

// phase 5 deep dive

Team &

Build



Assemble a small, reliable team, then build your design like an engineer, not a tutorial follower. This is a startup, so build it like one.

baswe.ai // become a software engineer

This is the in-depth guide to phase five of the Signal Project. Work through it once you have your system design and design document from phase four. By the end you will have a team and a working MVP, built like an engineer and ready to launch.

Why this phase is where it becomes real

Everything up to now has been thinking: choosing a target, researching it, picking a problem, and designing a system. This is where you actually build the thing. But how you build matters as much as what you build. You are not following a tutorial, you are executing a design you reasoned through, with a team, like a startup. The build is the proof that your design was real and that you can ship.

The output of this phase is a working MVP, built with intention and documented as you go, ready to be launched in the next phase.

You are building a mini startup, not a class project. Build it like one, document it like one, and treat your teammates like cofounders.

Part one: build your team

Do not build alone if you can help it. A small, efficient team lets you build something bigger and more impressive than you could solo, ships faster, and means your teammates land jobs off the same project too, which grows everyone's network. The rule is simple: the more efficient people you have, the bigger the project you can build.

▪ Why a team beats going solo

A team gives you bigger scope, faster shipping, complementary skills, and the accountability that keeps momentum going when motivation dips. It also multiplies reach: three or four people who all launch the same project and talk about it publicly create far more attention than one person alone. And building with others is closer to how you will actually work at a company, which is itself a signal. The one caveat: a team of unreliable people is worse than working solo. Quality of teammates beats quantity every time.

▪ Where to find teammates

The community. This is your best source. Everyone here is working toward the same goal, has complementary skills, and is motivated to build. There is a separate step-by-step SOP for exactly how to find and form a team inside the community, so use it.

Friends and former classmates. People you already know who can build, and who you already trust to follow through.

People you meet online. Developer communities, Discords, hackathons, meetups, and bootcamp cohorts are full of people looking to build something real.

Wherever they come from, prioritize people who are serious, reliable, and bring skills you do not have.

▪ What to look for in a teammate

Reliability and follow-through. This is the single most important trait. Skill without follow-through is useless. Someone who ships solid work on time is worth more than a genius who disappears.

Complementary skills. Fill your gaps: frontend, backend, AI, design, product. A team that covers the whole stack can build more.

A matching commitment level. Everyone needs to be in for the same intensity and the same one to two month window. Mismatched commitment breaks teams.

Genuine interest in the problem. People who care about the space do better work and stay engaged.

Good communication. You will coordinate constantly. People who communicate clearly and often make everything smoother.

▪ How big the team should be

Two to four people is the sweet spot for a one to two month MVP. That is big enough to build something substantial and cover the skills you need, and small enough to coordinate without heavy overhead. Solo is possible but harder. More than four or five people gets difficult to organize for a short, fast project.

▪ Set the team up right

Before you write any code, align on the essentials: the shared goal, which is that everyone wants to land a role and the project serves that; who owns what; the timeline and how often you will check in; the tools you will use, meaning one repo, one project board, and one communication channel; and clear expectations, including what happens if someone goes inactive. A little alignment upfront prevents most team failures. And make sure everyone understands they are both a builder and a launcher, because each person will post about this project, so each has real skin in the game. The community team-up SOP walks through all of this in detail.

Part two: build like an engineer, not a tutorial follower

This is the mindset that makes or breaks the project. You are not assembling a tutorial. You are executing the design you reasoned through in phase four, making intentional decisions, and documenting them. The entire point is to prove you can build something real that solves a real problem, which by definition cannot be copied from a step-by-step guide.

▪ The tutorial test

Here is a hard rule to hold yourself to. If you can find a tutorial that walks through building your exact project, your project is not good enough, and something went wrong earlier. The real problems that real companies have do not come with step-by-step tutorials, because if they did, they would already be solved. If your project is a build-a-chatbot or clone-this-app tutorial, go back to phase three and pick a better

problem. The absence of a tutorial for your project is not a problem, it is the proof that you are solving something real.

If there is a tutorial for it, it is not a Signal Project. The whole value is that no one has handed you the answer. You are figuring it out like an engineer, which is exactly what companies pay for.

To be clear, you will still use tutorials, documentation, and AI tools for individual pieces, like how to use a library or a specific technique. That is normal and expected. The test is about the project as a whole, not its parts. No tutorial should exist for the thing you are actually building.

▪ **Make every decision intentional and documented**

Extend the decision log from phase four into the build. As you build, you will hit decisions the design did not fully answer: a library choice, a data model, an API shape, how to handle an edge case, a tradeoff between speed and quality. For every meaningful one, make a deliberate choice and write it down, using the same format from the last phase: what you decided, the options, why, and the tradeoffs. This does three things at once. It forces good engineering, it keeps the team aligned, and it hands you the exact material you will use to talk about your work like an engineer, in your launch article and in interviews.

"I used this because a tutorial did" is a junior answer. "I chose this over that because of these constraints" is a top one percent answer. The build is where you generate those answers, so keep a running decision log or engineering journal as you go.

▪ **Treat it like a startup**

This is not a hobby project, and you should not build it like one, because you are going to launch it like a startup in the next phase, and companies will evaluate it like one. In practice that means a few things:

- ▶ Build for a real user, not for a grade. Every feature should serve the user and the problem.
- ▶ Ship an MVP, not a perfect product. Scope hard and get to something that works and shows value, fast.
- ▶ Document as you go: progress, decisions, and demos. This fuels your launch and shows momentum.
- ▶ Care about what startups care about: does it work, does it solve the problem, is it usable, would someone actually use it.
- ▶ Use real engineering practices: version control, a repo other people can read, clean-enough code, a clear README, tests where they matter, and a real deployment. Companies will look at your code.

▪ **Scope and ship in one to two months**

The timeline is a constraint, not a suggestion. Startups routinely ship a first version in a weekend, and you are building a focused MVP in one to two months. Cut

scope ruthlessly to the core of your design, build the smallest thing that proves the value, ship it, and only then improve it. Do not gold-plate. A shipped, rough MVP beats a perfect one that never gets finished.

▪ **How to work as a team during the build**

A simple, real workflow keeps a team productive:

- ▶ Break the design into tasks and assign owners on a shared board, using something like GitHub Projects, Linear, Trello, or Notion.
- ▶ Use one repository on GitHub, with branches and pull requests, and review each other's code. Code review catches bugs and is itself a signal.
- ▶ Hold short check-ins a few times a week to stay unblocked and aligned.
- ▶ Keep all communication in one channel, like Discord or Slack.
- ▶ Integrate your work early and often. Do not let branches diverge for weeks.
- ▶ Keep the design doc and decision log updated as the team's single source of truth.

▪ **Build the evidence as you go**

While you build, capture the raw material for your launch: screenshots, short demo clips, the problem and research story, the decisions you made, and any early results or metrics. Set up the open-source repository cleanly, with a README that explains the problem, your approach, and how to run it. Doing this as you go makes the launch easy and gives companies real engineering to look at.

Common mistakes to avoid

- ▶ **Building a project that has a tutorial.** This is the fatal one. If a guide exists for your exact project, go back and pick a real problem.
- ▶ **Going solo when you could build with a team.** Smaller impact, harder work, and far less reach at launch.
- ▶ **Picking teammates on skill alone.** Reliability matters more than raw talent. A flaky genius will sink the project.
- ▶ **No alignment upfront.** Skipping roles, timeline, and expectations is how teams fall apart. Set the ground rules first.
- ▶ **Following tutorials for the whole build.** Use them for pieces, never for the whole thing. Make intentional decisions.
- ▶ **Not documenting decisions.** You lose your interview ammunition and your engineering signal. Keep the log.
- ▶ **Gold-plating and never shipping.** Perfect is the enemy of shipped. Get the MVP out.
- ▶ **Building it like a hobby project.** A messy repo with no README and no user focus reads as junior. Build it like a startup.
- ▶ **Letting scope balloon.** Keep it inside the one to two month window by cutting, not adding.

What you should have before moving on

By the end of phase five, you should have:

- ▶ A team with clear roles and a timeline, or a deliberate decision to build solo.
- ▶ A working MVP that solves the problem from your brief.
- ▶ A clean, public repository with a README, because companies will read it.
- ▶ An updated design doc and a decision log capturing your intentional choices.
- ▶ The raw material for your launch: demos, screenshots, the story, and the decisions.
- ▶ Everyone on the team ready to launch and post about it.

That working MVP and the story behind it are exactly what the next phase turns into attention and interviews. Do not move on until it works and you can explain how you built it, decision by decision.

→ **With your MVP built and your team ready, move on to Phase 6: Launch.**