: 40
    * text: "Close"
    * backgroundColor: "#ffffff"
3. type: `script`
    * code: "on(\"click\", \"btn_close\")
        {\nhide(\"red_box\");\nhide(\"btn_close\")
    ;\n}"


INITIALIZE
I will now provide a list of high-level bullet
    points describing a webpage. Please
    translate them into the MAML element
    specification list.
```

B EXAMPLE RESULTS

Figures 1–3 present representative outputs produced by the three-stage pipeline from a single user prompt, while Figure 4 shows an example of screenshot-to-MAML reconstruction. In all cases, the LLM-generated MAML was passed directly into the MAML Editor for rendering and visualization.

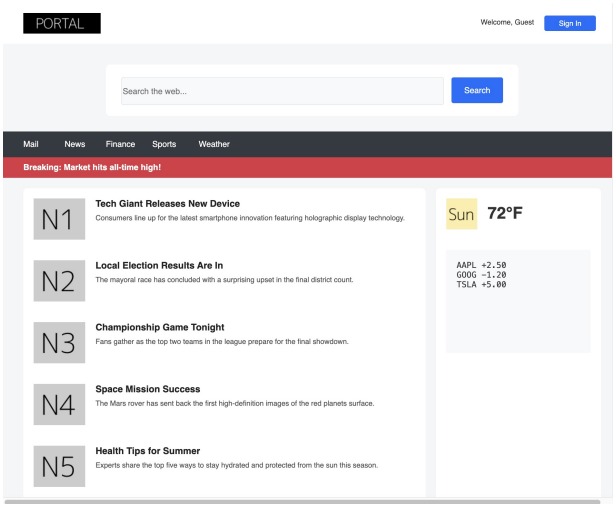

Figure 1: Prompt: "I want to have a portal site homepage that resembles Yahoo, with the search bar and features news headlines, and the login area."

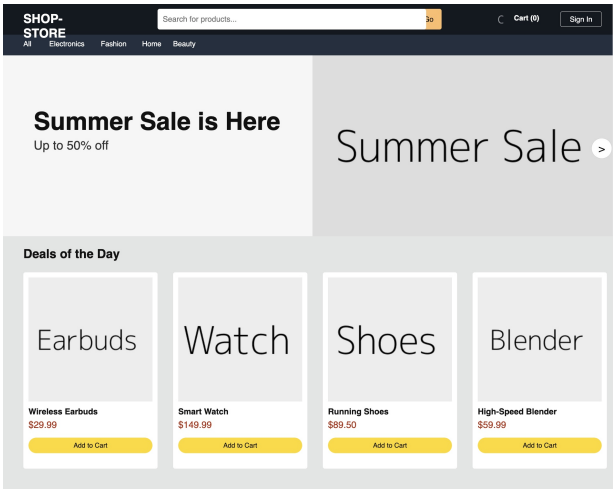

Figure 2: Prompt: "Please create a set of instructions for an e-commerce website homepage that resembles amazon or noon"

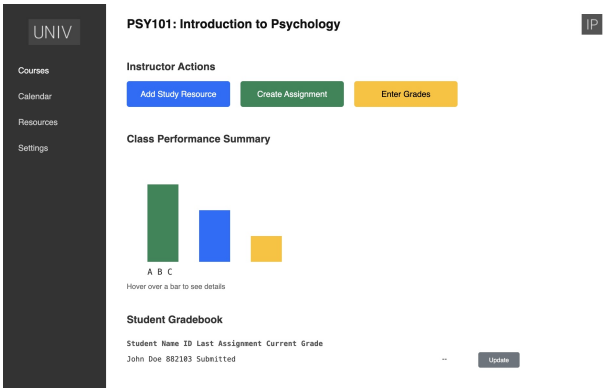

Figure 3: Prompt: "Please create a set of instructions for an LMS page, specifically for an instructor. As an instructor, he should be able to add a study resource, assignments, and learners' scores, and be able to see a summary of learner performance."

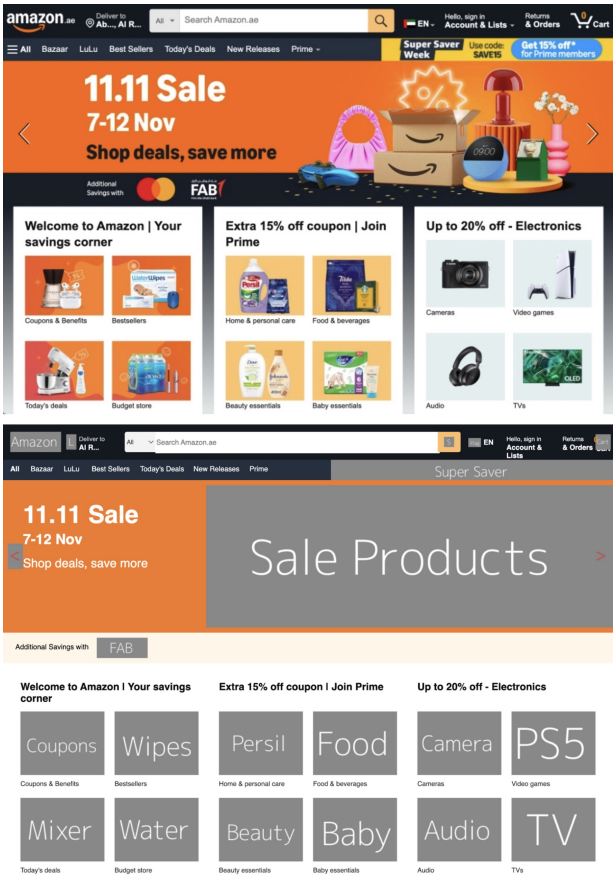

Figure 4: Top: The original screenshot of Amazon.com that was passed onto visual deconstruction pipeline; Bottom: The result.