# Tell me about yourself

I am a software engineer with 5+ years of experience leading large-scale, reliability-focused engineering efforts at major organizations, including **Microsoft, Capital One,** and **Ontario Health**. My work bridges **Site Reliability Engineering (SRE)**, **distributed systems**, and **AI-driven automation**.

At **Microsoft**, I was part of the **Edge engineering systems team**, responsible for the **distributed test resolver platform** used by 800+ developers to run **millions of tests daily**. I led key SRE initiatives:

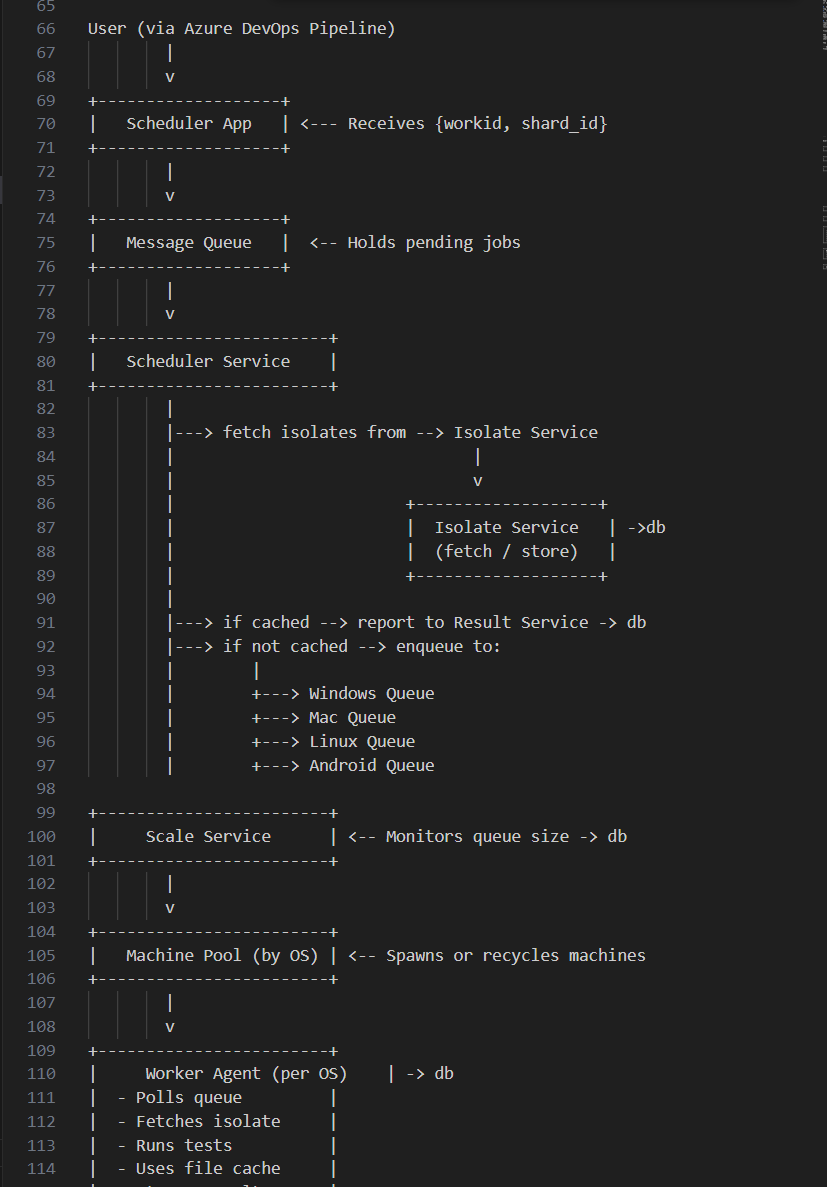
* Implemented an **auto-scaling scheme**, enabling rapid expansion to **2,000 resolver machines within 5 minutes** of demand spikes. Previously it was scheduled scaling
* Developed an **auto-recycler** to maximize resource utilization and reduce waste by adding a buffer time to device restart and intelligently detect if the machine will receive similar jobs
* Independently built a **distributed temporal fault localizer**, leveraging early OpenAI Codex to detect and isolate faults — much earlier than today’s AI-driven fault localizer. It detects cosine similarity of upstream commits and downstream commits, in git bisects quickly point devs to where a potential problem is
* To maintain visibility and system health, I leveraged **Azure Monitor** for infrastructure metrics, **Application Insights** for service-level observability, and **Kusto (Azure Data Explorer)** for large-scale log analytics and diagnostics.

At **Capital One**, I led a migration project for the **secured card division**, transitioning critical workloads from a **monolithic system to a resilient, serverless microservices architecture** on **AWS**. This improved the platform’s **scalability, fault isolation, and maintainability**, following best practices for reliability engineering.

Earlier in my career, I served as **lead frontend engineer** at **Ontario Health**, developing the province’s **COVID-19 test result portal** under heavy public load, ensuring **high availability and reliability** during a critical time.

Across these roles, I’ve built deep expertise in **.NET, C#, Java, Azure, AWS**, and using **AI technologies to enhance reliability, observability, and developer productivity**.

Beyond industry work, I have **two IEEE publications on autonomous vehicle systems (60+ citations)**, and I am currently researching **LLM-based semantic change point detection** in software history to improve fault localization and software reliability.



**Built and maintained the distributed test resolver** that scaled test execution across ~2,000 machines, supporting a global developer base of 800 engineers running millions of test jobs daily.

**Implemented intelligent autoscaling**, enabling the system to spin up the full 2,000-machine test pool within 5 minutes in response to demand surges.

**Introduced auto-recycling of machines**, allowing shared compute resources to be gracefully handed off before teardown, improving both resource utilization and test job continuity.

**Improved test routing efficiency**, ensuring test jobs were dynamically assigned to the most relevant machines (geographically or config-wise), minimizing wait time and maximizing throughput.

**Developed a distributed temporal fault localizer** that leveraged GPT-3.5 to identify when and where regressions were introduced — well ahead of ADR agents and papers like SWE-agent.

**Pioneered AI integration into engineering infra**, applying LLMs to analyze test logs, triage failures, and infer behavioral shifts in test flakiness over commit history.

**Led reliability initiatives** that reduced test flakiness and improved pass rates through automatic quarantining, rerun throttling, and flaky test detection patterns.

**Worked across the full Microsoft stack** (.NET, C#, Azure) and often collaborated with infra and telemetry teams to ensure visibility and traceability of test failures.

# Shorter Tell me about yourself

My name is Eugene Wang. I have 5 years of experience working in major American companies like Microsoft, Capital One, and in the public sector, often in lead engineering roles.

At Microsoft, I worked on the engineering systems team for Microsoft Edge, where I built and maintained a distributed test resolver used by over 800 engineers running millions of test jobs daily. I also developed an internal fault localizer using OpenAI Codex — well ahead of today’s LLM-powered debugging tools.

Before that, I led a cloud migration project at Capital One, moving secured card systems from a monolith to a fully serverless microservice architecture using Java and AWS. I also led the frontend for Ontario’s COVID-19 test result portal in 2020.

Alongside industry work, I’ve published two IEEE papers on autonomous systems and am currently researching how LLMs can detect subtle semantic changes in code history to improve debugging.

# Salary Expectation + address + work auth + gap

$115,000.00 - $260,000.00: $200,000

Address:

1149 N 85th St

Work Authorization

I currently hold a valid TN visa and am legally authorized to work in the U.S. I do not require visa sponsorship and can easily transfer my employment.

Career Gap

I underwent surgery in September 2023 and faced a longer-than-expected recovery, requiring extended treatment into early 2024. As a result, I stepped away from my previous role to fully focus on my health. I’ve made a full recovery as of February this year. During that time, I also worked on a research publication exploring the use of a custom-trained LLM model to bisect Git history and detect semantic code changes.

# Role you are Looking For

I’m looking for a senior engineering role where I can take ownership across the entire lifecycle—from architecture and design to implementation and optimization. I'm especially drawn to roles where I can apply creativity to solve complex, open-ended problems—whether that’s building platform capabilities from scratch or improving performance and scalability through innovative approaches.

# Communication Style

I prefer asynchronous communication, where one articulates his or her ideas clearly in docs, sent out to stakeholders, who can then understand and go through it at their own time and bring their opinions to the the room. If one wants to initiate a sync session, he or she could just hop on the chat room, want asks others to do so, so the threshold for a sync session is much lower than scheduling meetings. Only when working with external teams or stakeholders, do we formally schedule.

Work Environment:  
I work best in a team where the leader has a clear vision and meaningful goals, and where our day-to-day work ties directly back to that bigger picture. I like a team culture where people naturally help each other out — where we’re teammates, not competitors. It’s also important to me that everyone has a good sense of what others are working on and what’s going on across the company, so we stay connected and not just heads-down in our own tasks. I really appreciate leaders who are supportive and who make sure the team is aligned with leadership and the company’s direction.

# Why Geico:

1. I am a Geico customer, I have switched around for my car insurance from farm to progressive and eventually to Geico, and I have not made a switch since. It is the lowest price yet reputable insurance I can find on the market. I did have 1 collision and there was no pain no hassle with Geico, so I feel like it is an extremely well ran company. Because of this I am curious about the inner workings of Geico.
2. I aim to continue advancing my skills as a software engineer while gaining more experience solving designing and building customer facing system and exploring new stuff
3. I believe the greatest opportunities today lie in applying AI to bridge the last-mile gaps in business operations. Industries like government, insurance, and banking has more scenario that could be solved, and I see significant potential in helping these sectors modernize and unlock greater value.

# Working with a difficult person

**Situation:**

While working on the **Secure Card Migration project at Capital One**, I was responsible for designing the APIs and internal transaction flow, based on a high-level architecture blueprint. The goal was to record transactions, apply credits, and forward them to the scoring team.

**Task:**

Although the system requirements only specified end-of-day delivery for transactions, I proactively explored different methods for sending them. I experimented with **real-time transmission** to improve performance, but it introduced delays, so I reverted to **batch processing**. Despite the rollback, my experimentation was seen as “unpredictable,” and the team lead expressed concerns about **trust in my work**.

**Action:**

I recognized that constant stakeholder meetings for every technical iteration would be impractical. However, I also understood the need for **better visibility** into my decisions. To address this, I implemented a **lightweight communication process**:

* For any design or implementation changes, I would post a **simple heads-up in our shared group chat**, e.g.,  
  *“Planning to update transaction sending method to X”*.
* This kept the team informed without creating bottlenecks, and because it was framed as an update, not a request for approval, it preserved agility while respecting transparency.

**Result:**

After adopting this approach, feedback about my “unpredictability” ceased. Technical debates still happened, but they shifted to **proactive discussions before changes were made**, which **improved collaboration** and **restored trust**. This small but effective adjustment allowed me to maintain **technical ownership** while aligning better with the team’s communication expectations.

**What this shows about handling difficult people:**

* **Emotional intelligence**: You identified the real pain point (lack of visibility) behind the trust issue.
* **Proactive adaptation**: You created a solution that respected both your workflow and the team lead’s concerns.
* **Diplomatic conflict resolution**: You avoided escalation and instead built a process that turned a difficult relationship into smoother collaboration.
* **Leadership**: You led by example, managing up and driving alignment without needing formal authority.

# Problem-Solving & Critical Thinking

* Chromium pump pumps code in
* Very hard to collocate which upstream change caused weird downstream behavior
* So I looked into OPENAI codex, and built a semantics analyzer,
* Greatly increased the productivity of pump merge resolver.

# How do you structure your day?

* Label things by mental capita: happy, neutral, burdensome
* Try to sandwich things at different mental capacity
* Check communications early in the day, then before lunch, then after lunch then afternoon break, then summarize the day, move update project status end of the day.
* Arrange things in

# Example: Meeting a Critical Deadline

Biden's Executive order on Guarantee AI safety, so we switched to doing security ops, detect easiest lowest hanging fruit, configuring security certificate. quickly switched.

**🧩 Suggested Story Arrangements**

**1. Ownership + Bias for Action + Deliver Results**

### **Story**: When you implemented an incremental update system for **cloud VM pools to reduce outage risk**.

* **S**: Large-scale update to agent pool risked global outage.
* **T**: Build a safer, partial rollout system that reports failures early.
* **A**: Designed system to test updates in waves; monitored logs; reported metrics in real time.
* **R**: Reduced failure fallout to <5% of affected machines. Became a best practice adopted org-wide.

**2. Invent and Simplify + Insist on Highest Standards**

### Story: LLM-powered Git bisect tool for error localization.

* **S**: Traditional Git bisect too slow and manual for identifying faulty commits.
* **T**: Accelerate debugging via automation and ML.
* **A**: Built tool integrating LLMs and bisect metadata; showed confidence estimation.
* **R**: Cut debugging time from hours to minutes; shared internally as an engineering showcase.

**3. Dive Deep + Are Right, A Lot**

### Story: Diagnosed high-latency in Azure DevOps pipeline affecting swarm deployments.

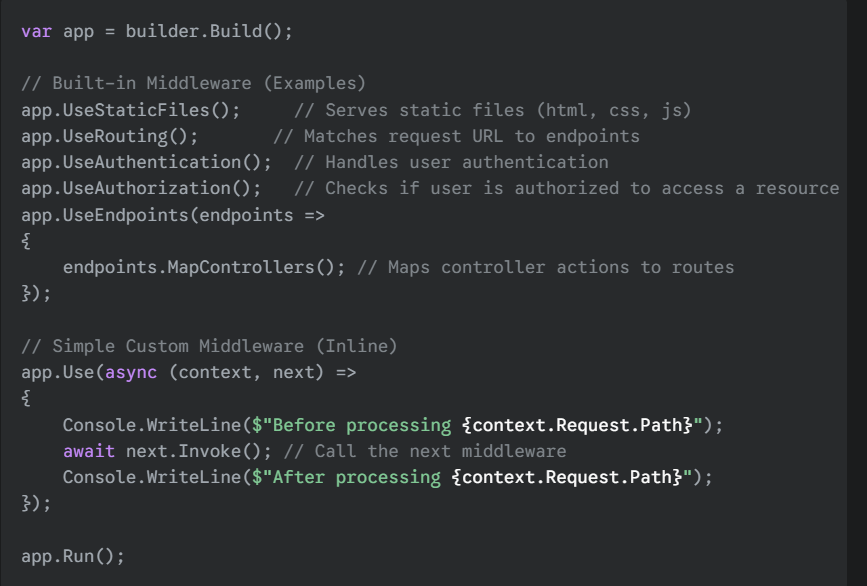
* S: Our Azure DevOps pipeline for Swarm deployments experienced sudden latency spikes, blocking team velocity and delaying critical updates.
* T: Identify the root cause across a complex chain of templates, VM setup, and service calls.
* A: I instrumented key pipeline stages, analyzed logs, and traced the delay to a degraded test provisioning service that all deployments were waiting on unnecessarily for a removed nested vm stage.
* R: After decoupling the dependency, deployment time improved by to 3 hours not 6 restoring velocity and stability.

**4. Customer Obsession + Think Big**

### Story: Covid Test Viewer

* **Story**: Designing a medicine web app that uses LLMs to break down components and suggest alternatives.
* **Situation**: During the peak of COVID, users urgently needed reliable and user-friendly access to their test results. However, the existing legacy application—built in PHP intended for practitioner—was fragile, difficult to update, and dependent on an external contractor no longer available.
* **Task**: Modernize the frontend experience to make test data easily accessible, while preserving the backend and avoiding major disruptions or costly third-party involvement. Basically created a middle ware that pulls everyday for new test results, and then batch that to our own database
* **Action**: I led the design and development of a new frontend using Angular. The application securely interfaced with the existing backend database to retrieve and present COVID test results in a clear, accessible format. I ensured backward compatibility and handled all deployment and integration tasks internally.
* **Result**: The new system was successfully built and deployed in-house, removing reliance on external contractors and significantly improving both user satisfaction and system maintainability. The solution was delivered on a tight timeline and supported thousands of users during a critical public health window.

### DOTNET APP



**Application Entry Point:** The application starts in Program.cs, which is responsible for building the host and configuring the fundamental services and the request processing pipeline (for web apps).

**Dependency Injection (DI):** This built-in pattern manages the creation and provision of service dependencies, promoting loosely coupled and testable code by injecting required objects rather than having classes create them directly.

**Configuration:** A flexible system allows applications to load settings from diverse sources like JSON files and environment variables, enabling easy adaptation to different environments without code changes.

**Logging:** Provides a standardized way to record information, warnings, and errors from within the application, which is crucial for monitoring, debugging, and understanding application behavior.

# Questions to ask

- What is the nature of the project? What is the goal, and how do you define success?

- What opportunity do I have for this role?

- What is the most urgently problem that needs to be solved right now?

- How do you maintain the quality of service?

- How does a new project start look like?

- How do engineers communicate.

- If an engineer has blocker, what steps he or she should follow

# Initial Recruiter Screen

* Brk
* Ai
* Senior in teams
* Staff
* Team = 30 size
* Smaller teams
* Seattle, WA

Working on:

1. Modernizing policy system: c# to java, auto and home : home chatbot, mobile
2. Inquire policy + line of communications
3. Partner with business side + they will be communicating with business, alongside other engineers IC
4. Exposure?
5. New team? Always in place, some of the initiatives communications subteams inherited
6. What is the teams and role: senior engineer C# react microservice

* 30 minutes hm: high level technically background
* Virtual onsite 4 rounds
* 2 rounds to start with coding
* 2 final rounds system

Looking for

* OOF Friday and Monday

Service engineering

* Insurance system
* Modernization based on modern tech
* Better tools and better patterns

All cloud to hybrid: modernization effort: more java java framework, balancing in dotnet core. Ai enable backend system.

Lots of innovation