KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

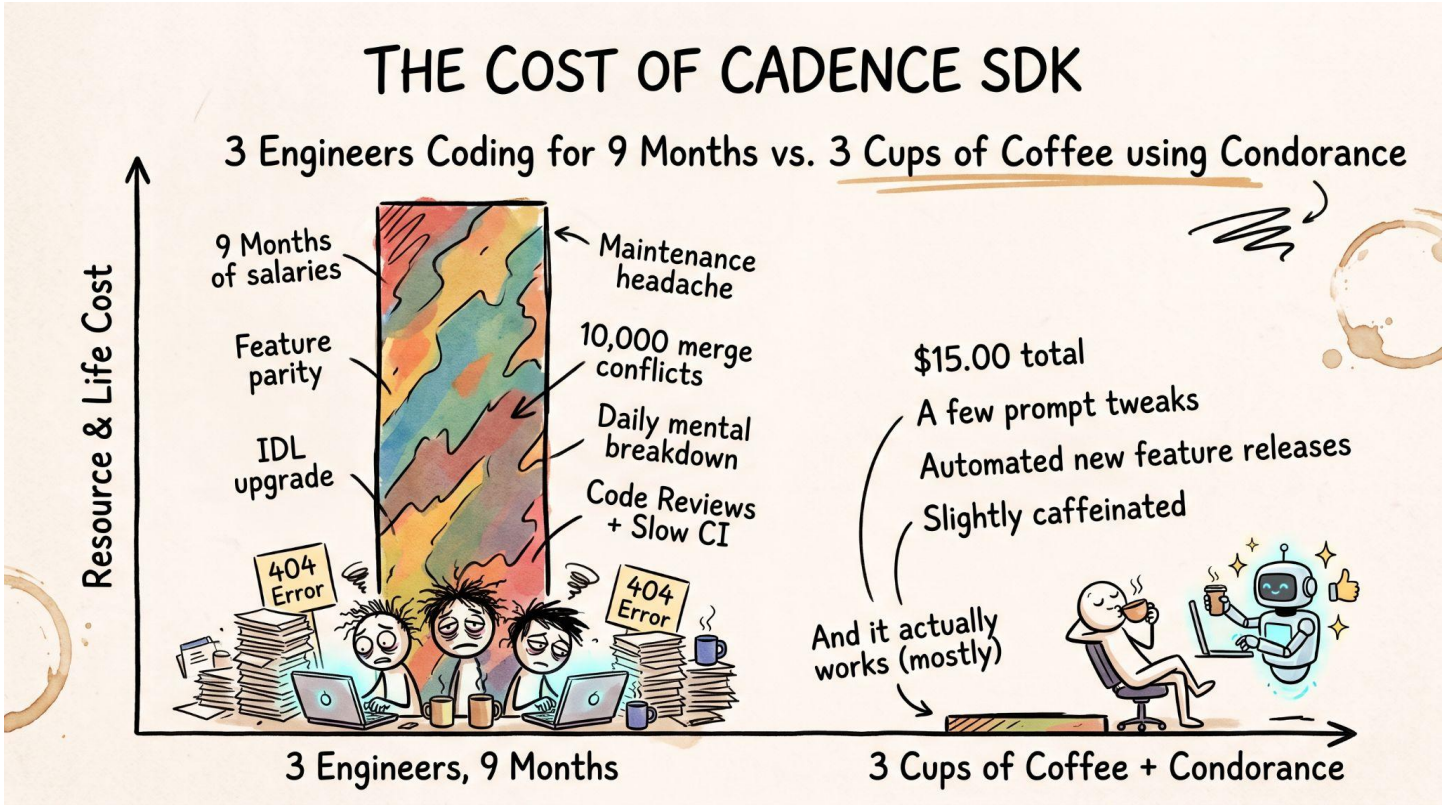
Building and Maintaining Production-Ready SDKs with LLMs




Ender Demirkaya


Shijie Sheng

Why do we want to do it?



Why do we want to do it?



Multiply an existing SDK with low effort

→ or **Migrate** to another language



Support more languages => More users



Avoid **SDK divergence**



Standardize codebase



Familiar paths across SDKs



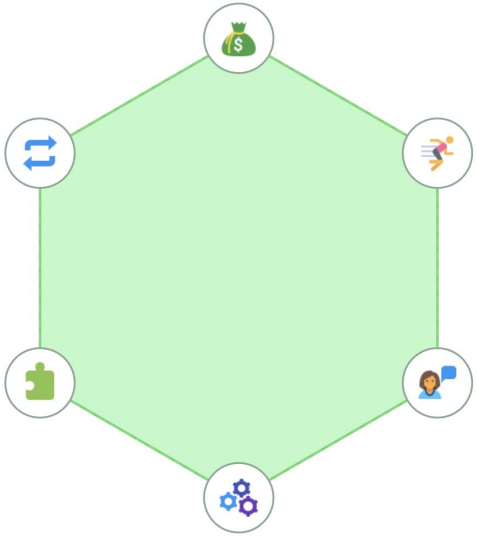
Incentive for **cleaner code** in the source SDK

How do we want it?

Cheap

Fast

Consistent



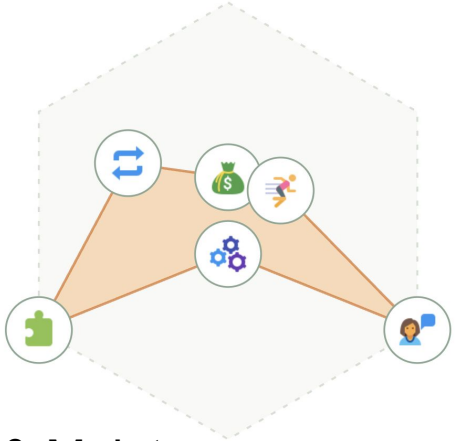
Idiomatic

Native

Automated

Past Attempts (1): Fully disjoint Language-Native SDKs

- Currently Active since 2017
- Golang, Java, Python SDKs



Pros:

- **Idiomatic** APIs
- **Native** ecosystem integrations
- Correct / Best **abstractions**
- Language Specific **Optimizations**
- Independent Evolution (pro?)

Cons:

- Most **expensive**
- **Duplicate** Effort & Maintenance
- Behavioral **Drift**
- **Uneven Feature Coverage**
- Cross-language **friction** (e.g. naming)

Past Attempts (2): Shared Core

- Considered
- Researched / Tried the Existing Experience



Pros:

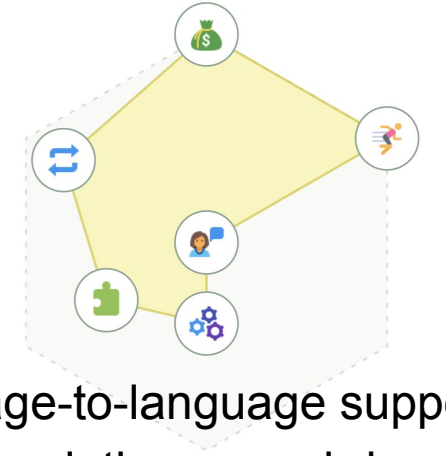
- Some parts are **implemented once**
- Wrappers preserves **idiomatic features**
- Shared core's **performance** can be higher than wrapper language's

Cons:

- Wrappers are **still manual** = still need similar amount of developers
- **Divergence** still happens
- **Inter-process communication** is heavy
- **Debugging** is hard: split stack traces
- **Release coordination** needed

Past Attempts (3): Traditional Transpilers

- Attempted 6-12 months
- Dafny, Haxe, goscript …



Pros:

- **Exact copy** of the source language
- Generates in **seconds = lowest cost**
- Outputs only in target language
- **No FFI*** / inter-process calls
- Call stack stays in **one language**
- **Familiar idioms** / APIs with wrappers

*FFI: Foreign Function Interface, cross language calls

Cons:

- **Limited** language-to-language support
- **Line-by-line** translation => weird
- Works but too **obscure** to follow
- **Intermediate types** for compatibility (custom ints, strings, …)
- **Third-party dependencies** won't work
- Heuristics **fail hard problems**

Traditional Transpilers: Dafny

```
// added {:axiom} to silence warning about nat's requirements (it cannot be ensured here)
method {:extern} {:axiom} Random(n: nat) returns (r: nat)

// helper to check expectations, as extern cannot do it alone (ensures cannot be declared there)
method Random'(n: nat) returns (r: nat)
    requires n > 0 // else r < n cannot be guaranteed
    ensures r < n
{
    r := Random(n);
    expect r >= 0; // NOT required, ensure it is a nat

    // either this expect or the if statement is required, to check the `ensures`
    expect r < n;
    if r < n {
        return r;
    } else {
        return n-1;
    }
}
```

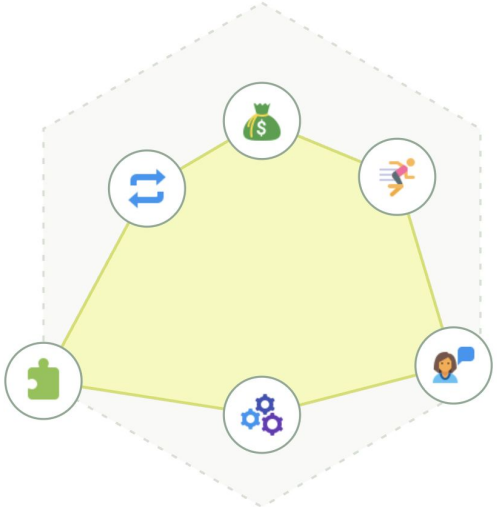
An intermediate Dany (.dfy) file

Traditional Transpilers: goscript

```
// Code Snippet (full example https://github.com/shijiesheng/cadence-goscript-client)
export async function HelloWorldWorkflow(ctx: workflow.Context, name: string): Promise<[string, $.GoError]> {
    const greeting = $.varRef("")
    const future = await workflow.ExecuteActivity(
        ctx,
        HelloActivityDefinition,
        $.arrayToSlice<any>([name]),
    )
    const err = await future!.Get(
        ctx,
        $.interfaceValue(
            greeting,
            "*string",
            $.pointerType($.basicType("string")),
        ),
    )
    if (err != null) {
        return ["", err]
    }
    return [greeting.value, null]
}
```

Past Attempts (4): AI Based Transpiler (Attempt #1)

- Tried 3-6 months
- Per file/package translation
- Hands-off: LLM does everything*



Pros:

- Output is in **native language**
- **Solves hard problems**
- Grasps **context** (vs line-by-line)
- **Idiomatic** Translation
- AI **gets better** everyday
- Open ended: **no limitation**

Cons:

- **Expensive**
- **Hallucinates**
- **Inconsistent** / non-deterministic
- **Fine tuning** is quite hard / impossible
- **Prompt iterations**
  - fixes one thing, breaks N things

Meet **Condorance** (5): AI Based Transpiler (Attempt #2)

- MVP took ~6 months
- Any language to any language
- Working on a release
- GitHub: **cadence-workflow/condorance**



Inspired by the Andean condor soaring over Patagonia.

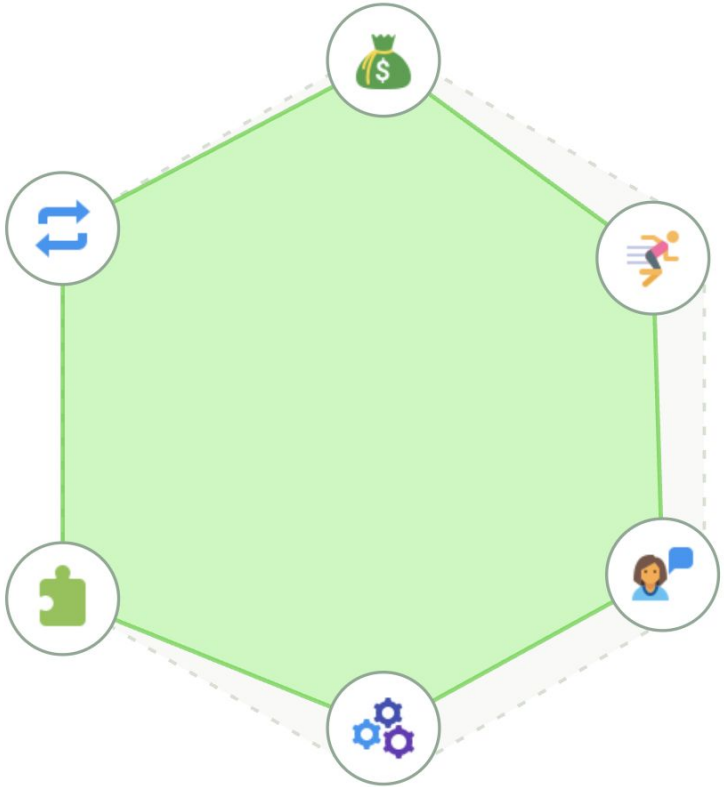
- Soars high: understands the high level objectives
- Dives deep: captures details

- Creates a "**codegraph**" (not a callgraph!!)
  - Indexes calls, variables, comments, any relationship
- Creates **contracts** for functions, files, packages
- Generates the target skeleton with **heuristics**
- Translates code into a desired language



Where does it **land**?

- **Cheap**: ~$5 per Cadence SDK
- **Fast**: A couple of hours
- **Idiomatic**: preserves conventions
- **Automated**: almost fully hands-off
- **Native**: integrates well
- **Consistent**: keeps SDKs in sync



Problem #1: Expensive

- Use **cheap models**: Qwen, Kimi, GPT6-Luna
  - Entire codebase generation costs ~$5
    - **GPT5.6 Luna**, also works with local LLMs Qwen, Kimi
- Only **generate the diffs**
  - Don’t regenerate already generated (stable) code
  - Understand changes, files, functions, variables
  - Generate from the leaves => parents

but what about …

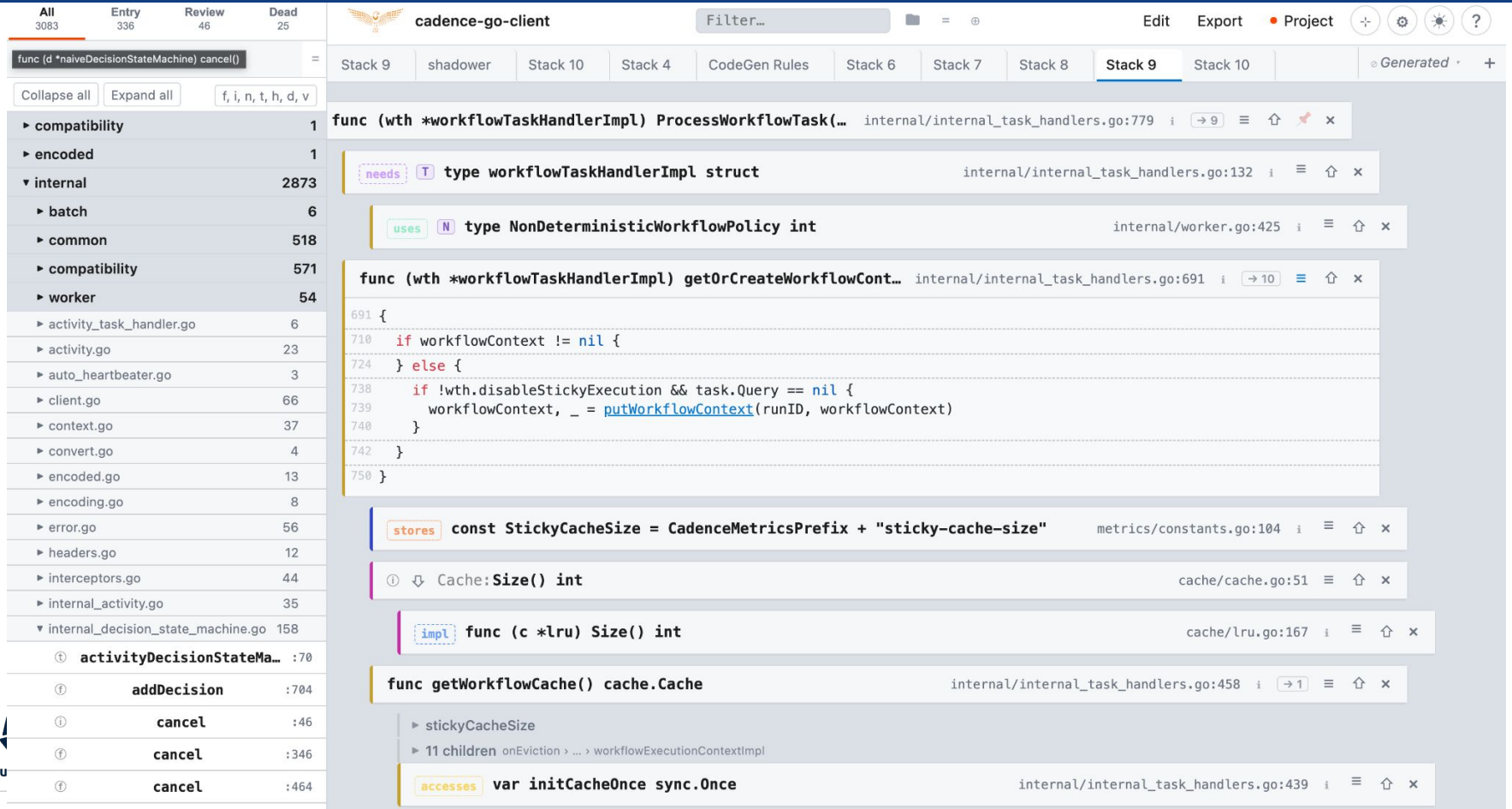
Problem #2: Hallucinates / Inconsistent

Make problems simpler & smaller

- ***Codegrapher*** creates the entire codegraph # including comments
  - Context for AI translation
  - **Inspector** visualizes the codegraph for humans to understand the codebase
- *Generated* ***Contracts*** drives AI-TDD oriented development
- Translation **Hints**
  - Third party library replacements
  - Target Language rules
  - Source to target language idiom translation guidance
- Use **heuristics** where you can (e.g. skeleton)
- AI translates **small** contained **pieces**

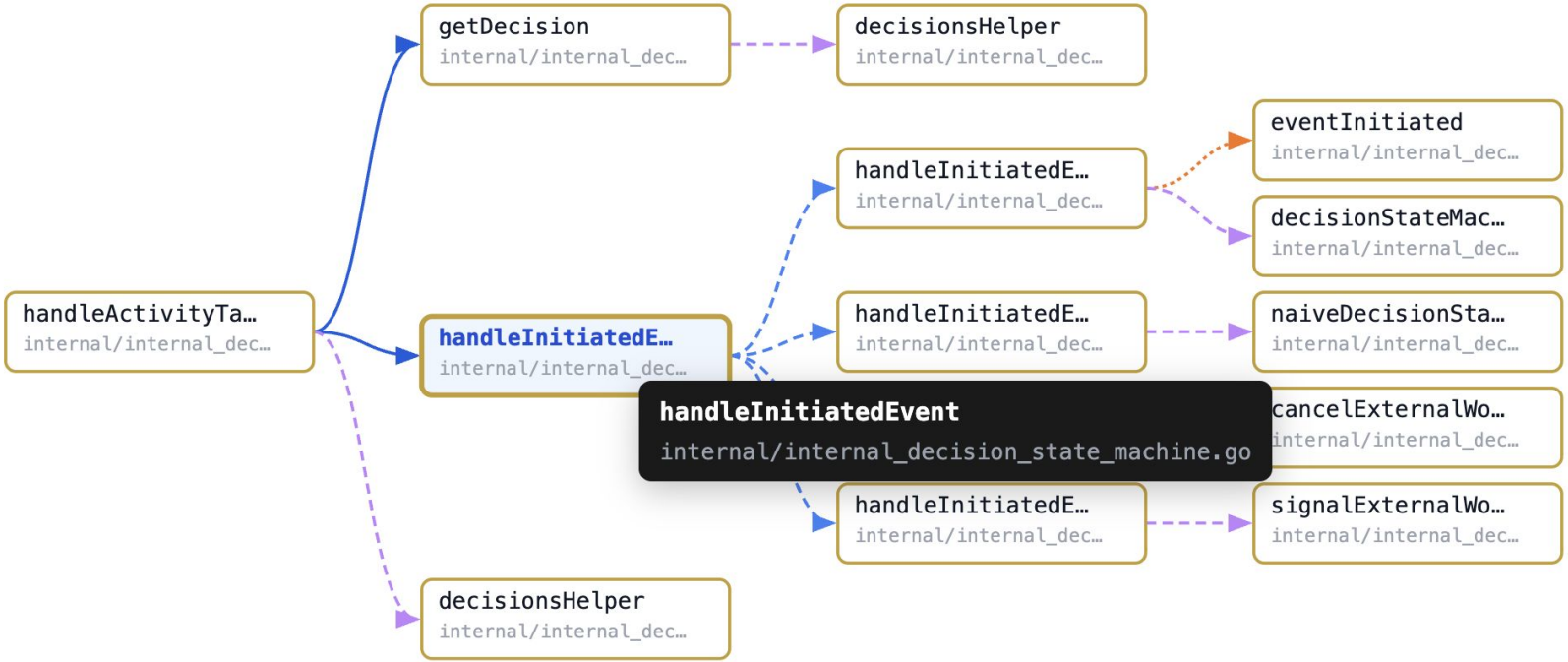
… and it does a great job at that!

Codegrapher & Inspector

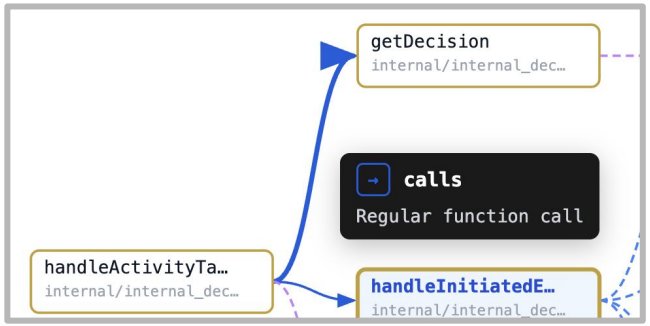


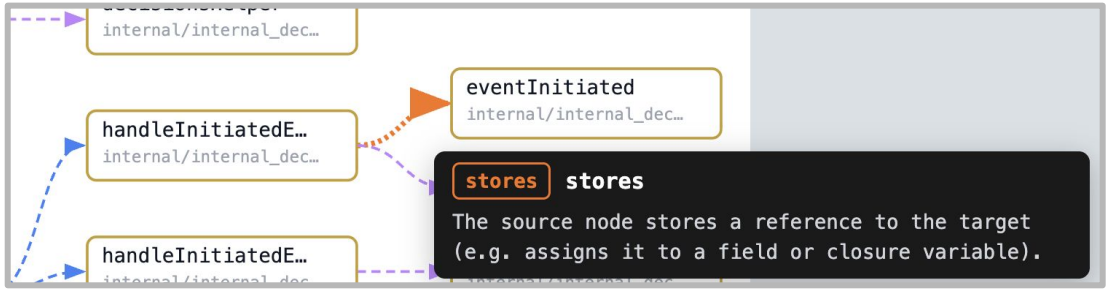
Codegrapher & Inspector

Projects are exportable as standalone **HTML, Markdown, PDF** and **Importable JSON**



Codegrapher & Inspector



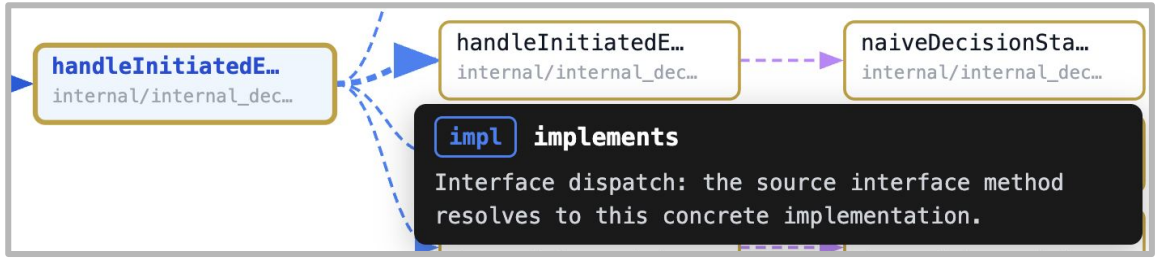


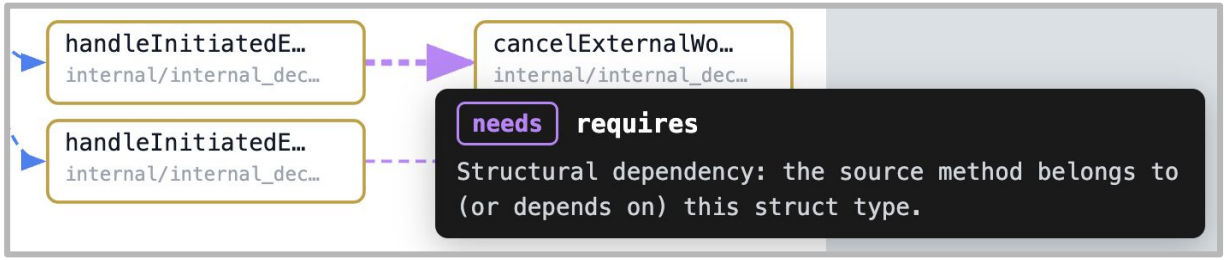
11

Connection Types

7

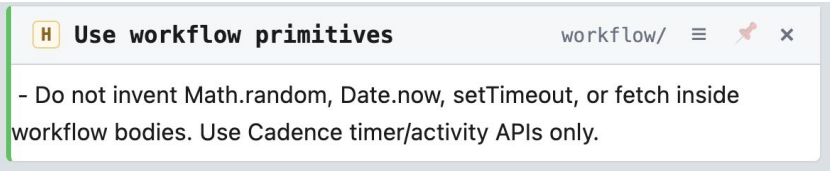
Node Types

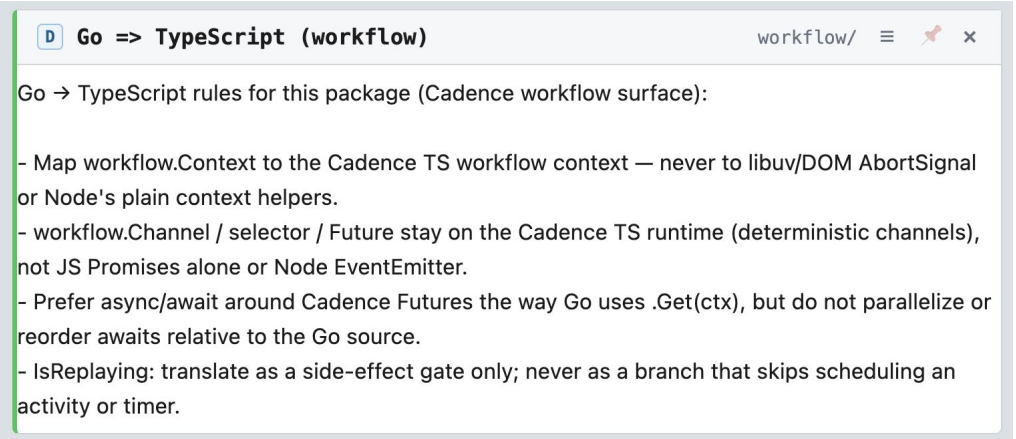




Problem #3: Fine Tuning / Preferences

- **User hint files (preferences)**
  - General guidance **overrides**, project-specific guidance
  - Can be provided **at any level**: project, package, file, function
- **Contracts**
  - Defines exactly what a function, file, package is supposed to do
  - Code, comments and hints influence contracts

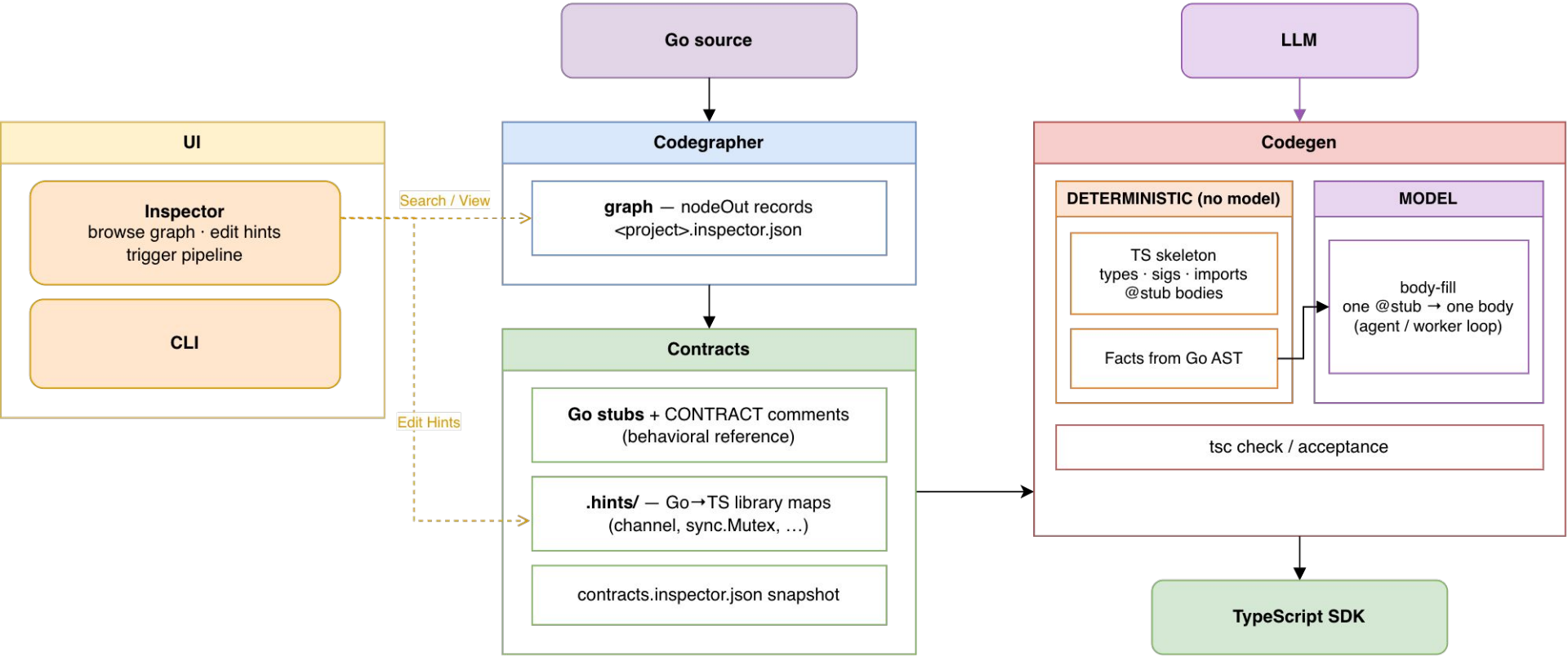




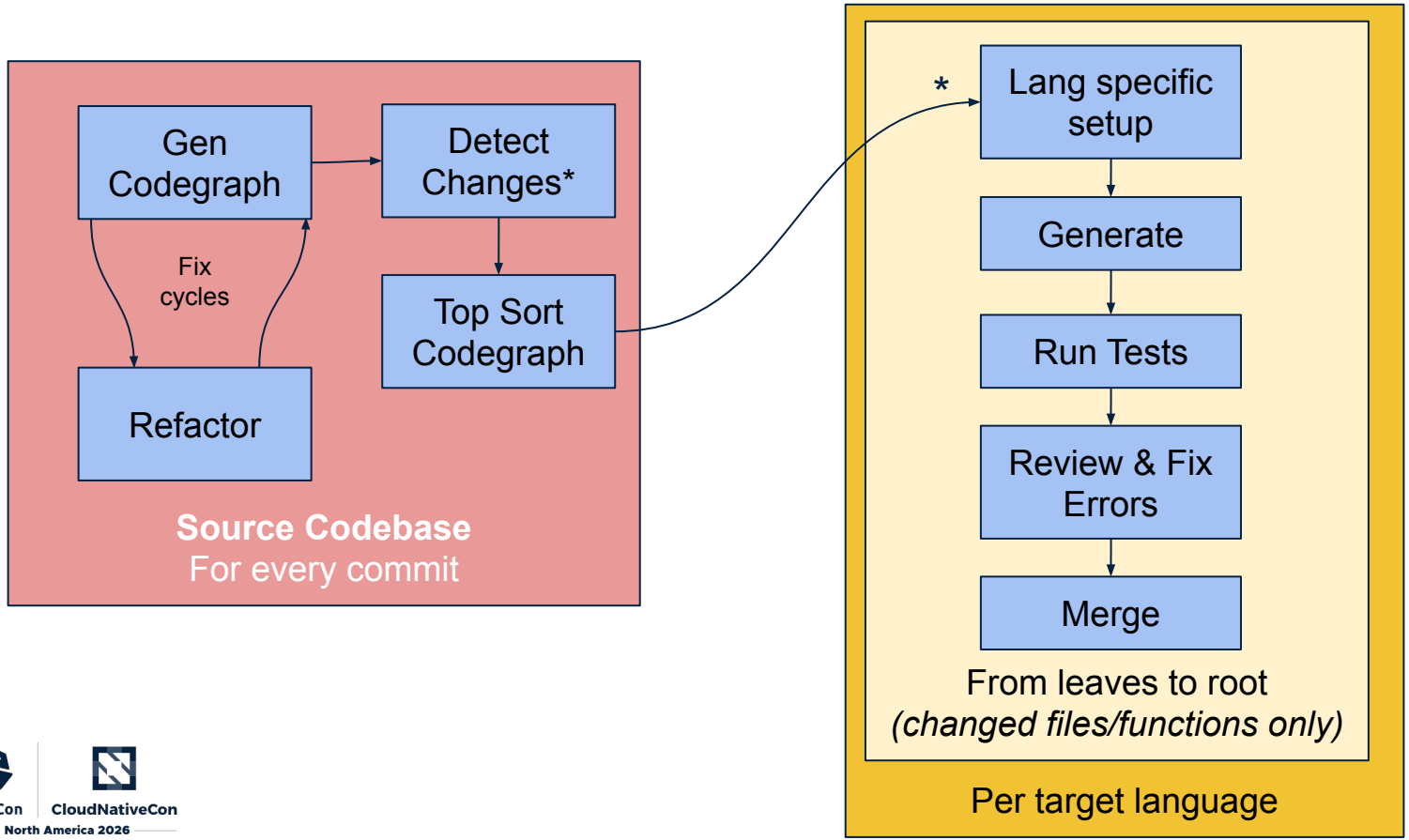
Problem #4: Iterations => Prod Ready

- Maintain high **first generation** quality
- **Test & Iterate** to finish
  - Type checks, linters
  - Builds
  - Unit tests
- **Error output**s used as inputs to iterate
- **Manual fixes:** Last line of defense
  - Retained (remember, we only generate diffs)
  - **Used as an input** when those parts are edited
  - So far we haven't edited generated code

Architecture



Continuous Generation on Source Changes



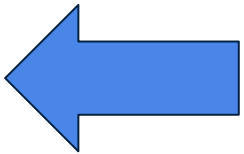
TS Heuristic Skeleton

Example: [Tunny](#) (Library for Goroutine Pool)

TypeScript Skeleton:

```typescript
// Pool is a struct that manages a collection of workers
export class Pool {
    constructor(init?: Partial<Pool>) {
      Object.assign(this, Object.fromEntries(Object.entries(in
    }
    queuedJobs: number = 0;
    ctor?: () => Worker;
    workers: workerWrapper[] = [];
    reqChan?: Channel<workRequest>;
    workerMut: Mutex = new Mutex();

    async Process(payload: unknown): Promise<unknown> {
      throw new globalThis.Error("not implemented"); // @stub
    }
  }
```



Golang Source

```go
// Pool is a struct that manages a collection of workers
type Pool struct {
    queuedJobs int64

    ctor    func() Worker
    workers []*workerWrapper
    reqChan chan workRequest

    workerMut sync.Mutex
}

// Process will                                    oad
func (p *Pool) Process(payload interface{}) interface{} {
    atomic.AddInt64(&p.queuedJobs, 1)

    request, open := <-p.reqChan
    if !open {
        panic(ErrPoolNotRunning)
    }

    request.jobChan <- payload

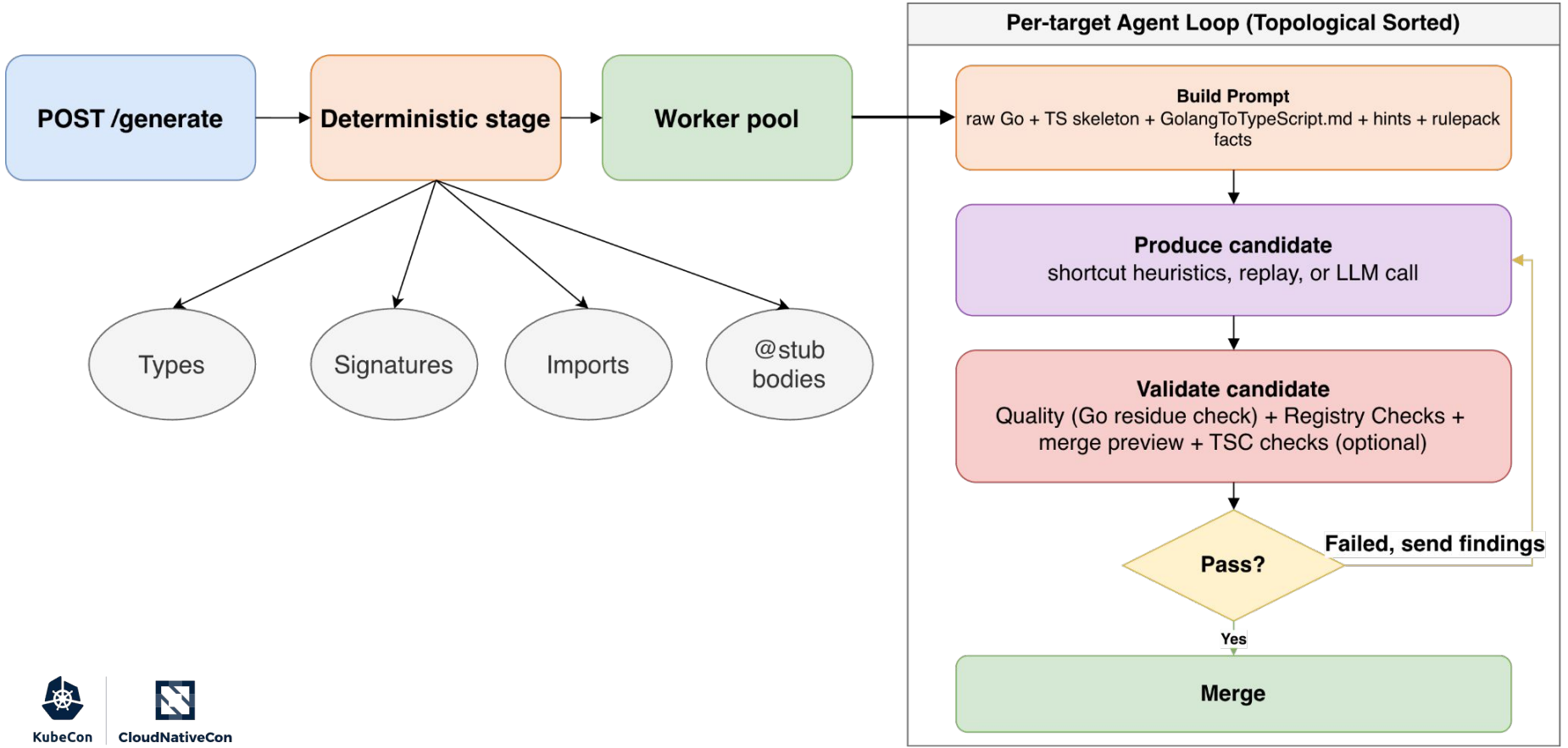
    payload, open = <-request.retChan
    if !open {
        panic(ErrWorkerClosed)
    }

    atomic.AddInt64(&p.queuedJobs, -1)
    return payload
}
```

Translation Hints

```
# HINT — sync.Mutex
# Category: sync
#
# The best available alternative has been pre-selected for each language.
# Change "selected:" to any other option listed below, or write "skip" to omit.
# Leaving selected blank (or picking multiple) will fail "contracts check".
#
# To refresh them, click "Generate Contracts" on the struct node in the
# inspector with the force option enabled. This rewrites the inline field
# comments inside the contracts file while preserving everything else you
# have written.
[typescript]
#   Mutex      [async-mutex]  Go-style async mutex for TypeScript
#   Semaphore  [async-mutex]  weighted semaphore (allows concurrent readers)
#   skip       [-]            omit — JavaScript is single-threaded; explicit locks are rarely needed
selected: Mutex
[python]
#   asyncio.Lock      [-]  async mutex for coroutine-based code (stdlib)
#   aiorwlock.RWLock  [-]  async read-write lock — pip: aiorwlock (for sync.RWMutex)
#   threading.Lock    [-]  mutex for multi-threaded synchronous code (stdlib)
#   skip              [-]  omit — restructure to avoid shared mutable state
selected: asyncio.Lock
```

CodeGen (Agent Loop)



Generating the ~~Typescript~~ Vybescript SDK

```typescript
export interface HelloInput {
  message: string;
}

// HelloWorldWorkflow calls HelloWorldActivity and returns a greeting string.
export async function helloWorldWorkflow(ctx: workflow_Context, input: HelloInput): Promise<string> {
  const logger = GetLogger(ctx);
  logger.info('HelloWorldWorkflow started');

  const activityCtx = WithActivityOptions(ctx, {…
  });
  const greeting = { value: '' };
  const future = await ExecuteActivity(activityCtx, helloWorldActivity, input);
  await future.Get(activityCtx, greeting);

  logger.info('Workflow result', `greeting=${greeting.value}`);
  return greeting.value;
}

// HelloWorldActivity formats and returns a greeting for the given input.
export async function helloWorldActivity(_ctx: unknown, input: HelloInput): Promise<string> {
  console.log('[INFO] HelloWorldActivity started');
  return `Hello, ${input.message}!`;
}
```

Sample TS Cadence workflow

Total lines of code in [cadence-typescript-client](): **47,801**

Structure Choice

Heuristical (current)

A heuristic creates the initial skeleton, LLM fills inside

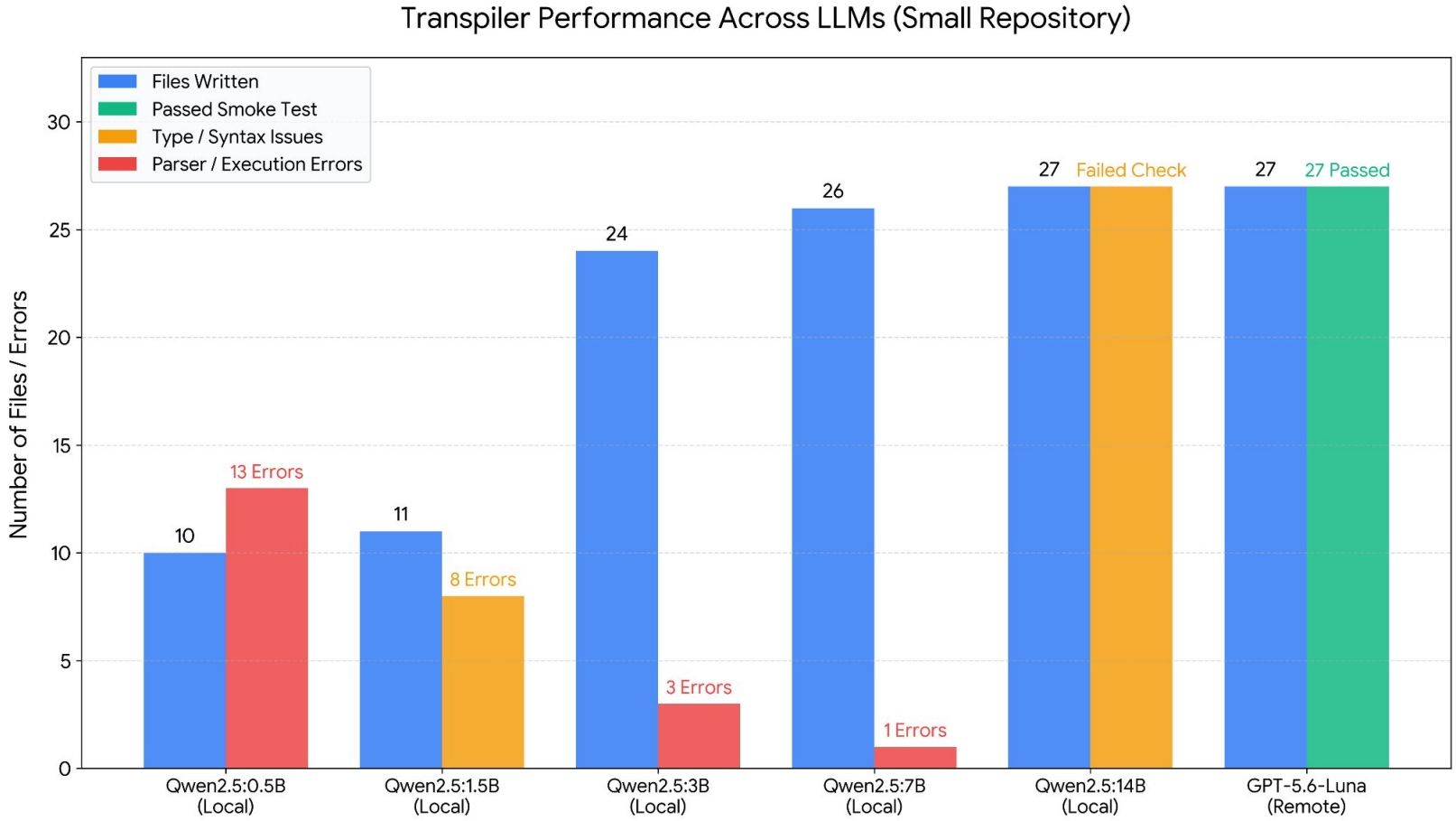
- **Pros**:
  - Deterministic
  - Parallelizable
  - Cheap
- **Cons**:
  - Less idiomatic
  - Output feels like the source
  - Needs a heuristic for every target

Objective Oriented 

Exposed APIs are taken as objectives, LLM has more freedom towards leaves

- **Pros**:
  - Idiomatic
  - Optimized
  - Intuitive / Natural
- **Cons**:
  - Less deterministic
  - More error prone
  - Harder to fine-tune

LLM Model Choices (Local LLMs)



How can you use it?

- Release cadence-workflow/condorance (in progress)
  - To be announced in #cadence-users, LinkedIn, cadenceworkflow.io
- Continuous Generation Pipeline (in progress)
  - “Generate B from A considering diffs between two commits on A”
- Transpiling from Language A to B
  - Add LanguageB.md (if it doesn’t exist already)
  - Add AtoBTranslationGuide.md (if it doesn’t exist already)
  - (Add hints for guidance override if you need them)

That’s it!!



Acknowledgements

KubeCon CloudNativeCon North America 2026



Hengchen Yuan

https://yhcrown.github.io/

The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering



August Shi
Associate Professor

https://ece.utexas.edu/people/faculty/august-shi

Q&A

KubeCon CloudNativeCon North America 2026

```
// GENERATED: DO NOT MANUALLY EDIT THIS FILE
//
// Generated by Condorance: github.com/cadence-workflow/condorance
//
//
```



```
//
//
//
```

KubeCon CloudNativeCon
North America 2026

Thank You!



Ender Demirkaya
Uber, Cadence
https://www.linkedin.com/in/enderdemirkaya



Shijie Sheng
Uber, Cadence
https://www.linkedin.com/in/liouvetren/

Backup Slides



Background: where academia stands

Still a research frontier: the newest results are poor.

> *Best method — an agent framework on frontier models, with test-failure feedback — reaches **32.8% success** across 1,897 real repositories; plain prompting is far lower.*
>
> **— RepoTransBench: A Real-World Multilingual Benchmark for Repository-Level Code Translation (TSE 2026)**

The larger the project, the worse the accuracy.

> *A sharp scaling collapse: agents average **91.3%** on repositories under 10K LOC, and **15.3%** beyond 50K LOC. "Agents… handle the most visible features while omitting the long tail of functionality."*
>
> **— RepoMod-Bench: 21 real repositories, 8 languages, implementation-agnostic tests (2026)**

What breaks is the cross-file structure, not the syntax.

> *LLM translation is feasible for small programs, but generating functional **build systems** and managing **cross-file dependencies** prevent scaling to full codebases.*
>
> **— ParEval-Repo: Repository-Level HPC Translation (ICPP 2025)**

> **Our subject: ~380K lines of Go, with a ~45K-line hand-written concurrent core. Far past the scale where these numbers collapse.**

Where Industry Stands

Google (2025): about 70% of internal migration edits were LLM-generated, saving about half the time. Same-language version and API upgrades, every change human-reviewed.

Amazon Q Code Transformation (2024): Java 8/11 → 17 at fleet scale, about 4,500 developer-years saved. Same language.

Airbnb (2025, published): 3,500 Enzyme test files migrated to React Testing Library in 6 weeks, against a hand-migration estimate of 1.5 years. JS to JS, codebase structure unchanged.

Bun (2026): a popular JavaScript runtime, ~535K lines of Zig, ported to Rust by 64 concurrent Claude agents (~6,800 commits in ~2 weeks). The closest attempt yet to "agents translate a whole repo". The result today: still experimental, not the default binary, 19 known regressions, 13K+ unsafe blocks. The Zig language creator publicly called it "unreviewed slop".

The shipped wins keep the language and the structure fixed, and humans review every change. The one whole-repo attempt by agents is still not trusted.

Automatic, faithful, cheap, and verified on a live system: that slot is open. It is what this project targets.

Challenges: Sync => Async

Problem:

TS requires explicit async keyword for async functions

vs

Go doesn't have such requirement

Solution:

Mark async Go functions with static analysis

```
export class localActivityTask {

  Mutex: Mutex = new Mutex();

  async Lock(): Promise<void> {
    await this.Mutex.acquire();
  }

  TryLock(): boolean {
    if (this.Mutex.isLocked()) {
      return false;
    }
    void this.Mutex.acquire();
    return true;
  }

  Unlock(): void {
    this.Mutex.release();
  }
}
```

With the skeleton, can a strong model just fill the bodies?

We tested it: Fable 5, one method (`pollTask`), given the full emitted class skeleton.

```
✗ Go stdlib idioms turned into namespaces that exist nowhere in the tree:
if (!errors.Is(context.Cause(this.limiterContext!), errShutdown)) { ... }

✗ wrong construction protocol. The real channel payload convention is send({ task }):
await Promise.race([
  this.taskQueueCh!.send(new polledTask(task)).then(() => "sent" as const),
  this.shutdownCh!.recv().then(() => "shutdown" as const),
]);

✗ wrong member on the mapped npm rate-limiter. Its real method is removeTokens:
await this.pollLimiter.Wait(this.limiterContext!);

✗ not awaited. Whether Throttle is async is a property of OUR emitted declaration:
this.retrier!.Throttle();
```

> **Better models close the inference gap. They cannot close the information gap.**