Multi-Agent Planning with MCP Tools for Web Form Generation

Matheus G. S. Dalmolin[1],
Giovanna A. P. Soares[1], **Lays D. C. da Silva**[1],
Karolayne S. de Azevedo[1], **Marcelo A. C. Fernandes**[1,2]

[1]InovAI Lab, nPITI/IMD, UFRN, 59.078-900, Natal, RN, Brasil

[2]Departamento de Engenharia da Computação e Automação (DCA), UFRN, Natal, Brasil

matheusdalmolinrs@gmail.com, giovanna.assuncao.705@ufrn.edu.br

crisantolays1@gmail.com, karolayneazevsantos@gmail.com

mfernandes@dca.ufrn.br

Abstract. *Large language models can generate web forms from natural language, but direct code generation is hard to audit, reproduce, and repair. We present an approach that converts user requests into an editable plan of typed Model Context Protocol (MCP) tool calls, executed by specialized HTML, CSS, and JavaScript components after human approval. A JSON-schema response forces the HTML planner to enumerate its expected field identifiers before emitting tool calls, an LLM-as-a-Judge critic audits the plan for coverage, and a deterministic normalizer aligns identifiers to the schema. Over three prompts replicated across* $N = 10$ *independent trials per condition, the approach reaches an aggregate score of* 99.1 ± 2.2 *against* 82.1 ± 16.0 *for direct generation and perfect coverage of the requested behaviors (*1.00 ± 0.00 *versus* 0.55 ± 0.15*), at about* $4.8\times$ *the latency. A deterministic plan-execution ablation reaches* 100.0 ± 0.0*, an empirical upper bound for the current tool catalog.*

Resumo. *Modelos de linguagem conseguem gerar formulários web a partir de linguagem natural, mas a geração direta de código tende a ser pouco auditável, pouco reprodutível e difícil de reparar. Apresentamos uma abordagem que transforma solicitações do usuário em um plano editável de chamadas tipadas a ferramentas Model Context Protocol (MCP), executadas por componentes especializados em HTML, CSS e JavaScript após aprovação humana. Uma resposta com esquema JSON obriga o planejador HTML a enumerar os identificadores de campo esperados antes de emitir as chamadas, um nó crítico no estilo LLM-as-a-Judge audita a cobertura do plano e um normalizador determinístico alinha os identificadores ao esquema. Em três prompts replicados em* $N = 10$ *repetições independentes por condição, a abordagem atinge pontuação agregada de* $99{,}1 \pm 2{,}2$ *contra* $82{,}1 \pm 16{,}0$ *da geração direta e cobertura perfeita dos comportamentos solicitados (*$1{,}00 \pm 0{,}00$ *contra* $0{,}55 \pm 0{,}15$*), a cerca de* $4{,}8\times$ *a latência. Uma ablação de execução determinística do plano atinge* $100{,}0 \pm 0{,}0$*, limite superior empírico do catálogo atual.*

1. Introdução

Modelos de linguagem de larga escala (LLMs) ampliaram a capacidade de gerar código a partir de descrições em linguagem natural [Brown et al. 2020, Chen et al. 2021]. Em interfaces de formulários, essa capacidade é atraente porque usuários podem descrever campos, validações, máscaras e regras condicionais sem escrever HTML, JavaScript ou CSS manualmente. No entanto, a geração direta de código tende a produzir saídas sensíveis ao prompt, difíceis de auditar e inconsistentes entre execuções. Em contextos empresariais, pequenas inconsistências entre campos, identificadores, eventos e regras de validação podem tornar o artefato gerado incorreto ou custoso de reparar.

Este trabalho investiga uma alternativa: representar a solução como um plano intermediário, composto por chamadas tipadas a ferramentas determinísticas. Na abordagem proposta, o LLM não escreve diretamente o artefato final; ele atua como planejador e roteador em uma arquitetura multiagente, e o plano resultante pode ser exibido, validado, editado e executado de forma incremental por ferramentas Model Context Protocol (MCP) [Anthropic 2024]. A hipótese central é que separar o planejamento probabilístico da execução determinística aumenta o controle sobre a geração: torna possível avaliar etapas intermediárias, reutilizar templates, detectar inconsistências antes da escrita final e aplicar reparos sem regenerar todo o formulário.

O objetivo deste artigo é projetar e avaliar empiricamente essa arquitetura. Para isso, contribuímos com uma arquitetura multiagente de geração de formulários baseada em um plano intermediário editável de chamadas tipadas; com um catálogo MCP de ferramentas determinísticas para HTML, CSS, JavaScript e edição de plano; com um mecanismo de controle humano e reparo incremental por operações sobre o plano; e com um protocolo experimental que compara a geração estruturada a linhas de base de geração direta e a uma ablação determinística, sob replicação estatística. Antecipando os resultados, em três prompts replicados em dez repetições independentes por condição, a abordagem proposta eleva a cobertura dos comportamentos solicitados de $0{,}55 \pm 0{,}15$, obtida pela geração direta, para $1{,}00 \pm 0{,}00$, e a pontuação agregada de $82{,}1$ para $99{,}1$, ao custo de cerca de $4{,}8\times$ em latência.

2. Trabalhos Relacionados

A geração de código a partir de linguagem natural evoluiu da síntese de programas [Chen et al. 2021, Zan et al. 2023] para a geração automatizada de interfaces. Trabalhos de front-end como Design2Code e DesignBench [Si et al. 2024, Xiao et al. 2025] produzem o artefato final em uma única passagem e medem fidelidade visual; mais próximo do nosso domínio, a geração de formulários eletrônicos por modelagem explícita [Tekinerdogan and Aktekin 2020] já separa especificação de execução, mas exige que o usuário construa modelos formais. A abordagem proposta interpõe um plano tipado e auditável entre a solicitação em texto livre e o artefato, e avalia cobertura de campos e comportamentos em vez de aparência.

A proposta também se apoia em LLMs aumentados por ferramentas [Schick et al. 2023, Mialon et al. 2023]. Sistemas como Gorilla e ToolLLM [Patil et al. 2024, Qin et al. 2024] treinam o modelo para selecionar e invocar autonomamente grandes catálogos de APIs, gerando a chamada como artefato final. Seguimos a direção oposta: restringimos intencionalmente o modelo a um catálogo

pequeno e tipado de ferramentas de domínio e inserimos aprovação humana antes da execução, trocando cobertura autônoma por controle. Sobre esse contrato atua o MCP [Anthropic 2024], hoje também objeto de estudo acadêmico quanto a ecossistema e segurança [Hou et al. 2025].

Separar uma fase de planejamento probabilístico de uma fase de execução não é, por si só, uma contribuição deste trabalho: é canônico no paradigma *plan-and-execute* [Wang et al. 2023, Xu et al. 2023]. Herdamos essa estrutura, mas diferimos no que é planejado e em como o plano é governado — uma lista de chamadas `ToolCall` tipadas, exibida para aprovação e edição humana antes de uma execução MCP determinística, em vez do encadeamento passo a passo do ReAct [Yao et al. 2023]. A arquitetura é multiagente, mas, diferentemente do AutoGen [Wu et al. 2023], em que agentes resolvem tarefas por conversação livre, e do MetaGPT [Hong et al. 2024], que codifica procedimentos em papéis fixos para gerar código, empregamos um supervisor determinístico que roteia agentes especializados por artefato e materializa um plano intermediário inspecionável.

Os trabalhos mais próximos combinam representação intermediária editável, aprovação humana e geração de interfaces. O AIPOM [Kim et al. 2025] expõe o plano de um sistema multiagente como grafo editável, mas planeja a orquestração de tarefas, não chamadas que materializam um artefato; o Athena [Beason et al. 2025] escalona a geração de aplicativos por representações intermediárias inspecionáveis, porém com um único modelo e sem execução por ferramentas tipadas. A Tabela 1 situa essas abordagens. Por fim, dois mecanismos internos conectam-se a linhas estabelecidas: a saída estruturada que enumera `expected_fields` é uma forma de geração restrita por esquema [Geng et al. 2025], e o nó crítico de cobertura segue o padrão LLM-as-a-Judge [Zheng et al. 2023] e a tradição de auto-refinamento [Madaan et al. 2023]; como crítico e planejador compartilham a mesma família de modelos, há risco de viés de autopreferência [Panickssery et al. 2024], razão pela qual a aprovação final é externalizada ao humano. Nenhum ingrediente é isoladamente novo; a contribuição está na combinação — usar chamadas MCP tipadas como a própria representação intermediária aprovável e editável, sob um supervisor multiagente com auditoria de cobertura — aplicada à geração controlada de formulários web.

Tabela 1. Posicionamento da abordagem proposta ante trabalhos representativos.

Abordagem	P.I.	Ed./Apr.	M.A.	Exec.	Domínio
ReAct [Yao et al. 2023]	–	–	–	–	geral
Plan-and-execute [Wang et al. 2023, Xu et al. 2023]	✓	–	–	∼	geral
AutoGen/MetaGPT [Wu et al. 2023, Hong et al. 2024]	∼	∼	✓	–	software
Gorilla/ToolLLM [Patil et al. 2024, Qin et al. 2024]	–	–	–	∼	APIs
Athena [Beason et al. 2025]	✓	✓	–	–	apps
AIPOM [Kim et al. 2025]	✓	✓	✓	–	orquestração
Design2Code [Si et al. 2024]	–	–	–	–	UI visual
Nossa abordagem	✓	✓	✓	✓	formulários web

P.I.: plano intermediário inspecionável; Ed./Apr.: edição e aprovação humana do plano; M.A.: multiagente; Exec.: execução determinística por ferramentas tipadas. ✓ presente, ∼ parcial, – ausente.

3. A Proposta

A proposta expõe uma API de orquestração e um servidor MCP de ferramentas de geração. A API é implementada como serviço web e o grafo de orquestração usa bibliotecas como FastAPI e LangGraph [FastAPI 2026, LangChain 2026]. O usuário descreve um formulário, escolhe provedor e modelo, e envia a solicitação ao backend; uma interface web fina expede o pedido e exibe o plano antes da execução, mas não participa do processo de planejamento. A Figura 1 resume essa arquitetura.[1] À esquerda, um supervisor baseado em LLM roteia o pedido a agentes especializados em HTML, CSS e JavaScript, que compõem uma lista única de chamadas `ToolCall` — o plano auditável; esse plano passa por revisão humana (aprovar, refinar ou editar) e, só então, é materializado pela execução determinística das ferramentas MCP de HTML, CSS e JavaScript. À direita, o formulário renderizado correspondente. Cada chamada contém o nome da ferramenta e seus parâmetros estruturados.

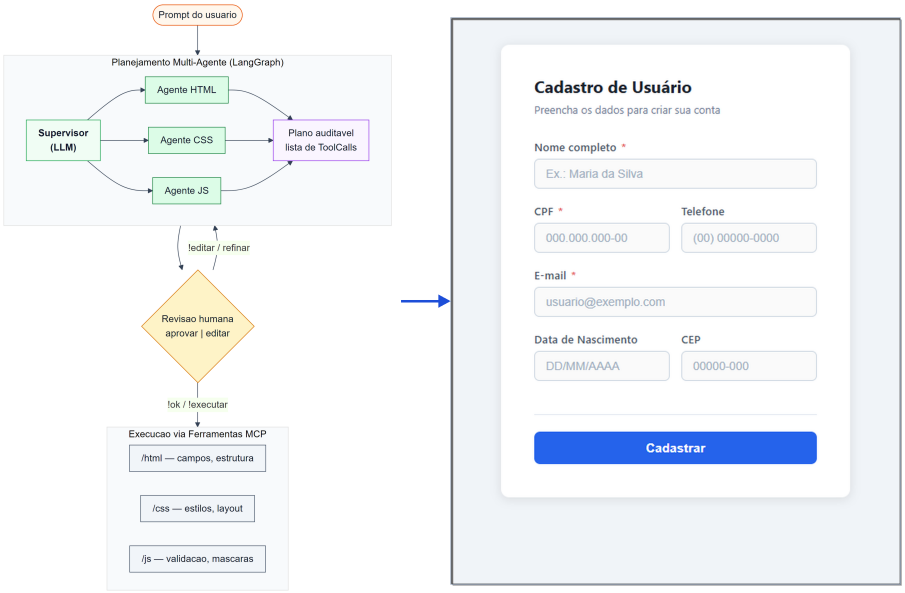


Figura 1. Arquitetura proposta: agentes especializados produzem um plano de `ToolCall`, revisado pelo usuário antes da execução pelas ferramentas MCP (formulário resultante à dir.).

O fluxo de geração possui cinco etapas principais. Primeiro, o backend seleciona o provedor de LLM e carrega as ferramentas MCP disponíveis. Segundo, o supervisor decide qual agente especializado deve contribuir para o plano. Terceiro, os agentes retornam objetos JSON com chamadas a ferramentas; o agente HTML, em particular, usa um esquema JSON que separa a lista de campos esperados das chamadas, detalhado adiante. Quarto, um nó crítico audita a cobertura do plano e completa omissões antes da aprovação humana. Quinto, após aprovação ou comando de execução, as ferramentas MCP são chamadas sequencialmente para produzir os artefatos finais.

O agente HTML responde sob um esquema JSON em duas partes [Geng et al. 2025]: primeiro enumera a lista de campos esperados — cada um com iden-

[1] As Figuras 1 e 2 foram geradas com auxílio de ferramentas de inteligência artificial generativa.

tificador, rótulo, tipo e obrigatoriedade — que o modelo deve declarar antes de emitir o plano; em seguida, produz a sequência de chamadas que materializa esses campos. A lista de campos esperados serve de referência explícita para os identificadores usados nas chamadas. Um extrator separado roda antes do agente HTML e injeta no prompt uma lista de identificadores canônicos previamente derivados da solicitação, reduzindo a frequência com que o modelo renomeia campos durante o planejamento. Para os casos em que o modelo ainda divirja, um normalizador determinístico de pós-processamento compara os identificadores do plano com os campos esperados e reescreve aqueles que diferem apenas por conectores em português (*de*, *da*, *do*) ou sufixos redundantes; a regra é conservadora e só reescreve quando exatamente um campo do esquema casa com o identificador original.

Entre os agentes de planejamento e a aprovação humana opera um nó crítico no estilo LLM-as-a-Judge [Zheng et al. 2023]. O crítico recebe o prompt original, o plano corrente e a lista de campos esperados, e deve listar chamadas ausentes que cubram itens enumerados ou condicionais do prompt não representados pelo plano. Sua saída é restrita a adições; remoções ficam sob responsabilidade do agente editor. O crítico complementa a estratégia de auto-refinamento iterativo em modelos de linguagem [Madaan et al. 2023, Shinn et al. 2023, Chen et al. 2023], mas mantém o processo limitado a uma rodada por ciclo, evitando custos de revisão recorrentes.

O plano intermediário também permite edição incremental. Quando o usuário solicita uma alteração, o estado associado ao identificador do formulário é recuperado, o agente de edição escolhe operações do MCP de plano, e o JSON é modificado em memória. Assim, pequenas correções podem ser aplicadas sem descartar todo o planejamento anterior.

As ferramentas MCP sobre as quais o planejador opera organizam-se em quatro domínios: HTML, JavaScript, CSS e Plan. As ferramentas encapsulam templates, validações de parâmetros e operações de escrita ou edição. A Tabela 2 resume o catálogo atual; a Tabela 3 reproduz os três prompts em linguagem natural que compõem o corpus de avaliação (Seção 4); e a Figura 2 mostra o plano de `ToolCall` gerado para um formulário de cadastro análogo.

Tabela 2. Catálogo de ferramentas MCP da abordagem proposta.

Domínio	Ferramentas	Responsabilidade
HTML	16	Estrutura base, painéis, campos, grupos, seletores e elementos de formulário.
JavaScript	17	Validações, máscaras, eventos, condições, envio de arquivos e cálculos.
CSS	4	Estilo base, responsividade e ajustes visuais do formulário.
Plan	18	Inserção, remoção, reordenação, correção e validação de chamadas.
Total	55	Catálogo completo usado pelo planejador e pelo editor de plano.

A validação ocorre em dois níveis. No nível do plano, são verificados no-

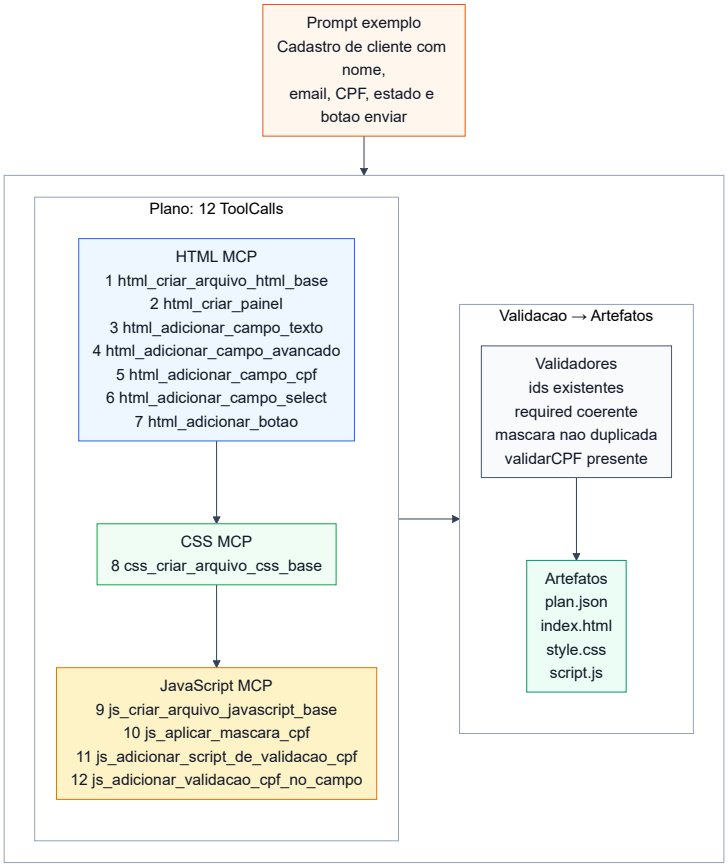


Figura 2. Plano MCP gerado para um formulário de cadastro: sequência de chamadas `ToolCall` (numeradas 1–12) nos domínios HTML, CSS e JavaScript.

Tabela 3. Prompts completos do corpus de avaliação (complexidade crescente). Cada prompt declara os campos e comportamentos esperados, usados na verificação automática de cobertura.

Prompt	Texto enviado ao modelo
`cadastro_contato`	Crie um formulário web de cadastro de contato em português. Deve conter nome completo, e-mail, CPF e telefone brasileiro. Nome, e-mail e CPF são obrigatórios. Aplique máscara e validação de CPF, máscara de telefone brasileiro, layout responsivo e botão de envio.
`inscricao_evento`	Crie um formulário web de inscrição em evento. Deve conter nome completo, CPF, e-mail, categoria de inscrição com as opções Estudante, Profissional e Comunidade, caixa de seleção para informar se precisa de acessibilidade e um campo de descrição de necessidade que aparece somente quando a caixa estiver marcada. Inclua validação de CPF, máscara de CPF, layout responsivo e botão de envio.
`reembolso_viagem`	Crie um formulário web de solicitação de reembolso de viagem. Deve conter funcionário, data da viagem, origem, destino, valor de passagem, valor de hospedagem, total de reembolso calculado automaticamente como soma dos valores e envio de comprovante. Campos principais devem ser obrigatórios, deve haver máscara de data, layout responsivo e botão de envio.

mes de ferramentas, parâmetros obrigatórios, consistência de diretórios e referências a campos. No nível dos artefatos, são avaliados IDs duplicados, campos obrigatórios, referências JavaScript a elementos existentes, eventos duplicados e presença das validações solicitadas no prompt, considerando os contratos das plataformas web [World Wide Web Consortium 2017, Ecma International 2025].

4. Metodologia Experimental

A avaliação foi organizada em torno de três perguntas de pesquisa, respondidas pelos resultados reportados na Seção 5. A primeira (RQ1) pergunta se a geração por plano e ferramentas MCP aumenta a cobertura de campos e comportamentos e a validade estrutural dos artefatos em relação à geração direta. A segunda (RQ2) examina a que custo de latência e com qual ganho de auditabilidade essa indireção opera. A terceira (RQ3) investiga se o catálogo MCP atual é suficiente para os formulários avaliados e onde se concentra a margem de melhora restante.

O corpus desta avaliação cobre três prompts de formulários web em português com complexidade crescente: (i) cadastro de contato com nome, e-mail, CPF e telefone, máscaras e validação de CPF; (ii) inscrição em evento com categoria, caixa de seleção e campo condicional de acessibilidade; e (iii) solicitação de reembolso de viagem com envio de comprovante e cálculo automático do total. Cada prompt declara seus campos e comportamentos esperados, usados para a verificação automática de cobertura.

Foram comparadas três condições sobre o mesmo conjunto de prompts: (i) `direct_llm`, em que o LLM emite HTML, CSS e JavaScript em uma única resposta JSON; (ii) `formgen_mcp`, com o pipeline proposto completo (supervisor, agentes especializados, planejamento, aprovação explícita e execução via ferramentas MCP); e (iii) `plan_no_edit`, ablação que executa um plano determinístico escrito previamente, sem etapa de planejamento por LLM nem edição. O LLM utilizado em (i) e (ii) foi um modelo da família Gemini Flash Lite acessado via OpenRouter [OpenRouter 2026], com o

mesmo identificador e parâmetros nas duas condições.

Para capturar a variação estocástica do provedor de LLM e separá-la do sinal de método, cada par (condição, prompt) foi executado em $N = 10$ repetições independentes. Para o pipeline proposto, cada repetição usa um `form_id` distinto, de modo que o estado persistido pelo LangGraph não seja reaproveitado entre execuções. As trinta execuções resultantes por condição dão a base para as estatísticas em média $\pm$ desvio padrão reportadas a seguir.

As métricas reportadas são automáticas (não houve avaliação humana) e cobrem: conclusão sem falhas de ferramenta, cobertura de campos, cobertura de comportamentos solicitados, validade estrutural de HTML e JavaScript (ausência de IDs e eventos duplicados e referências JavaScript a elementos existentes), latência e número de ferramentas MCP por formulário. A *pontuação* agregada por execução é 100 vezes a média ponderada de quatro componentes, cada uma em $[0, 1]$: $0{,}45$ para cobertura de campos, $0{,}35$ para cobertura de comportamentos, $0{,}10$ para estrutura e $0{,}10$ para artefatos.

5. Resultados

A Tabela 4 resume o desempenho das três condições sobre o conjunto de três prompts, agregando $N = 10$ repetições independentes por (condição, prompt), o que totaliza trinta execuções por condição. O `formgen_mcp` concluiu todas as trinta execuções com sucesso; o `direct_llm` concluiu 29 das 30, com uma falha de análise do JSON em `inscricao_evento` atribuível à saída do provedor. A diferença mais robusta está na cobertura de comportamentos: o `direct_llm` ficou em $0{,}55 \pm 0{,}15$ enquanto o `formgen_mcp` saturou em $1{,}00 \pm 0{,}00$, traduzindo cada máscara, validação e condicional em uma chamada MCP tipada; na prática, cobertura de $0{,}55$ significa que, em média, quase metade das regras solicitadas fica ausente do formulário gerado, tornando-o incorreto para uso, e não apenas esteticamente inferior. A cobertura de campos da abordagem proposta ($0{,}98 \pm 0{,}05$) ficou próxima do teto e acima da cobertura observada para o LLM direto ($0{,}97 \pm 0{,}18$, com desvio dominado pela falha mencionada). A pontuação agregada favorece a abordagem proposta ($99{,}1 \pm 2{,}2$ contra $82{,}1 \pm 16{,}0$), e o `plan_no_edit` atingiu $100{,}0 \pm 0{,}0$, funcionando como limite superior empírico do catálogo MCP atual. O intervalo de variação da abordagem proposta não sobrepõe o da linha de base direta.

Tabela 4. Comparação das três condições de geração (média $\pm$ desvio padrão em trinta execuções por condição).

Condição	Sucesso	Campos	Comport.	Pontuação	Lat. (s)	Ferram.
direct_llm	29/30	$0{,}97 \pm 0{,}18$	$0{,}55 \pm 0{,}15$	$82{,}1 \pm 16{,}0$	$4{,}24 \pm 2{,}02$	$0{,}0 \pm 0{,}0$
formgen_mcp	30/30	$0{,}98 \pm 0{,}05$	$1{,}00 \pm 0{,}00$	$99{,}1 \pm 2{,}2$	$20{,}37 \pm 4{,}19$	$17{,}2 \pm 2{,}2$
plan_no_edit	30/30	$1{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$	$100{,}0 \pm 0{,}0$	$0{,}26 \pm 0{,}31$	$16{,}3 \pm 0{,}5$

A Tabela 5 apresenta o detalhe agregado por prompt sobre as dez repetições. Os três prompts atingem cobertura plena de comportamentos para `formgen_mcp` e `plan_no_edit`; o `direct_llm` fica abaixo em todos eles. O prompt `inscricao_evento` concentra a maior parte da variação residual da abordagem proposta na pontuação: em quatro das dez repetições, a sequência de ferramentas selecionada pelo planejador omitiu a caixa de seleção `necessita_acessibilidade` – que condiciona a exibição do campo `descricao_necessidade` – apesar de listá-la em

`expected_fields`, produzindo pontuação $93{,}6$ nesses casos. Esse mesmo prompt foi o único em que o `direct_llm` falhou por erro de análise do JSON, refletindo a sensibilidade da linha de base não estruturada a prompts com múltiplas categorias e condicionais.

Tabela 5. Detalhe por prompt e condição, agregando $N = 10$ repetições (média $\pm$ desvio). Campos e Comport. variam em $[0, 1]$; pontuação em $[0, 100]$.

Prompt	Condição	Status	Campos	Comport.	Pontuação
cadastro_contato	direct_llm	10/10	$1{,}00 \pm 0{,}00$	$0{,}50 \pm 0{,}00$	$82{,}5 \pm 0{,}0$
cadastro_contato	formgen_mcp	10/10	$1{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$	$100{,}0 \pm 0{,}0$
cadastro_contato	plan_no_edit	10/10	$1{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$	$100{,}0 \pm 0{,}0$
inscricao_evento	direct_llm	9/10	$0{,}90 \pm 0{,}32$	$0{,}65 \pm 0{,}24$	$81{,}2 \pm 28{,}7$
inscricao_evento	formgen_mcp	10/10	$0{,}94 \pm 0{,}07$	$1{,}00 \pm 0{,}00$	$97{,}4 \pm 3{,}3$
inscricao_evento	plan_no_edit	10/10	$1{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$	$100{,}0 \pm 0{,}0$
reembolso_viagem	direct_llm	10/10	$1{,}00 \pm 0{,}00$	$0{,}50 \pm 0{,}00$	$82{,}5 \pm 0{,}0$
reembolso_viagem	formgen_mcp	10/10	$1{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$	$100{,}0 \pm 0{,}0$
reembolso_viagem	plan_no_edit	10/10	$1{,}00 \pm 0{,}00$	$1{,}00 \pm 0{,}00$	$100{,}0 \pm 0{,}0$

Quanto ao custo operacional, o `formgen_mcp` consome em média $20{,}4 \pm 4{,}2$ s por prompt contra $4{,}2 \pm 2{,}0$ s do `direct_llm`, refletindo o sobrecusto inerente ao planejamento, à extração de identificadores canônicos, à auditoria do plano e à execução sequencial de cerca de 17 ferramentas MCP por formulário. A ablação `plan_no_edit` fica em $0{,}26$ s porque o LLM é ignorado e apenas o motor determinístico das ferramentas materializa os artefatos. Em relação à auditabilidade, todas as execuções de `formgen_mcp` produziram um plano inspecionável antes da execução, comportamento que não existe na condição `direct_llm` por construção.

6. Discussão

Nos três prompts avaliados e com um único modelo, os resultados sustentam a tese de que a geração por plano e ferramentas MCP é mais controlável que a geração direta de código. A cobertura de comportamentos solicitados saturou em $1{,}00 \pm 0{,}00$ para o `formgen_mcp` contra $0{,}55 \pm 0{,}15$ para o `direct_llm`: cada máscara, validação ou condicional do prompt foi traduzida em uma chamada MCP tipada, dificultando o esquecimento silencioso que ocorre quando um único bloco textual precisa carregar HTML, CSS e JavaScript ao mesmo tempo. O intervalo de variação da abordagem proposta ($\pm 2{,}2$ na pontuação) não sobrepõe o da linha de base direta ($\pm 16{,}0$), o que indica que a separação entre planejamento probabilístico e execução determinística domina a flutuação estocástica do provedor. Todas as execuções da abordagem proposta produziram um plano intermediário inspecionável, oferecendo um artefato auditável que a linha de base direta não expõe por construção.

A principal vantagem qualitativa da abordagem proposta é tornar a geração observável. Em vez de confiar em uma resposta textual completa, o sistema apresenta um plano que pode ser examinado antes da execução. Essa propriedade também favorece reprodutibilidade, pois a lista de ferramentas e parâmetros pode ser armazenada, versionada e reexecutada. A separação de responsabilidades entre agentes especializados e ferramentas determinísticas reduz a probabilidade de misturar regras de apresentação, estrutura e comportamento em um único bloco de código gerado, e o slot `expected_fields` no esquema de planejamento, combinado ao nó crítico e ao normalizador de identificadores, ancora cada chamada à lista de campos declarada pelo próprio modelo antes da execução.

Os resultados também explicitam as limitações. A abordagem proposta pagou sobrecusto de cerca de $4{,}8\times$ em latência por prompt em relação à linha de base direta, porque executa supervisor, agentes, extração de identificadores canônicos, auditoria do plano e execução sequencial de múltiplas ferramentas. O catálogo MCP delimita o que pode ser gerado: o caso `inscricao_evento` mostra que campos combinados a múltiplas categorias ainda podem escapar do planejamento, mesmo quando o crítico audita o plano − em quatro das dez repetições desse prompt, o planejador omite a caixa de seleção `necessita_acessibilidade` da sequência de chamadas apesar de incluí-la em `expected_fields`. Esse caso responde pela variação residual da pontuação da abordagem proposta. Por fim, a ablação `plan_no_edit` indica que o teto empírico do catálogo atual é alto (pontuação $100{,}0 \pm 0{,}0$), e a diferença remanescente de $0{,}9$ ponto entre `formgen_mcp` e `plan_no_edit` sugere que a maior parte da margem de melhora está em consistência do planejamento por LLM em prompts com condicionais, não nas ferramentas em si.

Estes resultados devem ser lidos dentro do escopo avaliado. O corpus tem três prompts, em um único idioma e em um único domínio coberto pelo catálogo MCP, e ambas as condições com LLM usam um único modelo e provedor; não há, portanto, garantia de generalização para outros modelos, idiomas ou formulários fora do catálogo. Como o catálogo MCP e o corpus foram definidos pelos mesmos autores, a cobertura plena observada deve ser interpretada como o teto do catálogo — e não como capacidade geral de gerar formulários arbitrários —, um risco de *teaching to the test* que a ablação `plan_no_edit` ($100{,}0 \pm 0{,}0$) torna explícito. As métricas de cobertura são aproximações automáticas baseadas em regras sobre os identificadores e termos presentes nos artefatos, sem avaliação humana, e podem não capturar aspectos semânticos ou de acessibilidade; como a saída da abordagem proposta é, por construção, alinhada à metodologia de verificação, existe ainda o risco de viés de medição favorável a ela. Por fim, as comparações não incluíram testes de significância estatística nem a mensuração de tokens ou custo financeiro por formulário: a latência de $4{,}8\times$ subestima o custo total, pois o pipeline invoca supervisor, três agentes especializados, extrator e crítico, resultando em mais chamadas e tokens que a geração direta.

7. Conclusão

Este trabalho tratou a geração de formulários web como a produção de um plano de chamadas tipadas, e não de código final, separando o que é probabilístico — o planejamento por LLM — do que pode ser determinístico — a execução das ferramentas. É essa separação, e não um componente isolado, que explica os ganhos: as ferramentas MCP dão a execução determinística e o plano inspecionável, enquanto a arquitetura multiagente dá a separação de responsabilidades e o ponto de auditoria de cobertura antes da aprovação humana. No conjunto avaliado — três prompts e um único modelo, replicados em $N = 10$ repetições por condição —, a abordagem foi mais controlável que a geração direta: pontuação agregada de $99{,}1 \pm 2{,}2$ contra $82{,}1 \pm 16{,}0$ e cobertura de comportamentos de $1{,}00 \pm 0{,}00$ contra $0{,}55 \pm 0{,}15$, ao custo de cerca de $4{,}8\times$ em latência. A queda do desvio de $\pm 16{,}0$ para $\pm 2{,}2$ e a ablação determinística em $100{,}0 \pm 0{,}0$ indicam que, fixada a execução, a variância restante é do planejamento: o caso `inscricao_evento` mostra que campos condicionais ainda escapam ao plano mesmo sob o nó crítico. Como trabalho futuro, pretendemos expandir o corpus para casos fora do catálogo e múltiplos modelos e

domínios, reportar testes de significância estatística e o custo de tokens por formulário, e reforçar o crítico para validar a identidade entre os campos esperados e os efetivamente presentes nas chamadas.

Referências

Anthropic (2024). Model context protocol. https://modelcontextprotocol.io/. Accessed: 2026-05-25.

Beason, J., Cheng, R., Schoop, E., and Nichols, J. (2025). Athena: Intermediate representations for iterative scaffolded app generation with an LLM. *arXiv preprint arXiv:2508.20263*.

Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chen, X., Lin, M., Schärli, N., and Zhou, D. (2023). Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Ecma International (2025). ECMAScript language specification. https://tc39.es/ecma262/.

FastAPI (2026). Fastapi documentation. https://fastapi.tiangolo.com/. Accessed: 2026-05-25.

Geng, S., Cooper, H., Moskal, M., Jenkins, S., Berman, J., Ranchin, N., West, R., Horvitz, E., and Nori, H. (2025). JSONSchemaBench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*.

Hong, S., Zhuge, M., Chen, J., et al. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the International Conference on Learning Representations*.

Hou, X., Zhao, Y., Wang, S., and Wang, H. (2025). Model context protocol (MCP): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

Kim, H., Mitra, K., Shen, C., Zhang, D., and Hruschka, E. (2025). AIPOM: Agent-aware interactive planning for multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

LangChain (2026). Langgraph documentation. https://langchain-ai.github.io/langgraph/. Accessed: 2026-05-25.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Welleck, S., Majumder, B. P., Gupta, S., Yazdanbakhsh, A., and Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., Grave, E., LeCun, Y., and Scialom, T. (2023). Augmented language models: a survey. *Transactions on Machine Learning Research*.

OpenRouter (2026). Openrouter api reference. https://openrouter.ai/docs/api-reference/chat-completion. Accessed: 2026-05-25.

Panickssery, A., Bowman, S. R., and Feng, S. (2024). LLM evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems*.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. (2024). Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*.

Qin, Y., Liang, S., Ye, Y., et al. (2024). ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the International Conference on Learning Representations*.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Si, C., Zhang, Y., Li, R., Yang, Z., Liu, R., and Yang, D. (2024). Design2Code: Benchmarking multimodal code generation for automated front-end engineering. *arXiv preprint arXiv:2403.03163*.

Tekinerdogan, B. and Aktekin, N. (2020). Model-based user interface design for generating e-forms in the context of an e-government project. *arXiv preprint arXiv:2003.08172*.

Wang, L., Xu, W., Lan, Y., et al. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

World Wide Web Consortium (2017). HTML 5.2. `https://www.w3.org/TR/html52/`.

Wu, Q., Bansal, G., Zhang, J., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Xiao, J., Wang, M., Lam, M. H., Wan, Y., Liu, J., Huo, Y., and Lyu, M. R. (2025). DesignBench: A comprehensive benchmark for MLLM-based front-end code generation. *arXiv preprint arXiv:2506.06251*.

Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., and Xu, D. (2023). ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations*.

Zan, D., Chen, B., Zhang, F., Lu, D., Wu, B., Guan, B., Yongji, W., and Lou, J.-G. (2023). Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7443–7464.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. (2023). Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*.