Multi-Platform Agentic Apps: Powering Flutter with Autonomous AI



Roman Jaquez
Flutter GDE / GDG Lawrence Organizer
Google Certified Cloud Architect

DevFest 2026 Google Developer Groups


duyupr

duuprmojis.duupr.io

Let's make it
Interactive!
Show your emotions via

DuuprMojis

or go to
mojis.duupr.io

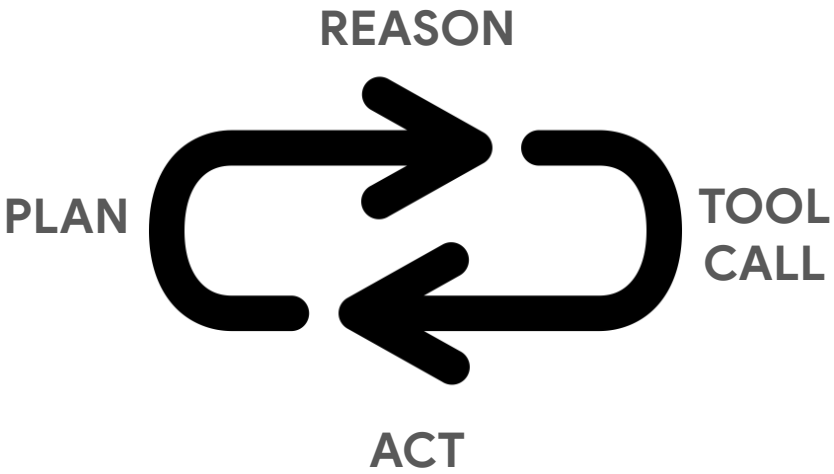
Enter code **TGKYDE**

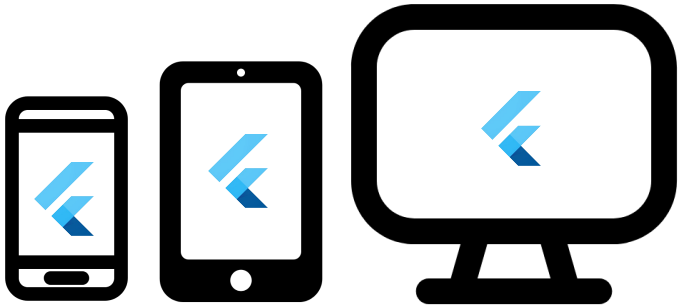


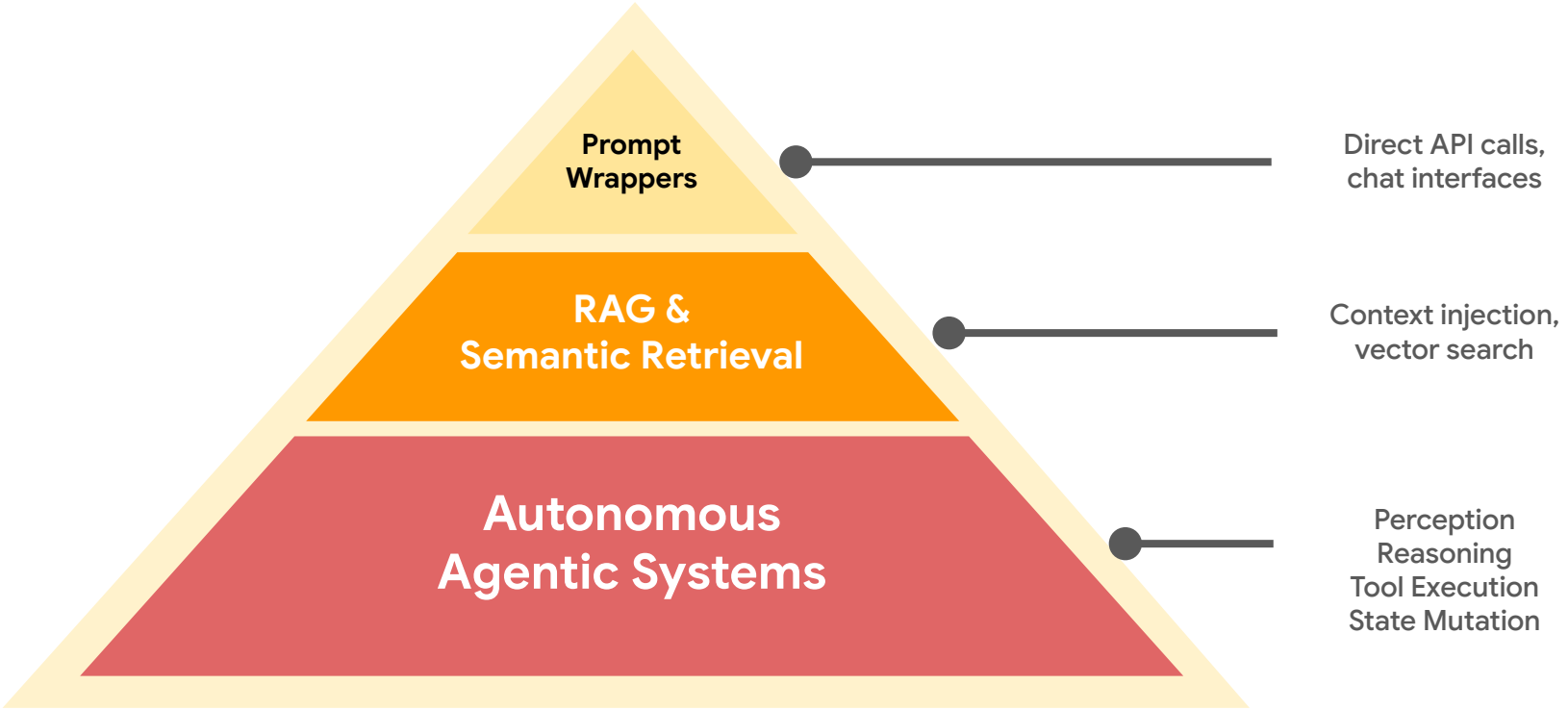
duupr mojis

Multi-Platform Agentic Apps

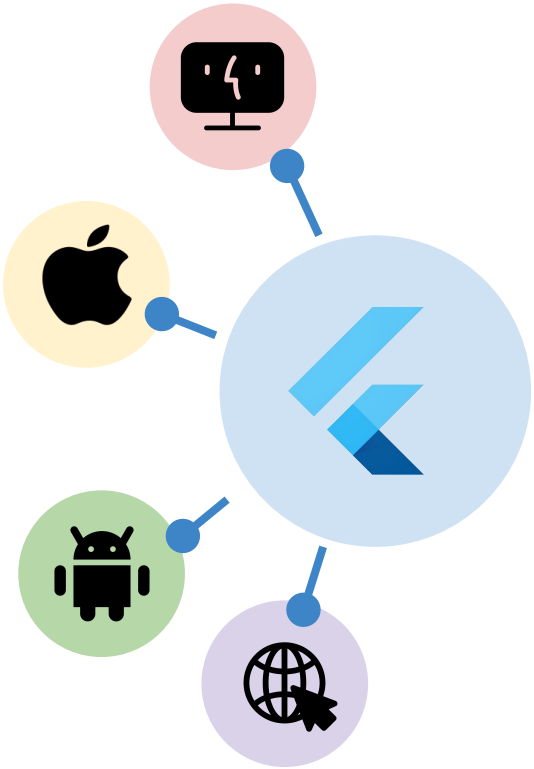
Why Flutter is the Ultimate Shell for Autonomous AI





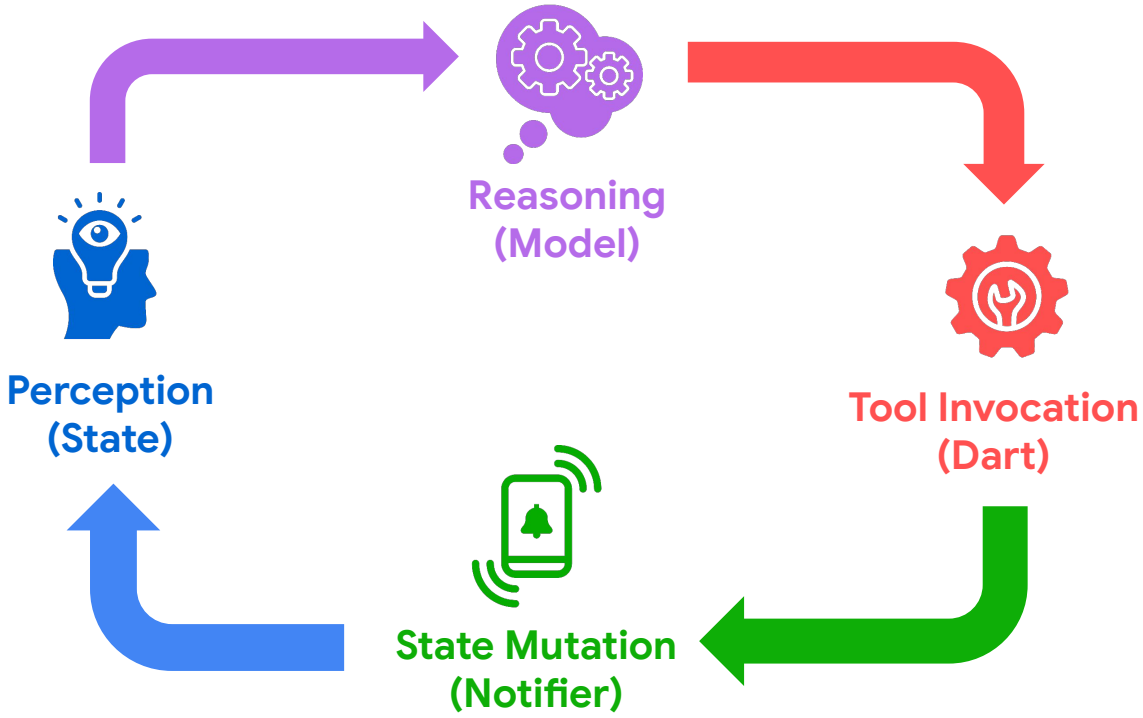

Prompt
Wrappers
RAG &
Semantic Retrieval
Autonomous
Agentic Systems
Direct API calls,
chat interfaces
Context injection,
vector search
Perception
Reasoning
Tool Execution
State Mutation

Why Flutter is the Native Shell for Agents



- Unified Tool Surface
- High-Performance Event Loop
- Ubiquitous Deployment

The Core Agent Engine Architecture



DEMO

Standardizing Tools with MCP & JSON Schemas

- Treating Flutter services
  (Location, SQLite, Bluetooth printing)
  as callable functions.
- Strict JSON schema validation
  for inputs and outputs.
- Error recovery: Handling LLM tool hallucinations
  deterministically in Dart.

```dart
// 1. The Tool Contract: The LLM ONLY returns matched Product IDs
class MenuRecommendationTool {
  static const name = 'show_recommended_products';

  static final inputSchema = S.object(
    description: 'Displays Kiosk MenuCards for the matched product IDs.',
    properties: {
      'product_ids': S.list(
        description:
            'List of product IDs from the menu JSON matching the user request.',
        items: S.string(),
      ),
    },
    required: ['product_ids'],
  );
}
```

```dart
// 2. Strict Dart Model for our SQLite Product Row
typedef Product = ({String id, String name, double price, String imageUrl});

// 3. Validate LLM Input JSON BEFORE querying SQLite
List<String> validateProductIds(Map<String, dynamic> json) => switch (json) {
  {'product_ids': List ids}
      when ids.isNotEmpty && ids.every((e) => e is String) =>
    List<String>.from(ids),
  _ => throw FormatException('Expected non-empty product_ids list. Got: $json'),
};

// 4. Validate SQLite Output (Catches hallucinated IDs not found in DB)
List<Product> validateDbResults(
  List<String> requestedIds,
  List<Product> dbRows,
) => dbRows.length == requestedIds.length
    ? dbRows
    : throw const FormatException('LLM hallucinated a Product ID not in DB.');
```

```
// 5. Wire Validated IDs -> Local SQLite Query -> Kiosk MenuCards
final menuCatalogItem = CatalogItem(
  name: MenuRecommendationTool.name,
  dataSchema: MenuRecommendationTool.inputSchema,
  widgetBuilder: (CatalogItemContext context) {
    // Grab and validate the IDs returned by the LLM
    final ids = validateProductIds(context.data as Map<String, dynamic>);

    // Query our local DB by IDs and render MenuCards
    return FutureBuilder<List<Product>>(
      future: _sqliteDb
          .getProductsByIds(ids)
          .then((rows) => validateDbResults(ids, rows)),
      builder: (ctx, snap) => !snap.hasData
          ? const CircularProgressIndicator()
          : Wrap(
              spacing: 16,
              children: snap.data!
                  .map((product) => MenuCard(product: product))
                  .toList(),
            ),
    );
  },
);

final kioskCatalog = Catalog([menuCatalogItem]);
```

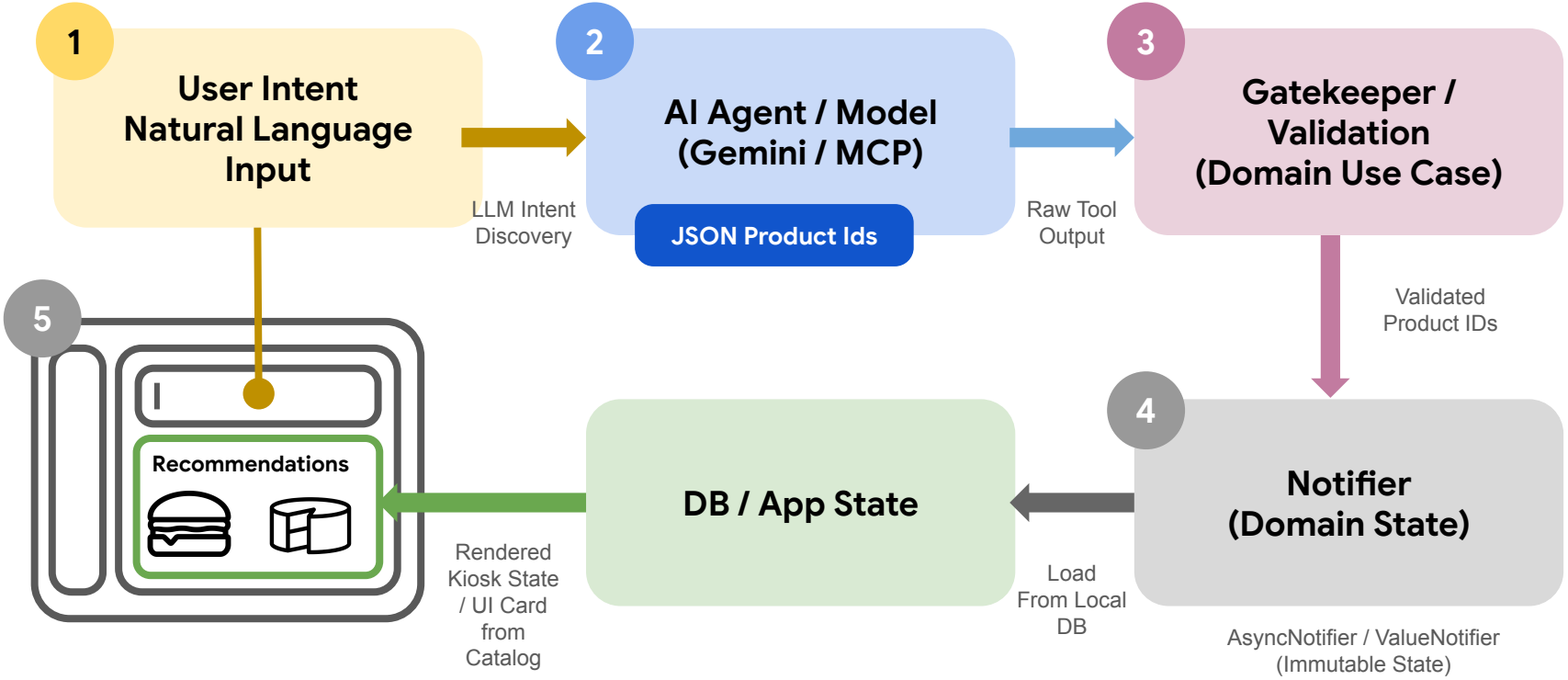
```dart
// 6. Send Menu JSON + Craving -> Get IDs -> Self-Correct -> Render UI
Future<Widget> recommendFromMenu(
  BuildContext ctx,
  String menuJson, // Whole menu JSON with product IDs
  String userCraving, // "Something with chicken, super tasty, and a drink"
) async {
  final req = <Content>[
    Content.text('MENU:\n$menuJson\n\nREQUEST: $userCraving'),
  ];
  var call = (await model.generateContent(req)).functionCalls.first;
  final item = kioskCatalog.items.firstWhere((i) => i.name == call.name);

  try {
    final ids = validateProductIds(call.args);
    final rows = await _sqliteDb.getProductsByIds(ids);
    validateDbResults(ids, rows); // Verifies no hallucinated IDs!
  } on FormatException catch (e) {
    // Hallucinated ID or bad JSON caught! Feed error + schema back to Gemini
    req.addAll([
      Content.model([call]),
      Content.functionResponse(call.name, {
        'isError': true,
        'error': e.message,
        'expectedSchema': item.dataSchema.value,
      }),
    ]);
    call = (await model.generateContent(req)).functionCalls.first;
  }

  return item.widgetBuilder(
    CatalogItemContext(data: call.args, buildContext: ctx),
  );
}
```

Intent-Driven UI

UDF (Unidirectional Data Flow)



DEMO


MANAGE 3 AI AGENTS TO GENERATE SWIFT, KOTLIN, & REACT
JUST USE FLUTTER FOR 1 SHARED CODEBASE

Why generate 3 codebases with AI when you can generate 1 that actually runs everywhere?



VS

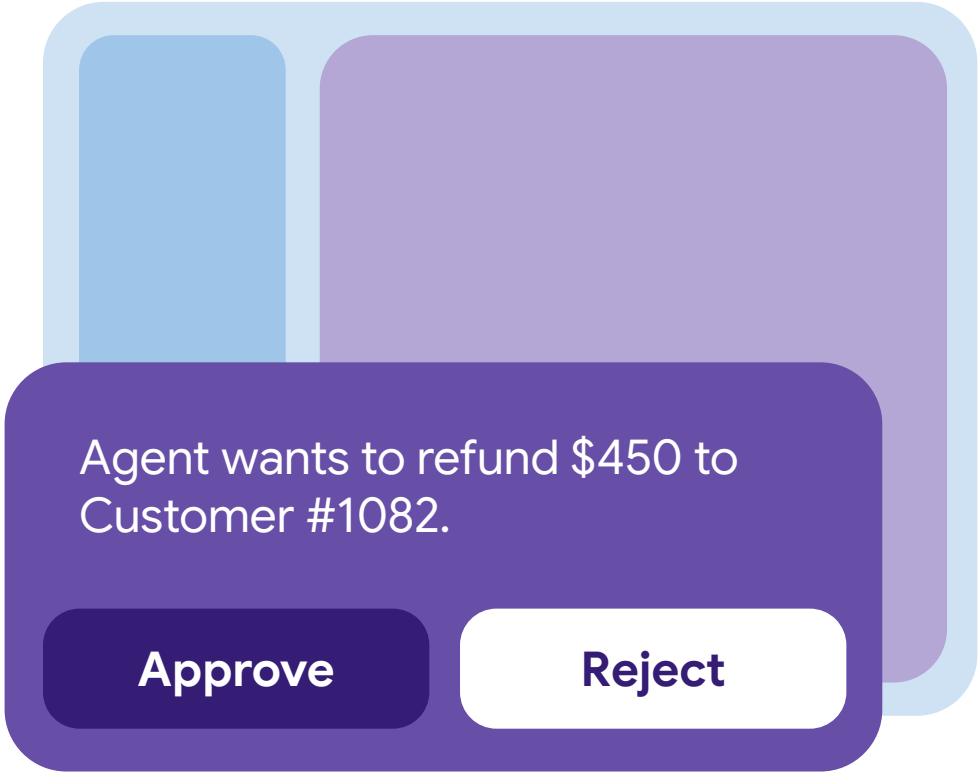


3 separate LLM context windows
(Swift + Kotlin + React) - high token usage
3 build chains, 3 async models, parity drift.

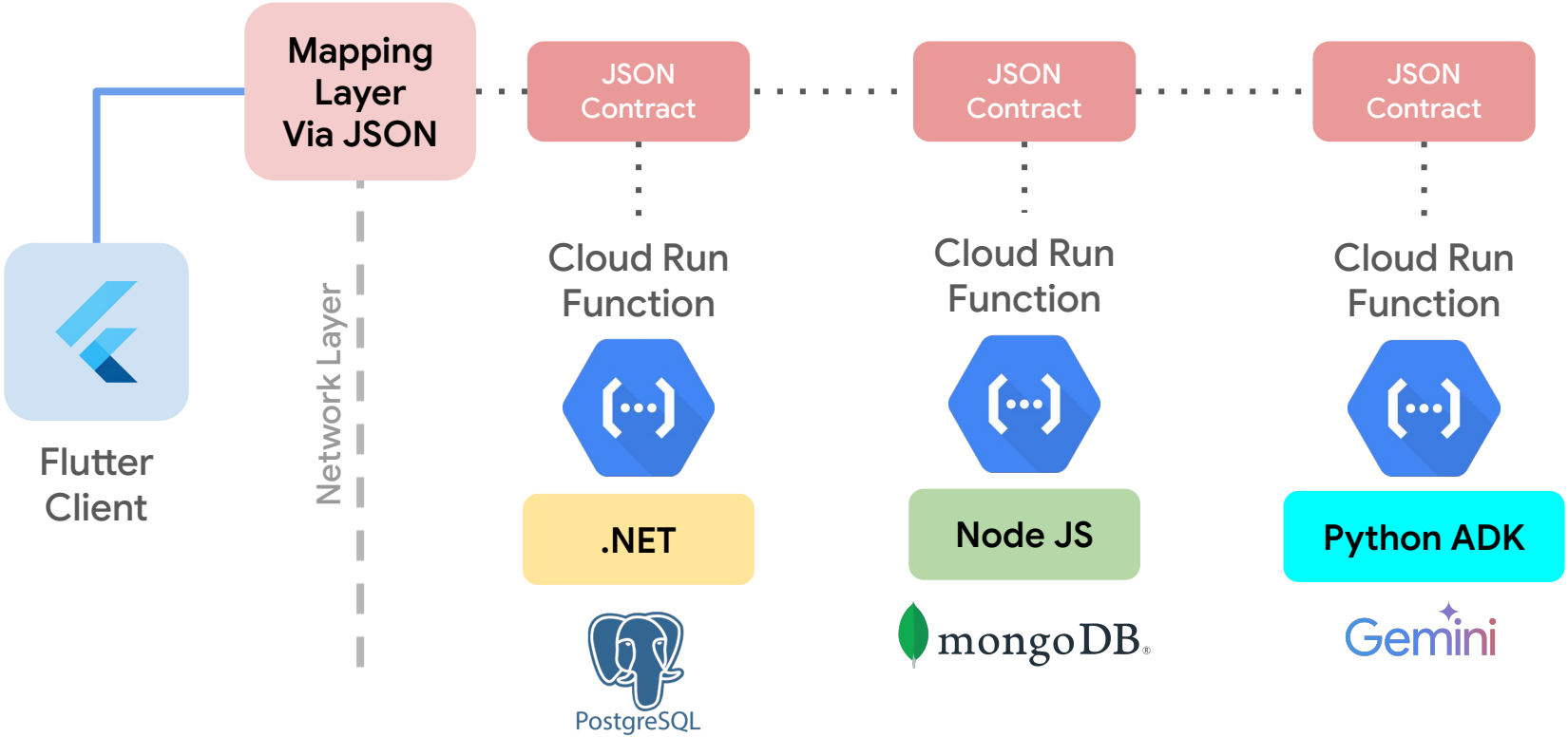
1 shared context window
1 token burn
1 single source of truth, instant hot reload.

Human-in-the-Loop (HITL) & Safety Guardrails

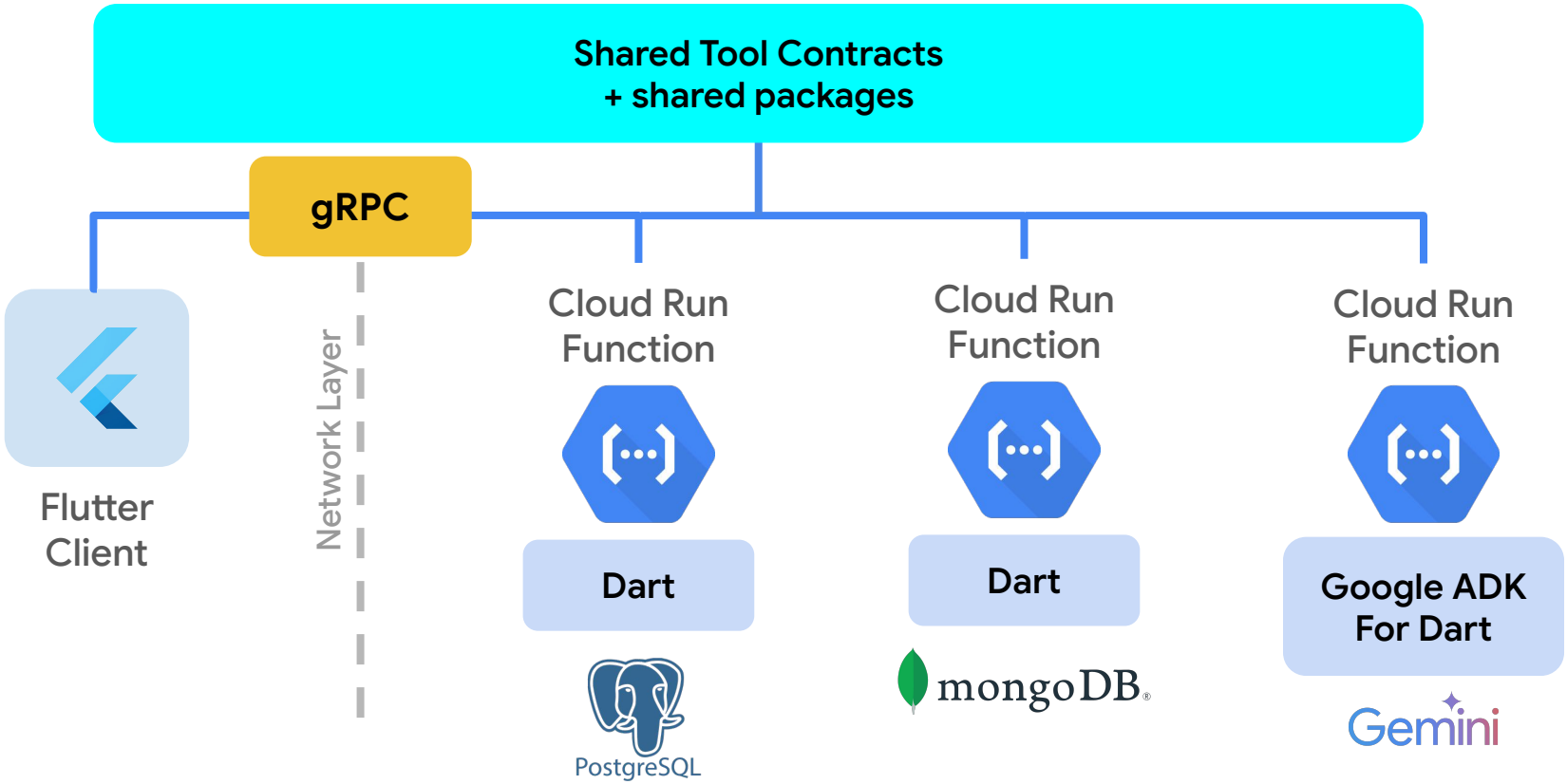
- Autonomy with boundaries: High-impact actions (financial transactions, data deletion, external dispatches) require explicit user sign-off.
- Guardrail layers: Rate limiting, schema validation, and fallback mechanisms.



The Full-Stack Holy Grail (Dart Everywhere)



The Full-Stack Holy Grail (Dart Everywhere)



```dart
import 'package:adk/adk.dart';

class EchoModel extends BaseLlm {
  EchoModel() : super(model: 'echo');

  @override
  Stream<LlmResponse> generateContent(
    LlmRequest request, {
    bool stream = false,
  }) async* {
    yield LlmResponse(content: .modelText('hello from adk package'));
  }
}

Future<void> main() async {
  final Agent agent = Agent(name: 'root_agent', model: EchoModel());
  final InMemoryRunner runner = InMemoryRunner(agent: agent);
  final Session session = await runner.sessionService.createSession(
    appName: runner.appName,
    userId: 'user_1',
    sessionId: 'session_1',
  );

  final List<Event> events = await runner
      .runAsync(
        userId: 'user_1',
        sessionId: session.id,
        newMessage: .userText('hi'),
      )
      .toList();

  if (events.isNotEmpty) {
    print(events.last.content?.parts.first.text);
  }
}
```

Conclusion



The Mental Model

The Agent is the Operator, Dart is the Nervous System, Flutter is the Viewport



The Architecture

Decoupled Notifiers, Intent-Driven Payloads GenUI when applicable



The Leverage

(1 Unified Codebase > 3 AI-Generated Native Codebases)

Resources

Flutter
https://docs.flutter.dev/

Riverpod
https://riverpod.dev/

GenUI
https://docs.flutter.dev/ai/genui

DuuprMojis
https://duuprmojis.duupr.io/

Follow me @drcoderz

Let's connect on LinkedIn



Thank You!