What if the harness mattered more than the model? - Aditya Bhargava, Etsy

32 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=2E9ANOOEN28
https://www.youtube.com/watch?v=2e9ANoOEn28

SUMMARY

Aditya Parikh discusses the importance of harnesses over models in AI development, advocating for a focus on building robust harnesses that can enhance the performance of open-source models. He introduces "Agency," a new programming language designed to facilitate the creation of agents, emphasizing the potential for local models to achieve high performance through well-constructed harnesses.

- *The tech industry often prioritizes sophisticated models, but this reliance can lead to dependency on proprietary solutions.*
- *A good harness can significantly improve the performance of weaker models, making AI development more accessible.*
- *Parikh introduces "Harness Bench," a benchmark demonstrating that different harnesses can yield varying performance results with the same model.*
- *He emphasizes the need for language-level support to build effective harnesses, leading to the creation of the Agency language.*
- *Key features of Agency include simple syntax for defining tools, safety mechanisms, and the ability to pause and resume execution.*
- *The development process involves iterative improvements: starting with basic models, adding tools, ensuring safety, and incorporating reasoning and optimization.*
- *Sub-agents allow for more complex tasks without overwhelming the main agent, improving efficiency and clarity.*
- *Self-optimization in Agency enables systematic performance improvements rather than trial-and-error methods.*

Hi everyone. My name is Audit, pronounced Audit like a tax audit. Um and I'm here to talk to you about what if the harness mattered more than the model?
A little bit about me first. I'm a staff engineer at Etsy. I'm also Etsy's IC initiative lead for agent of commerce. I wrote Grokking Algorithms, which is an illustrated book on algorithms.
Uh and I also do illustrated posts on

ducktyped.org.
Um
and I want to talk to you about
what if the harness mattered more than
the model?
By which I mean
what if we should be paying more
attention
to building expertise in harnesses?
Um
and my controversial take here is
I hear a lot of people say, "Oh, the
models are so good that you can just
keep the harness simple. Just give the
model a few tools and it'll do the
rest."
And I've been hearing this so often that
it has now
uh started becoming the wisdom of the
tech industry.
Uh
to which my answer is, "Yes,
but that's moving in the wrong direction
because that is making us reliant on
fancy proprietary models that can't be
run
locally.
So, what if we instead talked about the
other direction? What if we focused on
open-source models that can be run
locally?
What if we focused on building a harness
that is so good that we can get the
performance of a cutting-edge model
through a local open-source model?
So, what if the harness mattered
more than the model? What if our focus
should be on the harness?
Um how much does a harness matter?
Right? This is something I have been
researching a lot in my spare time.
Um

there's a really interesting paper that
talks about Harness Bench,
which is a benchmark for harnesses,
which is something we haven't seen a lot
of yet.
Um
it's got 106 tasks. Um
it's testing these different models in
different harnesses.
Uh but the point is
that it's running the same setup. It's
doing the same evaluation,
same model,
but testing different harnesses.
Um and the results are pretty
interesting.
So, scores range from 52.4%
to 76.2%.
So, more than a 20-point difference,
and only the harness changed.
Um and the really interesting thing is
that for weaker models, the harness
matters more.
Um if the harness matters more than the
model, that's great news.
Uh because if the model matters more,
then we're dependent on a handful of
companies who are able to build and
train these models.
Uh but if the if a good harness can
compensate, and can make a weaker model
perform better,
then you can build our own harnesses,
and that's something that any of us can
do,
and we don't have to depend on paid
models.
Um so, I have been spending a lot of
time
and doing a lot of research specifically
into
what would it take to build a really

good harness.
Um
and
I realized early on that none of the
existing tools or frameworks
of were able to do what I wanted. And I
realized that what was actually needed
was a language-level solution. So, this
is my other controversial take in this
talk is I think building a good harness
actually requires language-level
support.
Um and this is kind of where my research
has gone in the last 6 months and I've
spent
a
good part of that 6 months
building a new language called Agency,
which is a language for building agents.
Um
and just to talk about the terminology
there real quick because the terms agent
and harness
aren't
super crisp in the industry yet. They're
not super well defined. Um but in this
talk I'll kind of use them
interchangeably. But when I talk about
the agent, I really mean
the harness and the model.
So if you take Claude code for example,
Claude code is an agent. It uses Opus,
which is the model, and then everything
else is the harness.
Um
but you know, when we talk about Agency,
it's really for building
agents, but the thing it's providing is
the harness.
Um
Agency is really good. Uh you know, it's
still new.

Building a language is a lot of work and it's only 6 months in.
Uh but
I love this quote by Alan Kay. Simple things should be simple, complex things should be possible.
And that's something I love about Agency is that
I think you know, when people build agents, there are a lot of primitives, a lot of things that every agent will need to be able to do and those things should be simple to do.
Um and there's a lot of complex things that we want to be able to do with agents and it should be possible to do those things in the framework of your choice and with Agency it is possible.
Um
So now we have a clear goal, which is try to build the best possible harness.
And we have a great tool to do it, which is agency.
Uh so, the rest of this talk I'm going to talk about building a coding agent in agency.
And we'll see the same agent evolve over seven different examples.
We'll see the same model, the same task, but the harness will improve each time, and we'll see the harness improve the agent's performance each time.
Uh
So, you know, like I said, I've been doing a ton of research on this. My goals with this talk are to give you like the basics of building a harness.
When we talk about building a good harness, what does that mean? You know, you're starting from zero, what are the basics?

Um
I'll cover a good bit about agent safety
because, you know, when we talk about
building agents,
what you really want them to be able to
do is take action on our behalf. Um But
the big problem with that is how do you
get how do you give agent capabilities
so they can take actions,
but
not so much capability that they do
something unsafe. So, when we talk about
agent quality and capability, agent
safety goes hand in hand with that.
Um At the end, I'll do an introduction
to some of the more advanced concepts
like sub-agents and self-optimization.
I think that's where, you know, more
interesting things come in, and we
really start to talk about like
what are all the possibilities, you
know, because when you talk about
building a good harness, there are so
many things to explore.
And so, I hope you stick with me till
the end there because that's where we
get into some really interesting stuff.
Um so this is the code we're going to
ask our coding agent to edit.
This is a code to calculate the median
of an ordered
list of numbers.
Um
and if it's an odd length list, you just
take the middle element. If it's an even
length list,
you need to take the middle two elements
and take the mean of those two elements.
But right now this function doesn't do
that.
So there is a bug in this function.
There's a test

that shows the failure um
and we're going to ask our agent
to fix this code.
So let's go ahead and start. Let's just
dive in.
Um this is the first example. This is
agency code now.
And you'll notice that
agency looks a lot like TypeScript. It's
heavily inspired with by TypeScript with
some
Python syntax thrown in.
Um so the entry point is this node main.
I have a simple prompt. The Python
function median in demo/median.py
has a bug. Please fix the bug.
And I just passed that prompt into the
LLM function. I get the result and I
print the result.
Um
And of course this isn't going to work
because it can't read or write to this
file. So let's run this example
real quick.
Okay. So of course as expected it says
you know, in order to help me fix the
bug I'll need to see the existing code.
Um
so this is just the model. Very little
harness and of course it can't do
anything. So, the most obvious
improvement
to any harness is give it some tools.
So,
real quick, let's talk about how tools
work in the agency cuz it's very simple.
Um I'll define a function. In this case,
just for an example, I'm going to define
a greet function
that greets a user.
And then I'll make an LLM call that
says, "Use your greet tool to greet

audit." And then pass in that tool.
Um
and every function can automatically be used as a tool in agency. So, there is nothing else to do.
Uh agency creates a JSON schema out of this function and sends it to the LLM call.
The doc string here will get used as a tool description.
So, let's go ahead and run the second example.
So, NPX agency
Oops, actually.
NPX agency example since actually 01 cuz we're going to talk about tools first.
So, howdy audit?
And how can we know that it ran that tool and then just generate an example, right? So, I'll just also show real quick.
Um
agency writes out logs and has this great log viewer built in.
So, I'll just expand this last run and you can see here it's making this tool call to the greet function with these parameters. The tool responds by returning howdy audit, and the assistant just passes that straight back to us.
Um
So,
that's tools. Now, let's look at using the tools
in Agency. So, again, this is you know, the next step of our agent, which is give it tools.
Um
here is an example where we're giving the agent read and write tools.
Again, read and write are just functions

that are built into Agency, and because every function can be used as a tool, I can just pass this straight to the LLM.
Uh except
giving the agent the ability to read and write arbitrary files on our file system is really unsafe. So, by default Agency won't allow you to do this. Agency has a human-in-the-loop feature that would require some sort of approval.
We're not providing it anywhere, so this code will crash and print an error. So, let me show you what that looks like.
Great. So, interrupt standard read was not handled.
Your interrupt message, "Are you sure you want to read this file?" You can see the file name it's trying to read, and it's basically telling us that we didn't approve this interrupt. And it's giving us a pointer to the guide, right? Tells us how to use handlers.
Um
so
we started with just a model, we tried tools, and the next step is to make the tools safe. So, we're going to do exactly what that message asked us to do and use a handler.
So, now, here's the same code
that you know, passing through that read and write
uh those read and write functions as tools.
Um but now it's wrapped in a handle block.
Uh and it has a handler function down here.
So, now when those tools call or raise an interrupt,
this code is going to execute.
So, it'll

print the information about that
interrupt.
Um,
it'll ask the user for input using the
built-in input function. And if the user
approves,
the tool call is approved.
And I know I'm saying interrupt, and I'm
not really defining what that means.
Um, and I can get to that a little bit
later, but it's essentially a way to
pause execution at some point.
Um, and ask for human input.
And so, interrupts are a great feature
in agency, and all the functions in the
agency standard library
that
mutate code or mutate something, or have
some sort of destructive action, or read
sensitive data data,
all
throw an interrupt first.
So,
here's the safe version of our agent
where
the tools
will raise an interrupt by default,
and now this code is going to ask the
user, do you approve?
Let's run this example.
Okay, are you sure you want to read this
file?
I'll just say yes.
You can see it's reading. Now it's
trying to write to that file. So, you
know,
that's an example of
>> [snorts]
>> us
making the agent safer by asking for
user input.
And this is better, and it is safe, but

it's also very slow.
So, the next thing we want to do is make the agent autonomous,
while still keeping it safe, and that's the tricky part, right? How do you give it
just enough capability
so that you don't have to manually approve
every single action
without giving it without giving it so much capability that it's like able to do something destructive with your file system.
Um so agency has a really cool feature to handle this. It that's called partial function application.
>> [clears throat]
>> Um
it's a concept borrowed from functional programming. It's a really fancy term for a pretty simple concept.
So, here's a signature of the read tool, right?
Takes a file name and a directory.
Um and when you read a file, it's going to look inside that directory for that file name.
And with partial function application, what we do is we take that read function and call the partial on it.
And give it a directory. And what we're doing here is we're locking the directory argument to demo.
Um
the LLM isn't going to be able to change that argument.
It's not even going to know that that argument exists. When we pass in the read tool now, it's just going to see one parameter, which is file name.
It's going to pass in a file name.

And it will read that file from this directory.
So, we've locked the directory parameter. Now, the agent can only read files
from and write files to this directory.
Uh
partial function application is a really great way to constrain the capabilities of your agent. Now, no human input is needed, but it's still safe.
So, let's see that in So, again, run NPX agency
examples.
PFA is partial function application.
So, here
it's thinking and you can see
it read the file.
Didn't need to ask us for permission.
And now it said, you know, to fix this issue, it's kind of giving us a new code.
But it's not updating the code. It asks, "Would you like me to update the file with this corrected function?" But it didn't do that itself. So,
you know, this
version of this agent
is the best one we've seen so far because it does give us the correct solution
without any human input needed, but it hasn't actually fixed the code.
So, the next step is to get it to actually fix the code.
Um and that's where the feedback loop comes in.
Um so, there's a very popular agent pattern called react. And react stands for reason and act. And essentially, you're asking the agent to work in this loop. Where you reason,

act,
observe the consequences of your action,
and then decide what to do next.
So, in this case, reason, read the two
files,
act, run the tests,
observe, if the tests fail,
read the error,
and then reason again. So,
look at that test failure and think
about what you want to do next. And just
do this in a loop
until
the tests pass.
So, this is a really key pattern to
making the agent better.
And so, now let's see the same agent.
The only thing we're changing is
the prompt
and giving it the ability to run these
tests and now the agent should
work
better.
In this case, I'm going to also
print all the tool calls just so you can
see what's happening.
Um
Well, should maybe
All right, hang on.
It's always there.
There we go. Okay.
Now, let's see it in action.
Um
So, now
you can see it's calling the tools. It's
reading those two files, like we said.
It's running the tests.
The tests fail.
It's writing to the file. It's running
the test again
and confirming success. So, this is a
really important

stage in the harness development because
now
we finally have an agent that
successfully
figures out the task, checks
that the code is failing,
modifies the code, and checks that the
code now passes.
Um
So, let's talk about how we have
improved the harness so far.
So, we started with just the model,
couldn't do anything.
Then we added tools,
but the agent wasn't safe. Then we added
handlers, which added safety,
but needed human input.
Then we added partial function
application, which was safety without
requiring human input. And finally, the
last one that we just looked at was a
feedback loop, which added reasoning.
So, the agent made the change and
confirmed that the change worked.
So, I have two more examples.
This one we're going to go in a slightly
different direction. And this is where
we're getting into some of the more
advanced concepts.
So, this one is sub-agents.
Um everything so far we've done has made
the same task better. But with
sub-agents, we're asking, "How do you
make the agent do more things?"
Um
and let me show you
what this code looks like real quick.
Okay. So,
this program, I'll start by just showing
you the prompt we're giving it because
it's slightly different.
So, fix the failing test in test

median.py.
Then research Jensen's inequality for
medians with the Wikipedia agent and
explain it to me in the network form.
So,
here is a case where we're actually
asking
the agent to do a little more,
right?
And this one is interesting because
this is our main agent,
but for tools, we're not giving it the
read and write functions. Instead, we're
giving it two sub-agents as tools.
And
I really like this pattern in agency
because I've seen a lot of frameworks
make sub-agents their own concept and
make them kind of hard to grasp. Um in
agency, a sub-agent is just another
function. And you just call it, use it
like a tool just like you would anywhere
else.
And that consistency has a lot of
benefits. Makes it really easy to write
and reason about.
So, here's the coding agent
sub agent.
Um
same one as before. You know, there's a
system message.
This sub agent has its own LLM call,
gets its own set of tools.
>> [snorts]
>> Similarly, you have the Wikipedia agent
that has its own tool, which is the
search tool.
Uh which is a tool built into this
agency standard library that lets you
search Wikipedia.
Uh so, agency has a big standard library
that lets you do all kinds of things,

including this Wikipedia search tool.
I'm also
using the callback feature to print the
tool call information.
And now,
let's see this example in action. And
this is an example where
we're actually asking the agent to do a
little bit more.
And we're giving it
two sub agents to accomplish its task.
So, here you can see it's calling those
two sub agents.
Uh it's calling them in parallel, so
the Wikipedia agent is doing the search,
the coding agent is modifying the file,
and
that fixed the error, and now here's a
limerick.
So,
sub agents are cool because they let you
add new capabilities
without
bloating context.
Um
and this is a good pattern because it
allows your agent
when it has a lot of different tools at
its disposal,
it allows your agent to do
the right thing more consistently.
Often when agents get confused and fail,
it's because
they have too much context blow, but
also because they have too many
unrelated concepts in their context. And
they have too many tools that are
unrelated. And so they have a hard time
picking the right tool to use. In this
case, we make that problem a lot
cleaner.
For the top-level agent, we just say,

"Pick the sub-agent you want to call."
And then the sub-agents have those tools
that are kind of grouped in a way that
makes sense.
Um
So now, let's look at the last example,
which is self-optimization.
Uh Agency has a Japa optimizer built in.
It actually has a few optimizers built
in, but the Japa one
uh is a really interesting one.
And
the way
the optimization
feature works is you mark any variables
that you want the optimizer to optimize.
You can just use this optimize modifier.
And then you run the optimizer
and give it a goal.
So,
you know, in this case, you can see that
the two initial prompts I have
are very basic. I'm giving it hardly any
information.
And then the goal I give it is, "Fix the
bug in median.py
using TDD."
And
the reason this is such a great step
for our harness is
we're no longer doing trial and error.
We're no longer just trying stuff to see
what works. Instead, now we have a way
to systematically measure and improve
performance.
So, I'm just going to go ahead and run
this code.
Um I'm actually going to also reset.
Here, I'm going to
modify this code.
Cool.
So now

I won't
show the full run here because
the optimizer takes a while, but you can
see first it's like
checking what
prompts it has that it needs to
optimize. It's going to run the agent to
establish a baseline. So the baseline
objective is 0.2 and it's going to try
to improve upon that.
So in this case I did run the optimizer
earlier just so I could show you what a
full run looked like.
Um
and so here is one where you can see
it achieved the objective in the first
iteration, so it was actually quite
quick in this case.
Um and you can see that it's
rewritten
this prompt. Um
So the reason this self-optimization is
so great is because we're not guessing
and checking. We're systematically
measuring and improving,
which is a big leap forward.
So this is the ladder we just went
through for improving the harness.
Um
you know,
first step was nothing, just a model
can't even read the file. Then we added
tools,
but in a way that was unsafe and not
allowed in agency.
Next we added handlers,
which added safety, but
made things slower because they required
human input.
Then we added partial function
application, which was safe and fast.
Then we added reasoning, which gave us

higher confidence in the output of the agent.
Uh and then we switched tracks to subagents, added more capabilities, and finally we did self-optimization. So we got measured improvement and not guess and check.
So,
you know, I hope this gives you an idea of
how you can build harnesses and then slowly improve those harnesses. And I hope, you know, you try doing that and come along on this, you know, experimentation with me to see if we can build a better harness.
Um
because what of the harness matters more than the model. You know, what if we can build a harness that's good enough that we can now start using local models and still get the performance we need for the tasks we want to do.
So, what language level support does Agency provide? You know, we used Agency for all these examples. How did it help?
Um simple syntax for defining tools. Again, you know, the simple thing should be simple. Every agent is going to need tools. Let's make it easy to use them.
Tools for safety, very important for agents. So, interrupts handlers, PFA.
True pause and resume execution. I didn't cover this a ton cuz I didn't want to make this talk too long. Um but one of the really killer features of Agency is that
when you raise that interrupt, it's actually going to pause execution and return control back to the user. But in a way where you can resume execution later. Uh

and as far as I know,
very few languages allow you to do
something like this. And so, typically
most frameworks you'll use, if they have
a human in the loop feature, it has a
lot of restrictions. There's a lot of
things you need to do to work around
that. Agency is completely different. If
you wanted, you could, you know, raise
interrupts inside of a for loop, inside
of a tool call, inside of a sub-agent.
Um, and it would correctly
pause execution,
go back to the user, and then resume
execution at that exact point inside
that subagent, inside that tool call, at
that exact point in that for loop.
Um,
which is really cool. You can serialize
the execution, you can come back to it a
week later,
and you don't need to change anything
about all the code we have been looking
at. It just works, which is really cool.
Uh, it also has these built-in
optimizers, which are great, you know,
which kind of get you to a more
analytical, mathematical way of doing
the improvements. So, if you want to try
it out, please do. You can PM install
All you need is the package and a API
key, and you're set.
Um,
main takeaways, you know, an agent is a
model plus a harness for this talk, at
least. Uh, better harness equals better
performance, especially for weaker
models.
And then, how do you build a better
harness? Give it tools, make it safe,
make it autonomous, make it reason,
give it subagents, and have it

self-optimize.
Thank you so much. Again, my name is Aditya Parikh, check out agency agencylang.com,
uh, and then please follow me on Blue Sky, if you wish. Thank you. Take care.