



# The Translation Hill Project

Bridging human intent and AI creation through structured conversation

## Overview

The Translation Hill Project is an experiment in designing repositories that think *with* people, not just for them. It starts from a simple observation:

**Coding isn't the hard part – translating human thought into something a machine can execute is.**

Modern AI code assistants often struggle not with writing code, but with understanding what we *really* mean. The gap between a person's idea and precise machine instructions is the "hill" we aim to flatten. This repository's goal is to help anyone – coder or not – express creative and functional ideas so clearly that AI tools can build them into working prototypes.

Instead of a traditional codebase, this project is a **conversation architecture**. It's a structured set of Markdown and YAML files that guide a human and an AI through the natural evolution of an idea – from first spark to production-ready system – via conversational development. In essence, it treats **dialogue as the new compiler**, turning clear instructions into running software.

## The Core Idea

Every successful AI build begins long before the first line of code. It begins with **alignment** – a mutual understanding of how a person thinks, communicates, and what "success" looks like for them. Miscommunications at this stage lead to AI outputs that miss the mark <sup>1</sup>. The Translation Hill Project formalizes the alignment and development process into structured phases. Each phase has its own folder, tone, and goal. Every document doubles as both a *conversation prompt* for the AI and a *record of human intent*.

By treating conversation like an iterative development environment, we ensure the AI "coder" fully understands the human "specification" before diving into solutions. This echoes the emerging *Plan-Act* paradigm: spend time planning and clarifying intent, then let the AI act on it <sup>2</sup>. The AI is encouraged to be inquisitive, ask questions, and mirror the user's language and preferences early on. This front-loads understanding because **intent is the one thing that can't be automated** <sup>2</sup> – only the human can provide it. Once alignment is achieved, the AI can leverage its full power to generate and build with minimal guesswork.

# The Four Phases

We structure the human-AI collaboration into four key phases (plus one meta-layer that *explains* the process). Each phase flows into the next, using lightweight YAML frontmatter to carry over crucial context (personality traits, decisions, requirements). This ensures that each new conversation begins a step ahead of the last, maintaining continuity. Similar multi-step approaches have been used in AI coding workflows – for example, taking a one-line user request and progressively generating a spec, then code, with each step informing the next <sup>3</sup>. Here are the Translation Hill phases:

## Phase 0: Alignment

**Purpose:** Build mutual understanding between human and AI. Define the tone, vocabulary, and complexity that suit the user.

**Human's Role:** Express preferences and examples of their communication style (in plain language). For instance, the user might specify, "I prefer concise responses with occasional humor," or share an example paragraph they liked.

**AI's Role:** Act as an *interpreter* and *mirror*. It asks clarifying questions, adapts its language to match the user's style, and notes key preferences. The AI essentially creates a user profile. *This is not about gathering private data – it's about calibrating to communication style.*

**Key Output:** `PROFILE.yaml` and `CONTEXT.md` – a profile capturing the user's preferences and any context the user finds relevant. These files serve as the "personality" and background for all future phases.

In practice, a good Phase 0 feels like a friendly interview. The AI might ask about the user's desired level of detail, analogies vs. numbers, or how it should handle mistakes. There are already community examples of such preference-gathering prompts. For instance, one approach asks if the user wants "*Concise responses (brief and to the point), Detailed responses (with supporting details), or Expansive responses (comprehensive with nuance)*" <sup>4</sup>. It might also ask about tone (humorous vs. formal) and whether to show its reasoning or not <sup>5</sup> <sup>6</sup>. The result of Phase 0 is a YAML profile that the AI can reference to stay aligned with the user's expectations in every reply going forward.

## Phase 1: Ideation

**Purpose:** Turn abstract desires into structured ideas and actionable concepts.

**Human's Role:** Dream, brainstorm, and react. The human shares high-level goals or even vague ideas ("I want an app that helps schedule my tasks with AI"). They don't need to specify technical details – just what they envision or need, in their own words. They also give feedback on the AI's suggestions, honing the direction.

**AI's Role:** Act as a creative partner – generating alternatives, asking questions to refine the idea, drawing analogies or examples to clarify. Here the AI might don a *MuseAgent* persona to offer outside-the-box ideas, or an *ArchivistAgent* to summarize and organize the discussion so far. The AI proposes multiple approaches or interpretations of the idea, helping the human explore the solution space.

**Key Output:** `IDEA_NOTE.md` – a coherent summary of the chosen idea or a set of notes capturing the brainstorming session. This document outlines what will be built: the core features, scope, and any constraints or preferences decided during ideation.

For example, if the user wants a "personal scheduling assistant," the AI might brainstorm features ("Should it integrate with Google Calendar? Perhaps it could use natural language to add tasks.") and list a few

concept variations. The human's reactions guide the AI to focus on one direction. All these insights are recorded in **IDEA\_NOTE.md**. This way, when building starts, there's a clear reference of *what* the user really wants. This practice of explicitly documenting requirements mirrors how a product manager might write a concept note before development – it saves time by preventing misunderstandings <sup>1</sup>.

## Phase 2: Proof of Concept

**Purpose:** Test technical feasibility with minimal implementations or experiments.

**Human's Role:** Evaluate and refine the quick proofs. The human reviews small code snippets or demos, giving feedback ("This API call looks right" or "That output isn't what I meant"). The goal is to learn how the AI interprets the idea before investing in a full build. The human might also ask the AI to research unfamiliar concepts during this phase.

**AI's Role:** Now the AI becomes more **technical** – researching solutions and building tiny prototypes. It may invoke a *ResearchAgent* to gather information from documentation or literature, and a *BuilderAgent* to write simple scripts or diagrams. Crucially, the AI explains its reasoning and cites sources during this phase. It's essentially doing homework: finding known patterns and verifying that the chosen idea can be implemented as imagined <sup>2</sup>. If Phase 1 decided on a certain approach, Phase 2 might involve, say, writing a 20-line script to call an API or a quick UI mockup, just to prove the concept works.

**Key Output:** `POC_PLAN.md` – a plan describing the proof-of-concept results and the next steps toward a minimum viable product. This may include code snippets, research findings, and architectural notes. By the end of Phase 2, both human and AI should be confident that the core idea is technically achievable.

Special attention is given to the **ResearchAgent** here. A well-designed research prompt can save days of trial and error. For example, the ResearchAgent might be tasked with: *"Find the 3 most common libraries for building a scheduling app in Python, and list one pro and con of each. Consider that the current year is 2025."* The output would be a concise bullet list of findings, ideally with sources. In multi-agent frameworks like CrewAI, such tasks are explicitly defined. A sample `tasks.yaml` might specify a research task with an expected output like *"a list of 10 bullet points of the most relevant information about {topic}"*, ensuring the results are structured and on-topic <sup>7</sup>. By asking for a specific format (e.g. bullet points) and context (e.g. "given the current year is 2024" in the CrewAI example <sup>7</sup>), the AI is guided to produce **focused, up-to-date** research. This structured output can then feed directly into the POC plan. For instance, if a refactor is planned, the ResearchAgent could gather best practices and present them as a short report (in Markdown), which the human and BuilderAgent use to inform the refactoring plan.

## Phase 3: MVP (Minimum Viable Product)

**Purpose:** Deliver a functional, end-to-end prototype with basic features, complete with documentation.

**Human's Role:** Act as the project lead – giving targeted feedback, prioritizing features, and verifying that the prototype meets their needs. At this stage, the human might say "We'll skip login for now, focus on scheduling logic," or "The UI is fine, but the recommendations are off – let's adjust that." They help steer the final touches and ensure the product is usable.

**AI's Role:** Orchestrate the "full build" using all available help. The AI remains the main developer but can summon specialized sub-agents (personas) as needed for specific tasks. For example, a *PRDAGent* can draft a Product Requirements Document or user guide. A *QAAgent* can simulate tests and find bugs. If design polish is needed, a *VisualMapper* persona might be invoked to suggest a layout. The AI coordinates these roles to maintain consistency and integrate changes. It's similar to how large multi-LLM systems work by dividing tasks among expert agents <sup>8</sup>, except here it's one AI adopting different mindsets.

**Key Output:** A new folder (e.g. `sprint-01/`) containing the deliverables: `PRD.md` (product requirements and documentation for the MVP), `agents.md` (a record of which agents/personas were used and any domain-specific instructions for them), plus all code or content produced for the prototype.

During the MVP phase, the repository comes alive as a working prototype. The AI writes code, then possibly switches persona to review its own code for errors (as a QAAgent). It might consult the earlier phases' outputs – the profile (Phase 0), idea note (Phase 1), and POC plan (Phase 2) – to ensure it doesn't drift from the original intent. This phase may iterate (e.g. `sprint-02`, `sprint-03`) until the human is satisfied that the MVP meets the intent defined back in Phase 1. Notably, some advanced multi-agent frameworks have shown that having distinct roles can improve quality – e.g. MetaGPT, a research project, simulated a whole software team (PM, Engineer, QA) to build software step by step <sup>8</sup> <sup>9</sup>. Our approach is lighter weight (the same AI with role prompts), but it embraces that "*different skills excel at different tasks*". You wouldn't use a single tool for coding, designing, and testing in traditional dev; similarly, we allow the AI to compartmentalize its focus via these agent roles.

Each phase transitions to the next with minimal friction. The key linking mechanism is the YAML frontmatter in each phase's documents, which quietly carries forward context and decisions. For example, **IDEA\_NOTE.md** might start with a YAML block recording the chosen idea name, target users, and key features. When Phase 2 begins, the AI reads that frontmatter to immediately recall the context without needing the human to restate it. This is analogous to how certain AI frameworks persist conversation state or how prompt-chaining languages like PDL (Prompt Declaration Language) allow passing along state between steps <sup>10</sup> <sup>11</sup>. The outcome is a smooth continuum from alignment to MVP, where nothing important gets lost in transition.

## The Translation Hill Principle

**The clarity of an AI's output is proportional to the clarity of the human's self-description.**

This principle is the philosophical core of the project. It means that if you can clearly explain *how you think* and *what you want*, the AI should be able to build it. Phase 0 (Alignment) is therefore not optional or overhead – it's the foundation for everything. In fact, we consider the Phase 0 document a form of *calibration* for the AI. It's like zeroing an instrument before taking measurements. If the AI knows the user's "true north," it can navigate complex tasks with far fewer mistakes or re-dos.

Many AI failures can be traced to poor initial alignment – the AI started building with incorrect assumptions about what the user meant. The Translation Hill Project avoids that by investing in a rich mutual understanding upfront. The repository itself guides the user in expressing themselves. For example, rather than asking for sterile config values, a good Phase 0 prompt might say: *"Tell me about how you like to learn. Do you prefer metaphors and visual examples, or step-by-step logic? There's no wrong answer – this will help me adapt to you."* By inviting self-expression, we turn *feedback into a form of code*.

This concept finds support in industry insights. The creators of the Cline AI partner noted that gathering clear requirements and intent from humans is critical, because "*intent is the one thing that can't be automated*" <sup>2</sup>. They emphasize creating a "plan" (much like our Alignment phase) to capture human intent, and only then letting the AI loose. In other words, the first conversation with the AI determines all success

metrics that follow. If we get the user's intent and style right at the start, every subsequent phase can proceed with confidence. Conversely, skipping alignment is like building on a shaky foundation.

The Translation Hill Principle is also a reminder that explaining your idea clearly is *itself* the act of programming it. By the time you've described what you want in unambiguous terms, you've essentially written the most important part of the code – the rest is just implementation.

## The Agentic Ecosystem

To support the phased conversation, this project uses a cast of lightweight “agents” – not external programs, but conceptual roles that the AI can adopt. Each represents a recurring skill or perspective in creative development. Defining these roles explicitly helps structure the dialogue (and helps any human collaborator understand the context at a glance). Here are the agents and their typical functions:

- **InterpreterAgent** – Converts everyday human language into structured goals or tasks. (Phase 0, Alignment)
- **EmpathAgent** – Tunes the AI's responses to the user's emotional tone or stylistic preferences, ensuring the AI's communication stays comfortable and on-brand for the user. (Phase 0)
- **MuseAgent** – Generates divergent ideas, creative prompts, and unexpected associations to inspire the user. (Phase 1, Ideation)
- **ArchivistAgent** – Summarizes and organizes discussion points into coherent notes or documentation. (Phase 1)
- **ResearchAgent** – Explores external information: libraries, documentation, academic papers, Stack Overflow — anything to gather facts or patterns relevant to the project. (Phase 2, Proof of Concept)
- **BuilderAgent** – Writes code, creates diagrams, and builds prototypes. It focuses on tangible outputs, whether it's a snippet of code or a config file. (Phase 2)
- **PRDAgent** – Drafts product requirement documents, user stories, or success metrics to ensure the project's goals are well-defined. (Phase 3, MVP)
- **QAAgent** – Tests the code or prototype, identifies bugs or edge cases, and suggests fixes. (Phase 3)
- **HistorianAgent** – Reflects on the process and records lessons learned or key decisions for future reference (e.g., in a `CHANGELOG` or retrospective document). (Phase 3)
- **SummonerAgent** – A special orchestrator that can spawn new, specialized sub-agents on the fly in Phase 3. For example, if visual design becomes important, the Summoner can instantiate a *VisualMapper* agent for UI/UX advice, or a *DebuggerSpirit* to focus intensely on a tricky bug. (Phase 3 as needed)

These roles are declared in plain English (in Markdown or YAML) so that a human can always see who's “speaking” or what hat the AI is wearing at a given time. They are not separate processes – rather, think of them as modes or contexts for the AI's behavior. This concept is inspired by how some multi-agent systems define roles in config files. For instance, CrewAI uses an `agents.yaml` to assign each agent a **role**, **goal**, and **backstory** <sup>12</sup> <sup>13</sup>. In our repository, we might specify in `agents.md` something like:

```
- name: ResearchAgent
  description: "A seasoned researcher with a knack for finding relevant, up-to-date information."
  goal: "Provide concise, cited findings on any topic the project needs."
```

By outlining these personas, the AI is guided to maintain consistency when performing each function. It knows, for example, to respond more analytically and cite sources when acting as ResearchAgent, versus being more whimsical and open-ended as MuseAgent. This approach of role specialization is gaining traction. Projects like MetaGPT have shown that giving AI “team roles” (PM, Engineer, QA, etc.) can result in more organized outcomes <sup>8</sup>. Similarly, the open format **AGENTS.md** has emerged in the developer community as a way to provide AI coding agents with project-specific context and conventions <sup>14</sup> <sup>15</sup>. We embrace that spirit here: **agents are first-class citizens in the development process**, and we document their guidelines so that both humans and AIs know the rules of engagement.

In a practical sense, you will see the agent names in our conversation transcripts (or generated content) to denote perspective shifts. For example, a Phase 2 **POC\_PLAN.md** might include sections like:

```
**ResearchAgent**: (findings...)
**BuilderAgent**: (code snippet + explanation...)
```

This makes the multi-faceted process transparent. Any collaborator reading the logs or docs can follow along with the reasoning (almost like reading a play where each character’s lines are labeled).

## Repository Structure

The repository is organized to mirror the conversation flow. All content is meant to be human-readable (even the YAML). Technical metadata and context are stored in YAML frontmatter at the top of Markdown files, keeping the main text clean. Here’s the skeleton:

```
/project-root/
├── 00_start_here.md          # Intro guide for human users and tips on giving
                                feedback (Phase 0 kickoff)
├── human-profile/
│   ├── PROFILE.yaml          # Structured profile of the user (communication
                                style, preferences)
│   └── CONTEXT.md            # Additional context or background info the user
                                provided (e.g. project domain knowledge)
└── ideation/
    └── IDEA_NOTE_001.md        # Output of the first ideation session (Phase 1)
└── poc/
    └── POC_PLAN_001.md        # Plan and notes from proof-of-concept work (Phase
                                2)
└── mvp/
    └── sprint-01/
        ├── PRD.md              # Product Requirements Doc for MVP (Phase 3)
        ├── agents.md             # Description of agents/personas used in this
                                    sprint
        └── (code files, prototypes, etc.)
```

```
|   └─ sprint-02/ ...      # Subsequent sprints or iterations, if any  
└─ README.md (or similar)  # Overview of the project (could be this document)
```

Every file has a clear purpose and is written in plain English (or whichever natural language the user prefers). The idea is that a non-technical user can navigate this repo and understand the story of the project, while a technical user can see all the necessary specs and context to continue development or integrate with traditional code.

A few conventions we follow:

- **YAML Frontmatter:** Each Markdown starts with a `---` delimited YAML section capturing key info (e.g., phase number, date, participants, summary of decisions). This frontmatter travels with the document and can be programmatically read by tools or by the AI in subsequent phases to recall context. It's our way of giving the AI a "memory" between phases without relying solely on chat history.
- **AGENTS.md:** We include an `agents.md` in each sprint (or at root if agents are global) to outline any project-specific agent instructions. This is analogous to the community practice where `AGENTS.md` serves as a "README for AI agents" in many repos <sup>16</sup>. For example, if the project has a custom build step or coding style guideline, we'd put it in `agents.md` so the AI (and any human devs) can consistently follow it. This separation keeps the main `README` user-focused, while `agents.md` is free to include technical details, test commands, code style preferences, etc., specifically for AI guidance <sup>15</sup> <sup>17</sup>.
- **Nested Phases:** As the project grows, each major iteration can be captured. For instance, if after MVP the user has new ideas, one could spawn a new Phase 1 (ideation for version 2) in a new folder. The structure is extensible.

This structured approach is influenced by real efforts to merge long-term context with iterative development. For example, GitHub's **GPT-Engineer** tool generates a technical spec and code from a prompt, and it stores conversational steps and files so it can refine outputs over multiple runs <sup>18</sup>. Similarly, OpenAI's ChatGPT **Projects** feature (recently introduced) groups chats, files, and instructions together to "keep context together" for ongoing tasks <sup>19</sup>. The Translation Hill repo acts like a manual version of that concept: everything relevant to the AI's understanding of the project *lives in the repository*, making the collaboration stateful and transparent.

## Lessons Learned (from Pixel Detective Manifesto and Beyond)

This project builds on hard-won truths from prior experiments (including an internal "Pixel Detective" project). These guiding principles are also echoed by many experts in AI-assisted development:

1. **Context is Everything:** The more structured and rich your background context, the smarter the AI's output. Large Language Models are highly sensitive to context – relevant information needs to be provided clearly and at the right time. Studies have shown that if crucial details are buried or provided late, the model might ignore them <sup>20</sup>. So, front-load important context and keep it accessible (we do this via YAML frontmatter and docs). When an AI has the complete picture of what we're doing, it's far more likely to build something that fits the vision.
2. **Research Before Prompting:** A bit of analysis upfront saves a ton of iteration later. In other words, don't jump blindly into generation – first gather requirements, examples, and constraints. This is akin to how a good developer googles or reads docs before coding. Frameworks like the Plan-Act paradigm reinforce this: they have a dedicated planning phase where the AI and human ensure they have all the info and clear goals before any code is written <sup>2</sup>. In our case, Phase 1 (and sometimes

Phase 0) involves explicit research and note-taking. By the time we prompt the AI to build something, we've already answered many potential questions. The AI coding agent **Cline** emphasizes that defining a detailed plan (intent) before execution leads to much better alignment with user needs <sup>21</sup> <sup>22</sup>. We follow the same wisdom.

3. **Documentation Compounds:** Today's notes become tomorrow's design doc. Writing down decisions and insights as you go creates a compounding knowledge base. By the time you reach a complex phase (like MVP), you have a trail of reasoning to refer back to. This prevents "why did we do it this way?" confusion. In our project, every phase outputs a Markdown doc – essentially self-documentation of the process. In multi-agent exercises like MetaGPT, intermediate documents (e.g., a Product Requirements Document produced by a PM agent) are then used by subsequent agents (like an Architect agent) <sup>9</sup> <sup>23</sup>. The same compounding effect happens here: the Ideation notes inform the POC, the POC plan informs the MVP. New contributors or even the future you can read these docs to quickly understand the project's history and rationale, improving continuity.
4. **Specification Beats Vagueness:** Ambiguity is the enemy of reliable AI output. The more explicit and specific you can be about what you want, the more predictable the results. This means using checklists, examples, and well-defined success criteria in your prompts. User studies have found that vague prompts (e.g. "Tell me the best X") lead to meandering, trial-and-error conversations, whereas prompts that include format requests and context produce quicker, more relevant answers <sup>24</sup> <sup>25</sup>. Thus, we strive to turn fuzzy wishes into concrete specs. For instance, instead of asking the AI "build me a website," we would specify "a 3-page website with a contact form, using a modern CSS framework, output as a zip file." Our YAML files and PRD documents capture such specifics. The mantra is: *if something can be explicitly stated, do it*. Leave as little as possible to assumption. This not only helps the AI; it also clarifies the human's thinking.
5. **Multi-Model (or Multi-Agent) Orchestration Works:** No single tool or model is best at everything. Complex projects benefit from using different AI models or modules, each specialized for a role – or using one model in different modes (personas) for different tasks. We've designed the Translation Hill with an ensemble mindset: the MuseAgent might use a more creative model, whereas the BuilderAgent might use a code-focused model, etc., or one capable model guided by distinct role prompts. External evidence backs this up. For example, Prompt Declaration Language (PDL) allows chaining multiple LLM calls and even tool integrations in a single YAML spec <sup>26</sup>. This lets you do things like have one model summarize text, then pass it to another model for analysis <sup>10</sup> – leveraging each step's strengths. Likewise, projects like MetaGPT and CrewAI explicitly split tasks between specialized agents to simulate a team <sup>8</sup> <sup>27</sup>. In our repo, composition is key: by orchestrating various agents and possibly various models, we aim to produce results that a single monolithic prompt might not achieve. The sum is greater than the parts.
6. **Human Alignment Matters Most:** At the end of the day, all the fancy agents and tools mean nothing if we aren't building the *right* thing. "Right" is determined by the human's intent and values. Ensuring the AI is aligned with that from the start and continuously throughout is the top priority. A renowned AI researcher, Nick, recently wrote that the future of coding is "*humans doing what only humans can do: providing intent, meaning, and purpose, while AI handles the implementation*" <sup>28</sup>. This captures our ethos exactly. The human defines the why and what, the AI figures out the how. Our first phase (Alignment) and our constant emphasis on user feedback are there to safeguard this principle. If something feels off to the user at any point, we consider that a misalignment bug and

address it before proceeding. AI may be powerful, but it's not a mind-reader – the human's clarity and guidance remain the north star throughout the project.

These six lessons form a feedback loop in our methodology. By giving context, doing research, documenting, specifying, dividing tasks, and aligning to intent, we dramatically increase the chances of a successful human-AI collaboration. If a collaboration fails, usually one of these aspects was lacking.

## What This Repository Teaches

By walking through the structured approach in Translation Hill, a user (technical or not) can learn some important skills and concepts:

- **How to translate intuition into structure:** You'll practice taking a vague idea ("I want something that does X") and breaking it into clear, structured descriptions that an AI (or any collaborator) can act on. This is essentially *intent-oriented programming* – describing *what* you want in a human-friendly yet structured way, which bridges to *how* the machine will implement it <sup>29</sup> <sup>30</sup>.
- **How to iterate with AI instead of relying on one-off prompts:** Many people treat AI like a magic 8-ball – ask a single question and hope for the best. This project shows a better way: a continuous conversation where you refine requirements, get feedback, and build step by step. The AI becomes a development partner rather than a one-shot oracle. You'll see how to use each phase to progressively move closer to your goal, which is more effective for complex tasks.
- **How to organize thinking like a developer without writing code:** Non-programmers will gain exposure to the thought patterns developers use – breaking problems into phases, specifying requirements, testing assumptions – all through natural language. For example, writing an IDEA\_NOTE is analogous to writing a mini design doc. Writing a POC\_PLAN is like outlining a development sprint. You learn these practices by doing them in conversation, lowering the barrier to entry for software design thinking.
- **How to document the process so others (and future you) can learn from it:** Everything in the conversation gets captured in files, which means the project comes with its own tutorial/history. If you (or someone else) return to the project later, you won't have to guess why a certain decision was made – it's in the docs. This teaches the value of good documentation and how to achieve it with minimal extra effort (since the conversation *is* the documentation). It's a bit like having a built-in project diary that also doubles as user manual.
- **Ultimately, that explaining your idea clearly *is* programming it:** By the end of the process, users often have the revelation that the heavy lift was explaining and refining the idea, not typing out code. The coding feels almost trivial once the idea is crystal clear. This flips the script on traditional coding – the natural language conversation becomes the true source code, and the actual code is a compiled artifact of that conversation. It's a powerful mindset shift.

All of these skills contribute to making AI development more accessible. If you can hold a conversation, you can create software with AI. The repository acts as a scaffold to support you in that journey, ensuring no step is missed or glossed over.

## Vision

Our vision is to **make AI development accessible to anyone who can articulate their thoughts**, and to help technical folks collaborate more easily with non-technical domain experts. We think of dialogue as the

new form of coding – a two-way compiler where human ideas are “compiled” into machine solutions through conversation. This doesn’t mean traditional coding is obsolete, but it means the entry point can be much more humane.

In practical terms, we aim to:

- **Empower non-coders:** Provide a path for people with zero programming knowledge to build useful tools by conversing with an AI (with the repository guiding them on how to converse effectively).
- **Augment coders:** Encourage developers to adopt this framework when working with stakeholders or co-pilots. It can help gather requirements more clearly and avoid the back-and-forth of unclear tickets or chat transcripts lost in Slack. Imagine a software engineer pairing with a subject matter expert – the Translation Hill structure gives them a common playbook to follow.
- **Promote best practices in AI-assisted development:** As discussed, things like planning, documentation, and modular design are best practices in software engineering. We want those to naturally emerge in AI collaborations too, rather than people treating AI like a hacky script generator. By open-sourcing this approach, we hope more projects will adopt structured prompt engineering, leading to more reliable outcomes across the board.

Ultimately, the vision is a world where describing a problem clearly to an AI yields a solution as reliably as writing the code yourself (or more so). We’re not fully there yet, but every experiment like this brings us closer. By treating **conversation as creation** and **clarity as architecture**, we believe even complex software can be built in a collaborative, conversational way.

## Contributing

Contributions to this project won’t look like typical open-source contributions. We’re less interested in code and more interested in **process examples and stories**. The most valuable contribution is to try the methodology on a new idea and then share your experience. Did the phases work for you? What tweaks did you make? For technical users, perhaps you integrated our approach with an existing codebase – how did that go? For non-technical users, maybe you built your first app with AI – what did you learn?

We invite you to fork the repository and document your own human-AI collaboration journey. You can expand on agent definitions (maybe you needed a DataScientistAgent or a UXAgent – add it!), or refine the prompts in each phase. If you have improvements on how to link phases (say, a better format for the YAML context that Phase 3 can parse automatically), we’d love to see those. **Pull requests are welcome**, especially if they include actual transcripts or files from real usage.

We also welcome contributions that connect this project to other tools. For example, integrating with an existing multi-agent orchestrator or using PDL to formalize some of the YAML flows. The only rule is that any addition should preserve the ethos: human-friendly, transparent, and focused on bridging intent to execution.

Think of contributions not just as code, but as *knowledge*. Even just writing a short case study in the repo’s Wiki about how you applied Translation Hill in a certain domain (education, small business, game development, etc.) can help others see what’s possible. The more diverse stories we gather, the more robust this approach becomes.

## License

This project is released under an open and permissive license (e.g. MIT or Apache 2.0). We believe that good ideas for improving human-AI collaboration should spread quickly and freely. You are free to use, modify, fork, and distribute this project, even in commercial settings. We only ask that you give credit where due so that others know where the approach came from and can trace back the conversation (quite literally!) about how it evolved.

In short: **share it, remix it, make it your own.** The hill we're trying to flatten – the miscommunication between humans and AIs – is a common obstacle. Let's solve it together, in the open.

## Closing Thought

"If you can describe it clearly, you can build it with AI."

This is the mantra of the Translation Hill Project. By the time you've clearly expressed *what* you want and *how* you want it, the heavy lifting is done – the AI can take it from there. This project is where clarity becomes architecture, and conversation becomes creation.

---

### Sources:

- 
- 1 2 3 4 5 7 8 14 15 31 12 24 26 28
-

1 2 20 21 22 28 Why Human Intent Matters More as AI Capabilities Grow - Cline Blog

<https://cline.bot/blog/why-human-intent-matters-more-as-ai-capabilities-grow>

3 8 9 23 What is MetaGPT ? | IBM

<https://www.ibm.com/think/topics/metagpt>

4 5 6 Creating a Personalized AI Profile | by ArtfullyPrompt - Nathan Cash | . | Medium

<https://medium.com/aimonks/creating-a-personalized-ai-profile-2a8dc2ab8cc2>

7 12 13 27 31 Building A Multi-Agent System with CrewAI and BentoML

<https://www.bentoml.com/blog/building-a-multi-agent-system-with-crewai-and-bentoml>

10 11 26 Mastering Prompt Engineering with PDL: The YAML-Based Solution for LLM Development | by

Seemabanu | Medium

<https://medium.com/@seemabanu1610/mastering-prompt-engineering-with-pdl-the-yaml-based-solution-for-lm-development-bf9245c1cd4b>

14 15 16 17 AGENTS.md

<https://agents.md/>

18 Introducing `gpt-engineer` One prompt generates a codebase Asks ...

[https://www.reddit.com/r/aipromptprogramming/comments/14cs4gq/introducing\\_gptengineer\\_one\\_prompt\\_generates\\_a/](https://www.reddit.com/r/aipromptprogramming/comments/14cs4gq/introducing_gptengineer_one_prompt_generates_a/)

19 Projects in ChatGPT - OpenAI Help Center

<https://help.openai.com/en/articles/10169521-using-projects-in-chatgpt>

24 25 Prompt Structure in Conversations with Generative AI - NN/G

<https://www.nngroup.com/articles/ai-prompt-structure/>

29 30 Intent-Oriented Programming: Bridging Human Thought and AI Machine Execution | by Marco

Kotrotsos | Medium

<https://kotrotsos.medium.com/intent-oriented-programming-bridging-human-thought-and-ai-machine-execution-3a92373cc1b6>